Towards Secure Machine Learning Acceleration: Threats and Defenses Across Algorithms, Architecture, and Circuits

by

Kyungmi Lee

B.S., Seoul National University, 2018
S.M., Massachusetts Institute of Technology, 2020

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Kyungmi Lee Department of Electrical Engineering and Computer Science May 17, 2024
Certified by:	Anantha P. Chandrakasan Vannevar Bush Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Studies

Towards Secure Machine Learning Acceleration: Threats and Defenses Across Algorithms, Architecture, and Circuits

by

Kyungmi Lee

Submitted to the Department of Electrical Engineering and Computer Science on May 17, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

As deep neural networks (DNNs) are widely adopted for high-stakes applications that process sensitive private data and make critical decisions, security concerns about user data and DNN models are growing. In particular, hardware-level vulnerabilities can be exploited to undermine the confidentiality and integrity required for those applications. However, conventional hardware designs for DNN acceleration largely focus on improving the throughput, energy-efficiency, and area-efficiency, while the hardware-level security solutions are significantly less well understood.

This thesis investigates the memory security for DNN accelerators, where the off-chip main memory cannot be trusted. The first part of this thesis illustrates the vulnerability of sparse DNNs to fault injections on their model parameters. It presents SparseBFA, an algorithm to identify the most vulnerable bits among the model parameters of a sparse DNN. SparseBFA shows that a victim DNN is highly susceptible to a few bit flips in the coordinates of sparse weight matrices, less than 0.00005% of the total memory footprint for its parameters.

Second, this thesis proposes SecureLoop, a design space exploration framework for secure DNN accelerators that support a trusted execution environment (TEE). Cryptographic operations are tightly coupled with the data movement pattern in secure DNN accelerators, complicating the mapping of DNN workloads. SecureLoop addresses this mapping challenge by using an analytical model to describe the impact of authentication block assignments and a simulated annealing algorithm to perform cross-layer optimizations. The optimal mapping identified by SecureLoop is up to 33% faster and 50% better in energy-delay product compared to conventional mapping algorithms.

Finally, this thesis demonstrates the implementation of a secure DNN accelerator targeting resource-constrained edge and mobile devices. This design addresses the implementation-level challenges of supporting a TEE and achieves a low overhead of less than 4% performance slowdown, 16.5% more energy consumption per each multiply-and-accumulate operation, and 8.1% of the accelerator area.

Thesis supervisor: Anantha P. Chandrakasan
Title: Vannevar Bush Professor of Electrical Engineering and Computer Science

Acknowledgments

I am truly grateful to the many people who supported me throughout my Ph.D. journey.

First and foremost, I would like to thank my advisor, Prof. Anantha P. Chandrakasan, for all his advice and support. Prof. Chandrakasan introduced me to the exciting research area of security and machine learning, encouraged me to work on important research problems, helped me tackle challenges in the research process, and taught me to present my research effectively. I thank Prof. Chandrakasan for being an amazing advisor and mentor and always being patient and nice. I was fortunate to join Prof. Chandrakasan's research group six years ago, which provided a collaborative and supportive environment throughout my Ph.D. journey.

Next, I am very grateful to Prof. Mengjia Yan and Prof. Joel S. Emer for being on my thesis committee and also being amazing collaborators. They provided valuable discussions for the second contribution of this thesis and provided detailed feedback on manuscripts and presentations. I really appreciate Prof. Yan and Prof. Emer for taking their time and sharing even more feedback and discussions for my thesis. I also thank Prof. Yan for being on my RQE committee and providing many insights on hardware security. I learned a lot from Prof. Emer on hardware architecture for deep learning as well, as a student for his course and later as a teaching assistant.

I would like to acknowledge the Korea Foundation for Advanced Studies, MIT Jacobs Presidential Fellowship, and Siebel Scholars Program for financial support. The projects in this thesis are sponsored by Samsung Electronics and Facebook in part, and the TSMC University Shuttle Program supported the chip fabrication. I thank all these sponsors for generously supporting my research.

Moreover, I thank Prof. Vivienne Sze for all her help. I took two of her courses on digital circuits and hardware architecture for deep learning, both of which provided a thorough understanding of hardware design and machine learning. I thank her for giving me an opportunity to be a teaching assistant for the hardware architecture for deep learning course. Also, I thank Prof. Sze for being my RQE committee chair and taking the time to provide feedback on my research.

I would also like to thank my graduate counselor, Prof. Jae S. Lim, for guiding me through the graduate program and following up on my progress.

I thank Dr. Saurav Maji, Dr. Donghyeon Han, and Gaurab Das for being my collaborators. Thanks to Dr. Maji, I could explore the physical side-channel and fault-injection vulnerabilities of a DNN accelerator. Dr. Han and Gaurab provided valuable discussions and help in implementing a secure DNN accelerator supporting a trusted execution environment in silicon. Also, I thank Dr. Alex Ji for the digital process flow setup and sharing resources for the chip design and FPGA implementations.

I am grateful to have had amazing labmates throughout the past six years. My labmates provided an intellectually motivating and friendly environment. I thank my past and present labmates, Preet, Sirma, Wanyeong, Utsav, Mohamed, Alex, Taehoon, Miaorong, Saurav, Aya, Ray, Rishabh, Vipasha, Jongchan, So-yoon, Kaustav, Maitreyi, Yeseul, Eunseok, Deniz, Zoey, Saebyeok, Adam, Patricia, Mingran, Dimple, Mohith, Donghyeon, Sarina, Jaehong, Seoyoon, Aylin, Jaeyoung, Gloria, Joanna, Haripriya, Young-hwa, Burcin, and Vincent.

I thank our past and present lab administrators, Margaret Flaherty, Jessie-Leigh Thomas, and Katey Provost for all their support with logistics and administrative work. I also thank Yuvie Cjapi for scheduling all meetings and my defense. Furthermore, Prof. Leslie Kolodziejski, Janet Fisher, Alicia Duarte, Meredith Bittrich, and other staff of the EECS graduate office provided immense support for my Ph.D. program.

I thank all my friends who made my Ph.D. experience more enjoyable. In particular, I would like to thank my partner, Gilhyun Ryou, for being my best friend throughout my Ph.D. journey.

Finally, I am really grateful to my parents for their unconditional love and support throughout my life. They taught me important life lessons and made me who I am today. Thank you so much for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have witnessed a rapid growth in artificial intelligence (AI), driven by successes of deep learning. AI is widely adopted for chat bots [1], virtual assistance for computer programming [2], search engines [3], image and video generation [4], design automation tools for semiconductor chip designs [5], autonomous driving [6], and drug discovery [7]. Following this widespread adoption of AI in consumer products and critical applications, ensuring the security of AI is becoming increasingly important. The goal of this thesis is to understand the security vulnerabilities of AI applications across the technology stack of algorithms, hardware architecture, and circuits, and to provide a defense solution without significant overhead to the system. In this chapter, we outline the necessity for secure hardware acceleration for AI applications and the key contributions of this thesis.

1.1 Motivation

1.1.1 Necessity for Security in AI Applications

Security has three important properties: confidentiality, integrity, and availability [8]. In this section, we illustrate how each of these three pillars has direct connections with AI applications.

First, confidentiality requires that the data should be kept private and not be accessed by unauthorized users. In the context of AI, confidentiality is crucial for applications handling sensitive user information, such as medical imaging for diagnosing diseases [9]. The user input data, which is the private health information of patients in those applications, should not be disclosed and can be protected under the regulations (e.g., the Health Insurance Portability and Accountability Act (HIPAA) [10]). This user data privacy can become even more important in the future, in light of recent regulations like the European Union's General Data Protection Regulation (GDPR) [11]. Furthermore, confidentiality is also applicable to the model parameters of deep neural networks (DNNs), which are core algorithms in many AI applications. Training a DNN can require large proprietary datasets and resources (e.g., it is claimed that the training process of GPT-4, a popular large language model, costs more than $100 million [12]; training large foundation models is becoming more expensive [13]), and the leak of model parameters can be detrimental to intellectual property rights of AI service providers.

Second, integrity stipulates that the data should not be improperly perturbed. Integrity is crucial for AI to deliver trustworthy decisions to its users, by guaranteeing that the data user requested for processing and the model parameters are authentic. Recent work demonstrated the vulnerability of DNNs to small adversarial perturbations both in their input data [14]–[16] and their model parameters [17], [18], and to soft errors arising from reliability issues [19]. For example, a few bit flips in the model parameters of a DNN can result in the complete degradation of performance [17], [18] or wrong classification to a particular output desired by an adversary [20]. Therefore, ensuring that both the data and the model parameters are unperturbed and intact is important.

Finally, availability means that the data should be reliably available to valid and authorized requests. For AI applications, availability can be undermined when an adversary wages denial-of-service attacks, which can prevent valid users from accessing the applications. Such a scenario can be dangerous for applications with real-time constraints.

These security concerns often prevent the adoption of AI applications. For example, the United States House of Representatives banned using Microsoft Copilot [21] due to data confidentiality issues. Security concerns are also manifest in a recent industry survey. In one recent report [22], security was the top concern for the infrastructure hosting AI applications, ahead of cost, for industry respondents.

Therefore, securing AI applications from potential sources of confidentiality, integrity, and availability breaches is necessary for AI to be adopted for important missions and for enhancing public trust.

1.1.2 Importance of Hardware Security

Security vulnerabilities can exist across multiple levels of the technology stack supporting AI. Among them, hardware architecture and physical layer vulnerabilities are particularly pernicious. First, unlike software-level vulnerabilities that can be readily fixed, hardware-level vulnerabilities are difficult to patch after the fabrication of the physical hardware.

Second, hardware-level vulnerabilities are often not a result of flaws in architecture or implementations. For example, micro-architectural side-channels arise from the sharing of resources (e.g., caches), which is intended by design and not a flaw in itself [23], [24]. Many physical-layer side-channels exploit the inherent characteristics of electronic circuits, such as the dependency of power consumption on the data [25]–[27]. In memory elements, such as DRAM, vulnerabilities are tied to the physical properties of the memory cells, including the data remanence property [28] and the interference between closely positioned cells [29]. Thus, we cannot easily remove the source of hardware-level vulnerabilities to provide a defense, as they are inherent or intended features.

Finally, providing a defense to hardware-level vulnerabilities can be “costly”. Typically, adding support for security trade-offs performance (e.g., applications will run slower), energy (e.g., in battery-operated devices, shorter battery life), and area (e.g., the cost for fabrication increases). However, these trade-offs can be intolerable to users and applications. For example, performance slowdown can be critical for applications running in real-time especially if the resulting latency exceeds the real-time constraint. Also, the willingness to accept performance loss gets significantly more expensive for a larger amount of slowdown (e.g., 30%) [30]. For mobile and edge devices, area and energy overhead can be concerning since they are battery-operated and sometimes target low-end cost-sensitive markets. Therefore, while security is critical in AI applications, support for security has to be *economic*, making the problem more challenging.

1.1.3 Necessity for Secure Hardware Accelerator for AI

Securing AI applications against hardware-level vulnerabilities can be done in many ways. However, we aim for a small hardware root of trust for providing a defense. For the hardware accelerators for DNNs, our goal is to reduce the root of trust to the accelerators themselves, instead of relying on other third-party hardware or software stacks.

First, we note that relying on third-party hardware for security has several downsides. Extending the root of trust to many hardware components manufactured by different vendors has a composability problem. Several hardware components should be coordinated well to guarantee security without exposing a new attack surface, which can be difficult. Also, memory components like DRAM generally lack computing capability, and engineering memory components to support security can be challenging. For example, with some exceptions for recent High Bandwidth Memory (HBM) and 2.5/3D integration, general DRAM lacks compute capacity and cannot perform complex operations required for supporting security. Therefore, reducing the root of trust to the processor itself and removing dependencies on third-party hardware components is beneficial for security and practical implementation.

Another benefit of a small hardware root of trust is that it does not rely on software stacks, and does not require algorithms to be modified to support security. For example, the training process for DNNs can be modified such that DNNs can recover from faults or detect faults using algorithms carefully tailored for this defense capability [31]–[34] However, these algorithm-level defenses often depend on a DNN-specific or an attack-specific characteristic, and they can turn out to be ad-hoc solutions. Considering that the training process for DNNs is expensive and time-consuming, relying on algorithms to provide security can be unscalable as well. A hardware-oriented solution is more general compared to these algorithm-level patches.

1.2 Thesis Contributions

This thesis focuses on hardware-level vulnerabilities of the off-chip memory for DNN accelerators. The goal of this thesis is to understand the significance of those vulnerabilities and to propose a secure DNN accelerator that supports confidentiality and integrity with low overheads. This thesis has three contributions, addressing the topics at the algorithm-, architecture-, and circuit-level. We briefly introduce the key motivations and the summary for each contribution.

1.2.1 Attacking Sparse Deep Neural Networks with the Worst-case Bit Flips in the Connections

First, we investigate the significance of fault injection attacks in sparse DNNs. Fault injection attacks in the off-chip DRAM, such as rowhammer attacks [29], [35], can induce bit flips in the model parameters of a victim DNN. However, there are billions of bits in a victim DNN, and most of the bit flips can be benign without causing a significant degradation in the model prediction. Thus, hardware-level vulnerabilities alone cannot visibly disturb the victim applications if they do not know which bits to attack.

In this work, we propose a search algorithm that identifies the most critical bit flips for DNNs with fine-grained sparsity in weight tensors. This algorithm leverages the characteristic of a compressed format used to store sparse tensors. It finds a small set of bits (i.e., 1 in 2 million bits or less than 0.00005% of total bits) that cause a significant accuracy drop in a victim DNN. Therefore, the algorithm's characteristics can make exploiting hardware-level vulnerabilities easier.

This work was published at IEEE International Conference on Acoustics, Speech, and Signal Processing in 2022, and Chapter 3 is based on this publication [36].

1.2.2 Design Space Exploration of Secure DNN Accelerators

In the second contribution, we present a framework for design space exploration of secure DNN accelerators. Several hardware-level vulnerabilities of the off-chip memory and attacks exploiting those vulnerabilities targeting AI applications [17], [18], [28], [29], including our first contribution, motivate securing DNN accelerators from the untrusted off-chip memory. A trusted execution environment (TEE) can be an appealing defense solution, where all off-chip data traffic is encrypted and authenticated using a cryptographic primitive [37]–[39].

However, the cost of supporting a TEE in DNN accelerators can be elusive. DNN accelerators have a large design space, targeting diverse deployment environments from resource-constrained edge devices to high-performance data centers [40]–[44]. Also, there are diverse design choices for hardware support for cryptographic operations as well, with different performance, energy, and area trade-offs [45]–[50]. Thus, the cost of securing one accelerator design cannot be easily generalized to other designs with distinct performance goals and area/energy budgets.

We propose SecureLoop, a design space exploration framework for secure DNN accelerators, to systematically understand the cost of supporting a TEE in diverse designs. At the core of SecureLoop is the search algorithm for identifying the optimal mapping for a given DNN workload and a design specification. We illustrate the unique challenges for developing this mapping algorithm, arising from the granularity of data accesses, and provide two key techniques to overcome these challenges. Using our mapping algorithm, we show the area vs. performance trade-off for secure DNN accelerators and provide design insights.

This work was published at IEEE/ACM International Symposium on Microarchitecture in 2023, and Chapter 4 is based on this publication [51].

1.2.3 Secure Off-chip Memory Interface for Neural Engines

Finally, we implement a DNN accelerator with a secure off-chip memory interface supporting a TEE. This security support has a low overhead of less than 4% performance slowdown, 16.5% more energy consumption per each multiply-and-accumulate operation, and 8.1% of the accelerator area, demonstrating that off-chip memory security can be practically achieved for DNN accelerators targeting mobile and edge devices.

For this final contribution, we implement the design in silicon. This circuit-level implementation provides a fine-grained understanding of overhead, such as the control logic for security features, compared to the software performance simulation of SecureLoop. Also, the impact of implementation-level decisions (i.e., the same architecture specification can be implemented in different manners) can be measured with this fine-grained implementation. Lastly, the software simulation can be validated using the hardware implementation results. When there are gaps between the software simulation and the actual implementation, those gaps can benefit the future development of simulations to improve their accuracy.

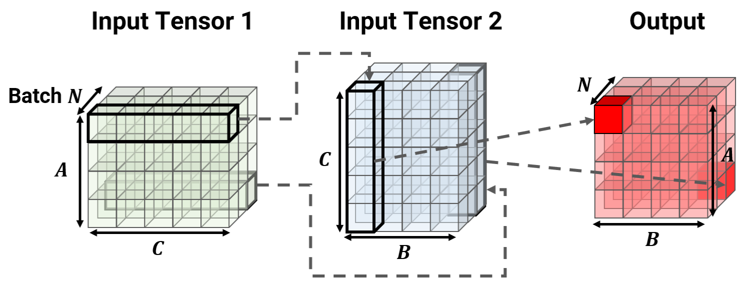
Chapter 2

Background

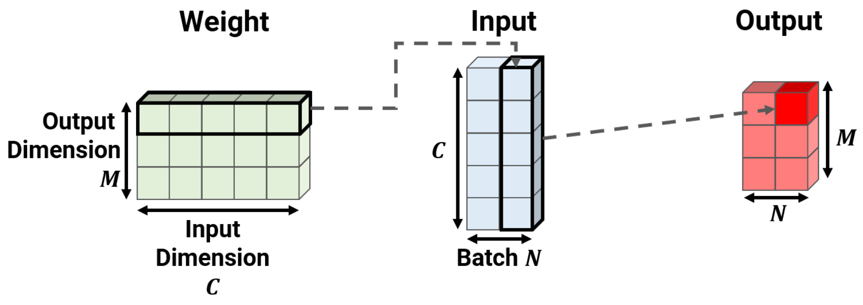
2.1 Computations in Deep Neural Networks

From the perspective of computation, DNNs are composed of multiple tensor algebra operations. Examples of widely used tensor algebra operations in a DNN include matrix-vector multiplications, matrix-matrix multiplications, and convolutions (Fig. 2.1). These operations take two tensors as operands, perform multiplications and/or summations across certain dimensions in those tensors, and produce a new tensor as an output. For example, a batched matrix multiplication (Fig. 2.1a) takes two input tensors, each with $N \times A \times C$ and $N \times C \times B$ elements, and produces an output tensor with $N \times A \times B$ elements. This computation requires one row in the input tensor 1 and one column in the input tensor 2 to be multiplied element-wise, then reduced using accumulation to generate a single output element. Note that a domain-specific language like Einsum [52], [53] can succinctly capture these tensor algebra operations, and popular software libraries for machine learning support Einsum to represent tensor algebra operations [54], [55].

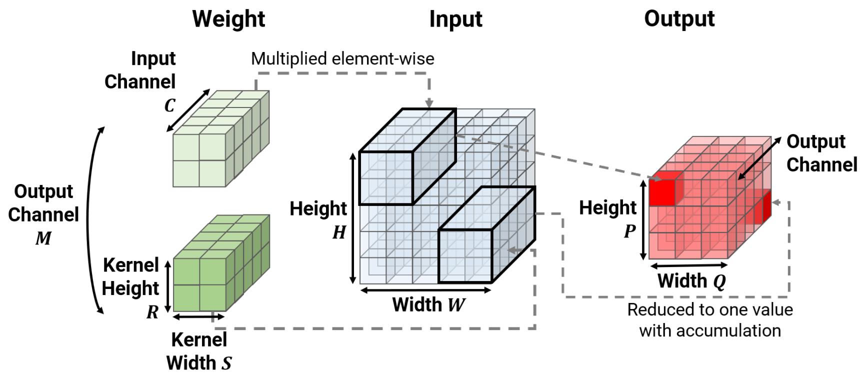
Other than these linear tensor algebra operations, DNNs often require special functions to be applied to a tensor. For example, activation functions like ReLU and softmax are applied to a tensor to add non-linearity. Also, pooling operations that compute a maximum



(a) Batched matrix-matrix multiplication. Two input tensors, each a batch of 2-dimensional matrices, are multiplied to generate the output tensor.

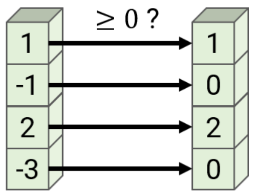


(b) Batched matrix-vector multiplication. A weight matrix is multiplied with a batch of 1-dimensional input vectors to generate a batch of output vectors.

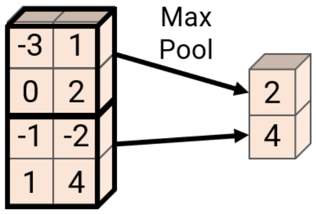


(c) 2-dimensional convolution. A 4-dimensional weight tensor is applied to a 3-dimensional input tensor to generate a 3-dimensional output tensor.

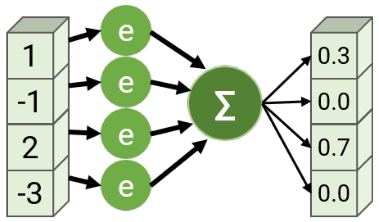
Figure 2.1: Examples of linear tensor operations widely used in DNNs.



(a) A pointwise nonlinear activation function, ReLU, applied to a tensor



(b) A reduction operation, MaxPool, is applied over a window of elements to generate a smaller output tensor



(c) A reduction operation can be applied to the entire dimension of a tensor, such as in Softmax

Figure 2.2: Examples of special funtions applied to a tensor.

or average value over a window of elements are applied to reduce the intermediate tensor sizes [56]. These special functions can be element-wise operations (e.g., ReLU) or require reduction over the entire dimension (e.g., softmax).

These tensor operations and special functions can be composed to build a complex computation graph for DNNs. For example, convolutional neural networks widely used for computer vision applications stack multiple 2-dimensional convolutions, and non-linear activation functions, followed by the final matrix-vector multiplication [57]–[62]. A self-attention module in Transformer models first projects the input tensor to three intermediate tensors using matrix-vector multiplication, then computes matrix-matrix multiplication over these intermediate tensors followed by a softmax activation function [63]–[65].

Often, a 'layer' in a DNN refers to one linear tensor algebra operation followed by a non-linear activation function (e.g., one 2-dimensional convolution followed by a ReLU activation)

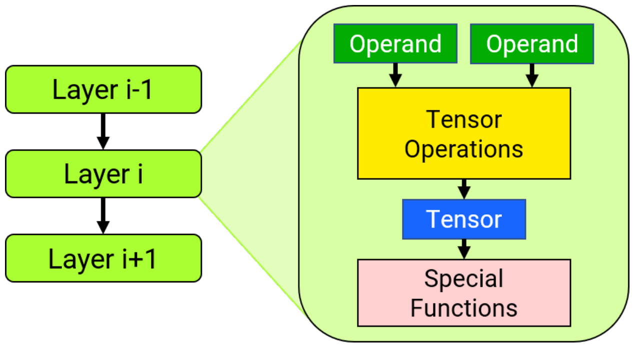


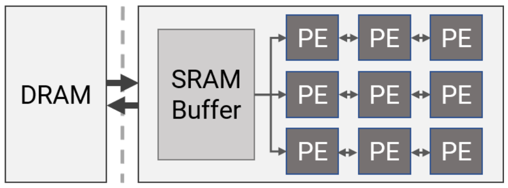
Figure 2.3: DNNs have multiple layers, where each layer is typically composed of a linear tensor operation followed by a nonlinear activation function.

(Fig. 2.3). This terminology is typically used to describe an algorithmic motivation of a DNN, where a DNN can be thought of as stacking multiple layers of neurons that perform a transformation on their inputs (e.g., multi-layer perceptrons) [56]. Since linear tensor operations with no intermediate non-linearity can be folded together as a single linear tensor operation, we refer to a 'layer' as one linear tensor operation followed by a nonlinearity in this thesis.

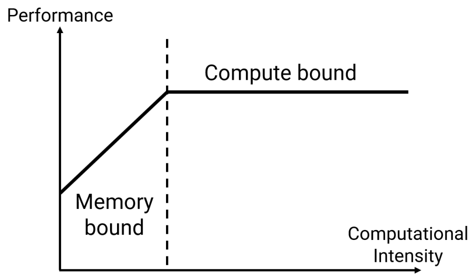
For linear tensor operations, the operands to one layer can be the output of its previous layer or a 'weight' tensor that represents the learnable parameters in a DNN. A weight tensor is fixed during the inference, and it has no dependency on other tensor data in a computation graph for a DNN. However, when an input operand is the output of a previous tensor operation, it creates a sequential (or 'cross-layer') dependency in a computation graph.

2.2 Hardware Support for Deep Neural Networks

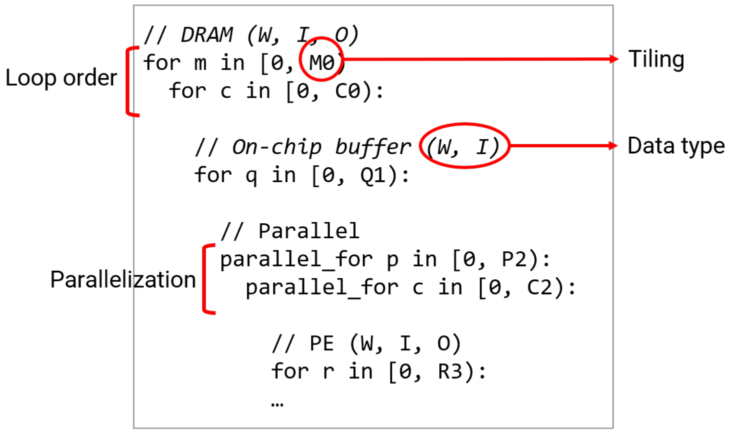
DNNs have a large number of parameters (i.e., the size of 'weight' tensors) and the total number of computations (i.e., floating-point multiplications and additions) required to get the final output is also large. Meanwhile, many of the computations in tensor operations can be parallelized. For example, consider matrix-vector multiplication. Each element in its output tensor can be viewed as an inner product of the input tensor and one row in the weight tensor, and each element can be computed in parallel as there is no dependency between



(a) An example DNN accelerator with a three-level memory hierarchy (DRAM, on-chip SRAM, and PE registers)



(b) A roofline model illustrating the relationship between the memory access and the computation



```
// DRAM (W, I, O)
for m in [0, M0):
  for c in [0, C0):

    // On-chip buffer (W, I)
    for q in [0, Q1):

      // Parallel
      parallel_for p in [0, P2):
        parallel_for c in [0, C2):

          // PE (W, I, O)
          for r in [0, R3):
          …
```

(c) A loopnest describing the mapping of a DNN workload, where the tiling strategy, datatypes kept at each memory hierarchy, parallelization, and the computation order are described

Figure 2.4: A DNN accelerator design can be conceptually described using a roofline model to capture the performance depending on the memory access and a loopnest to specify the actual computations.

the elements. Such parallelisms are key to accelerating the computing of DNNs. Among commercially available hardware, Graphical Processing Units (GPUs) are widely adopted for this purpose [57]. However, there have been an increasing number of efforts in designing domain-specific accelerators for DNNs, both in academia [40]–[42] and industry [44]. While these domain-specific accelerators have limited capacity for general-purpose computing like GPUs, they optimize the data movement and computation for tensor operations in DNNs and improve the performance and energy of computing.

Domain-specific DNN accelerators tackle the data- and compute-intensive nature of DNNs with efficient memory hierarchy and parallelism. Fig. 2.4(a) shows an example accelerator design. It has multiple *processing elements* that have multiply-and-accumulate arithmetic units and registers for holding the operands and results. These processing elements form a spatial array that can operate in parallel for computations.

While the size of this processing element array determines the maximum parallelism that can be achieved by a design, *memory* plays a critical role in the overall performance and energy. In this example, there are three different types of memory elements: an off-chip DRAM, an on-chip buffer, and registers at each processing element. They have different characteristics in their throughput, latency, energy, and area: the off-chip DRAM has the largest capacity per area, but has high latency and energy per access, whereas the registers are fast to access but are limited in their area-efficiency. As a result, different memory elements form a hierarchy, and an accelerator design aims to minimize the energy and improve the throughput by carefully coordinating the data movement across the memory hierarchy [40], [66].

Note that the bandwidth of memory elements can limit the data supply to the processing element array, preventing the design from achieving the maximum possible performance. This relationship between the memory elements and the processing elements can be concisely captured by a roofline model [67] (Fig. 2.4(b)). The x-axis and the y-axis of a roofline model represent the number of operations per byte of data (i.e., computational intensity) and the performance measured as the number of operations per second, respectively. The slanted line represents the 'memory-bound' region, where the data supply rate limits the accelerator from utilizing all processing elements. The horizontal line shows the 'compute-bound' region, where the data is sufficiently provided but the size of the processing element array itself limits the performance. Therefore, it is important to balance the parallelism with the memory hierarchy: if the computational intensity cannot be easily increased to fall in the compute-bound region, increasing the size of the processing element array will not improve the performance.

Tensor algebra operations in DNNs are *mapped* to the memory hierarchy and the processing element array of the DNN accelerators. There can be several different ways a tensor algebra operation can be actually performed in a DNN accelerator, including how tensors are partitioned to be stored at each memory level and how the computations are scheduled temporally [40], [52], [66]. Mappings refer to these different ways of performing the actual computation, and determining the optimal mapping for a given DNN workload and a DNN accelerator architecture can be thought of as 'compiling' a program in general-purpose computing [66].

A mapping can be concisely described using a nested for-loops (Fig. 2.4(c)). The processing element array will compute several arithmetic operations in parallel. Then, different data will be supplied to the processing element array across time, and the memory hierarchy will coordinate this data movement. This spatial and temporal coordination of data is described as a nested for-loop (Fig. 2.4(c)), where the parallel for-loops describe the computations that are spatially parallelized over the processing elements and the for-loops at each memory level describe the *tiling* of the tensors and the order of computation. Here we note that *tiles* are the basic granularity of data movement in the memory hierarchy of a DNN accelerator, where tiles are small partitions of tensors to enhance data reuse.

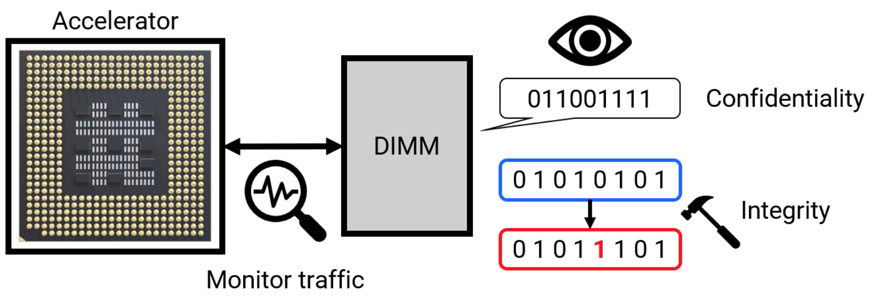


Figure 2.5: Off-chip memory components like DRAM can be exposed to several hardware-level attacks.

2.3 Off-chip Memory Vulnerabilities

This thesis focuses on hardware-level vulnerabilities of the off-chip memory components, especially DRAM. Modern DNNs have large memory footprints, and not all of the data can be stored in on-chip structures like global buffers or caches. Thus, the off-chip DRAM is necessary for computing large DNNs, and it acts as a main memory for the hardware accelerator.

However, compared to the on-chip structures, the off-chip memory components are exposed to potential threats (Fig. 2.5). First, the bus connecting the hardware accelerator and the off-chip memory can be monitored by an adversary. This bus-snooping attack can recover the memory traffic, including the data itself and the metadata such as addresses and request types for the off-chip memory access. Second, the data stored in DRAM can be exposed to an adversary exploiting cold-boot attacks [28]. Although DRAM is a volatile memory, meaning that it requires a power supply to keep the data, the data can be readable for a short period even after it is powered off. Using this data remanence property, cold-boot attacks recover the data stored in DRAM by rebooting the victim hardware system and dumping the memory content [68]. Third, the data stored in DRAM can be perturbed, undermining data integrity. [35] proposed a rowhammer attack following an observation that repeatedly accessing a row in DRAM causes bit flips in its neighboring row. The interference between neighboring rows can cause a predictable bit flip pattern, and the attack could evade the error-correcting mechanisms of DRAM [29].

These vulnerabilities of DRAM can be detrimental to the confidentiality and integrity of AI applications. Recent work [17], [18] showed that a rowhammer attack can be practically used to induce bit flips in the model parameters of a DNN. Also, successful cold-boot attacks will read out the weight and intermediate tensors of a victim DNN if they are stored in DRAM.

Chapter 3

Attacking Deep Neural Networks with the Worst-case Bit Flips

Off-chip DRAM is susceptible to fault injection attacks, such as Rowhammer [29], [35], that can flip values stored in its bit cells. As a result, the weight tensors of a DNN stored in the off-chip DRAM can be affected, and the integrity of AI applications can be undermined. However, many bit flips in the weight tensors can be benign and do not change the prediction of the victim DNN. Thus, for an adversary to successfully exploit DRAM vulnerabilities to mislead the prediction of a DNN, an adversary needs to understand which bits in the weight tensors are critical, or one has to inject a large number of faults into DRAM.

This chapter proposes an algorithm to identify the critical bits in weight tensors of a DNN, especially for a DNN optimized for low memory footprint in edge applications using model compression techniques [69], [70]. Those critical bits can be the target of fault injection attacks, and this algorithm shows that the characteristics of DNNs can make the hardware-level attack easier. This algorithm can be also used to reveal the worst-case behavior of DNNs under bit flips (i.e., bit flips are not only caused by fault injection attacks but can be induced by soft errors inherent to DRAM).

3.1 Background

3.1.1 Bit Flip Attacks

Fault Injection Attacks

Several fault injection attacks that exploit hardware-level vulnerabilities of DRAM have been proposed. Notably, rowhammer [29], [35] exploits the electromagnetic interference between physically close rows in DRAM to induce bit flips. This interference allows rowhammer to predictably cause bit flips in a victim row in DRAM when an adversary has access to the neighboring rows of the victim. Rowhammer repeatedly accesses the neighboring rows, such as requesting multiple reads to those rows. While using rowhammer requires a careful templating of DRAM to characterize the vulnerable bit cells and memory massaging to lead the processor to allocate the victim data to the desired bit cells, many works demonstrated that rowhammer can be practically utilized to wage privilege escalation attacks where an adversary can gain the root access of the processor [71].

Furthermore, if an adversary has physical access to the hardware, more direct sources of fault injection can be used. For example, simple timing and voltage glitches can be used to disturb the memory operations [72], [73], although their precision may be limited. Also, electromagnetic pulses can be injected, and precise bit flips might be induced for high-resolution pulses (e.g., pulses have high spatial and temporal resolution to target the desired component) [74]–[76]. Even for lower-resolution pulses, such indiscriminate faults can be still used to cause an abnormal execution pattern [77]. Finally, laser attacks cause precise bit flips, although they require high-cost setup [78].

Attacking DNNs with Fault Injection

Recent work showed how these fault injection methods can be used to attack DNNs. [18] showed that even a single bit flip can be detrimental for DNNs using floating-point number representations for their weight tensors. In particular, they found that the most significant exponent bit in a floating-point number can be critical, as this bit flip causes a significant change in the value. Their analysis showed that 40-50% of total parameters in weight tensors can cause more than 10% drop in the accuracy of a DNN when a single bit in their floating-point representations is flipped.

On the other hand, quantized weight tensors that use fixed point representations have a smaller dynamic range compared to floating point numbers. Consequently, indiscriminately flipping the most significant bit (MSB) of weights does not cause detrimental degradation of performance for DNNs [18]. However, [17] proposed a gradient-based search algorithm to attack quantized DNNs, and showed that only 2-24 bit flips are required to cause the accuracy of the victim DNN to drop to the random-guess level. This gradient-based algorithm requires an adversary to know the exact weight tensors (i.e., the white-box attack scenario where an adversary possesses knowledge of the victim model) and can be applied to many open-sourced DNN models [79].

Finally, [18], [80] demonstrated that rowhammer can be practically applied to induce the desired bit flips in an actual hardware system. For example, [80] showed that their rowhammer attack requires less than 100 seconds to flip 10-20 bits identified by the gradient-based search algorithm.

3.1.2 Pruning and Sparsity in Deep Neural Networks

Weight Pruning

Pruning has been proposed as a method to reduce the memory footprint of weight tensors [69], [70]. Pruning eliminates weights with less importance and sets those weights to zero, inducing sparsity in weight tensors. Several pruning algorithms have been proposed, with different approaches to quantifying the importance of weights and the pattern of sparsity. For example, [69] used the magnitude of weights as the importance and set the weights

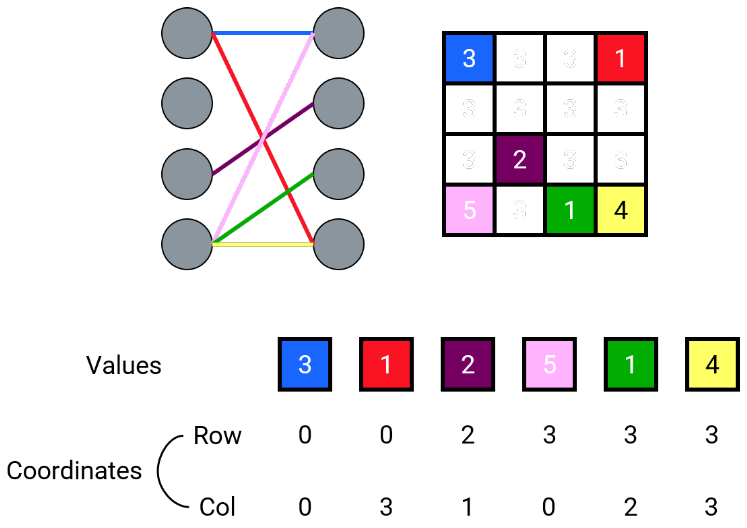
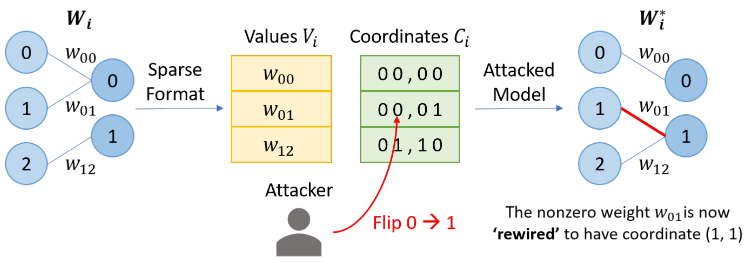


Figure 3.1: A sparse tensor can be represented with a sparse matrix format, that separtely stores the nonzero values and the coordinates of those values. In this example, a sparse weight tensor is stored using a coordinate list (COO) format.

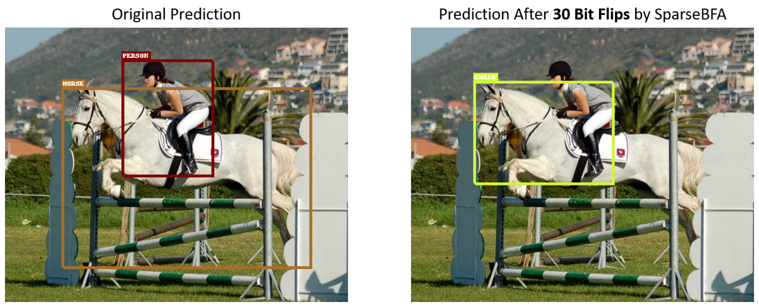
with a small magnitude to zero. This method results in fine-grained unstructured sparsity in weight tensors. Other algorithms proposed using the first-order Taylor's expansion term, the second-order derivatives, and the first-order derivatives (gradients) as the importance metric [81]–[84]. Also, several works proposed structured sparsity considering the hardware support for sparse tensor operations [85], [86].

Sparse Matrix Format

Sparse tensors can be compactly represented using sparse matrix formats [87], [88]. Instead of storing the elements in a sparse tensor contiguously including zero values, as in general dense tensors, sparse matrix formats separate the information about the nonzero values and the location of those values in the matrix. For example, the coordinate list (COO) (also known as coordinate-payload) format stores the list of nonzero values in a tensor and a separate list of their corresponding *coordinates* (Fig. 3.1). Different formats, such as the compressed sparse column (CSC) and compressed sparse row (CSR), further compress the coordinates in certain dimensions in a tensor. These sparse matrix formats reduce the memory footprint of tensors with a high sparsity level (i.e., if the tensor has a low level of sparsity, then the



(a) Visualizing how a bit flip in the coordinates affects a DNN



(b) Predictions of the object detection model are degraded by SparseBFA

Figure 3.2: (a) When an attacker flips a bit in the coordinates representing the location of nonzero weights, the connection between neurons is rewired. (b) SparseBFA aims to find the minimal number of bit flips in the coordinates to degrade the performance of DNNs. In this example, the object detection model [89] that originally predicts both a person and a horse correctly loses its performance after 30 bit flips found by SparseBFA, and wrongly predicts that a chair is present in the image (from PASCAL-VOC [90]).

coordinate information will only add additional storage overhead).

3.2 Sparsity as a New Vulnerability

Sparse DNNs have a unique source of vulnerability when their weight tensors are stored using sparse matrix formats. These formats explicitly store the coordinates of nonzero weight elements. When the coordinates are corrupted by fault injection attacks, the flipped bits will *rewire* the connections between the neurons (Fig. 3.2) instead of changing the values of weights directly. Therefore, the *connections* in sparse DNNs can be the target of attack, in addition to the values that have been previously considered in prior work [17], [18].

This section outlines the search algorithm, SparseBFA, for attacking the coordinates of the sparse tensors representing the weights of a victim DNN. Similar to quantized DNNs with fixed point representations that require gradient-based search instead of indiscriminate bit flips [17], random bit flips do not significantly affect the accuracy of a victim sparse DNN. To efficiently and effectively identify the vulnerable bits, SparseBFA employs gradient-based search with heuristics to reduce the search space considering the characteristics of bit flips on the coordinates.

3.2.1 Problem Definition and Notation

Our goal is to find the smallest number of bits in the coordinates of nonzero elements of a sparse DNN $f(\cdot)$ such that flipping those bits results in a large degradation of performance. Suppose $f(\cdot)$ is parameterized with $\Theta = \{W_i\}_{i=1}^{L}$ (ignoring biases for simplicity), where W_i represents the weight tensor of the ith layer in $f(\cdot)$, and its weight matrices/tensors are stored using sparse matrix formats, such as COO or CSC. Although we will be using COO when explaining our algorithm, note that different sparse matrix formats are interchangeable [87], [88] and the analysis we provide is applicable to the other formats without much modification. We denote the list of nonzero values for the ith weight matrix W_i as V_i, and the list of the coordinates corresponding to those nonzero values as C_i as in Fig. 3.2. The jth element in C_i (denoted as $C_i[j]$) represents a coordinate as a d-tuple of n_c-bit unsigned integers, where d is the dimension of W_i (e.g., 4 for weight tensors in 2-dimensional convolution layers). Also, $C_i[j][k][n]$ indicates the nth bit in the kth dimension of the coordinate $C_i[j]$, and its value will be either 0 or 1. Note that we can specify a specific bit in the coordinates using four variables, i (layer), j (element index), k (dimension), and n (bit).

We denote the loss of a DNN given input samples $\mathbf{x}$ and the corresponding ground truth targets $\mathbf{t}$ as $l(f(\mathbf{x};\Theta),\mathbf{t})$. This loss can be a cross-entropy loss for the classification tasks or multi-box loss [89] for the object detection tasks. Furthermore, we assume that $f(\cdot)$ is generated by iteratively pruning a dense DNN to have the sparsity of $r \in [0,1]$ for each layer, where r represents the proportion of zero values in the weight tensor, and that nonlinear quantization is applied for the value of weights [70].

Putting these notations together, we want to identify a set of bits specified as $((i_1, j_1, k_1, n_1), (i_2, j_2, k_2, n_2), ..., (i_M, j_M, k_M, n_M))$ that maximally degrades the performance of a victim DNN when flipped. Specifically, we want to minimize M while achieving

$$\text{Accuracy}(f(\cdot, \Theta^M)) < \text{Random guess} \tag{3.1}$$

$$\text{where } \Theta \xrightarrow{\text{flip}(i_1,j_1,k_1,n_1)} \Theta^1 \rightarrow \cdots \xrightarrow{\text{flip}(i_M,j_M,k_M,n_M)} \Theta^M \tag{3.2}$$

such that the final accuracy of a victim DNN is below the random guess level (i.e., the baseline accuracy achieved without any training) for the dataset this DNN has been trained, such as 10% accuracy for a 10-way classification task.

3.2.2 Threat Model

Throughout this chapter, we assume that an adversary has full knowledge of the weight tensors of a victim DNN, including the sparse matrix formats storing those tensors. Furthermore, an adversary requires a partial knowledge of the training dataset on which a victim DNN has been trained, although it does not need to know the entire dataset. For example, SparseBFA used 128 training samples for CIFAR-10 [91] and TinyImageNet [92] dataset, accounting for only 0.256% and 0.128% of the training dataset.

Once an adversary identifies the target bits using SparseBFA, we assume that an adversary can induce precise bit flips using fault injection methods discussed in Section 3.1.1, such as Rowhammer [29], [35]. For example, if the victim process running inference using the victim DNN is co-located with an adversary's process in the datacenter, an adversary can wage Rowhammer using techniques proposed in [18], [80].

Algorithm 1 SparseBFA for the COO format

```
1: $m \leftarrow 0$
2: while $m < M$ do                                        $\triangleright$ $M$: the maximum number of iterations
3:     loss_dict $\leftarrow \emptyset$
4:     for $i \in S_L$ do
5:         $C_i, V_i \leftarrow$ CONVERT_SPARSE($W_i$)
6:         $S_j \leftarrow$ GET_CANDIDATES_INDEX($C_i$)
7:         for $j \in S_j$ do
8:             for $k \leftarrow 0, d$ do
9:                 for $n \leftarrow 0, n_c$ do
10:                    $C_i^\star \leftarrow C_i$
11:                    $C_i^\star[j][k][n] \leftarrow 1 - C_i^\star[j][k][n]$
12:                    if IS_VALID($C_i^\star$) then
13:                        $W_i^\star \leftarrow$ RECONSTRUCT($C_i^\star, V_i$)
14:                        $\Theta^\star \leftarrow \Theta \setminus \{W_i\} \cup \{W_i^\star\}$
15:                        loss $\leftarrow l(f(\mathbf{x}; \Theta^\star), \mathbf{t})$
16:                        loss_dict$[(i, j, k, n)] \leftarrow$ loss
17:    $(i_m, j_m, k_m, n_m) \leftarrow$ GET_MAX_KEY(loss_dict)
18:    $C_{i_m}, V_{i_m} \leftarrow$ CONVERT_SPARSE($W_{i_m}$)
19:    $C_{i_m}[j_m][k_m][n_m] \leftarrow 1 - C_{i_m}[j_m][k_m][n_m]$
20:    $W_{i_m} \leftarrow$ RECONSTRUCT($C_{i_m}, V_{i_m}$)
21:    $m \leftarrow m + 1$
```

3.2.3 Algorithm: SparseBFA

We propose the SparseBFA algorithm, which iteratively searches for a bit in the coordinate lists of $f(\cdot)$ that results in the largest increase of the loss using a small number of samples $(\mathbf{x}, \mathbf{t})$. In each iteration, SparseBFA checks every possible valid bit flip for the selected elements in the coordinate lists and evaluates the loss for each bit flip by forward-propagating the samples.

This idea is outlined in Algorithm 1. First, SparseBFA visits each layer whose index is in S_L, a set of indices of layers that are to be attacked (e.g., if all layers are considered, $S_L = \{1, \ldots, L\}$). After converting the weight matrix W_i to the coordinate list C_i and the value list V_i, the `Get_Candidates_Index` function returns S_j, the indices of the selected elements in C_i. Then, for each selected element $C_i[j]$, SparseBFA checks all $d \times n_c$ possible bit flips for their validity, that is, whether the flipped coordinate list $C_i^\star$ satisfies the properties of the sparse matrix format used, such as no overlapping coordinates in $C_i^\star$. For example, a bit flip that results in rewiring to an already existing connection will be considered invalid as it can cause an attack to be detected due to the abnormal sparse matrix format. Then, for all valid bit flips, SparseBFA evaluates the loss for valid bit flips. Finally, after evaluating all candidates in S_j, SparseBFA will choose a single bit that results in the maximum loss.

The complexity of SparseBFA is determined by the `Get_Candidates_Index` function: for example, if this function returns every index of C_i, then the algorithm becomes an exhaustive search. Alternatively, the complexity can be reduced if this function proposes only a few candidates by approximating the importance of each element in C_i, although the flipped bit might not maximally increase the loss. We present both the exhaustive search method and the approximation method and analyze their effectiveness and computational complexity in the following section.

Exhaustive Search Method

An important characteristic of our problem that distinguishes SparseBFA from the prior work on bit flip attacks is that it is a combinatorial optimization problem since the bit flips in the coordinate lists 'rewire' the connections between neurons. Thus, we cannot directly adopt the previous BFA algorithms, such as [17] developed for linearly-quantized dense DNNs, that used gradient ascent to determine the candidate bits. For example, in linearly-quantized DNNs, the impact of a bit flip on the nth bit of a weight element w can be approximated as :

$$\frac{\partial l}{\partial w_n} = \frac{\partial l}{\partial w}\frac{\partial w}{\partial w_n} = w_g \times 2^n \tag{3.3}$$

where w_n denotes the nth bit of w and w_g denotes the gradient computed for this element. However, this relationship does not hold for rewiring, and a bit flip in the coordinates cannot be directly linked to the gradient information of each weight. Instead, we first start with an exhaustive search to solve our problem, which can find the optimal bit flip in each iteration.

We apply an exhaustive search for a layer with a relatively small weight tensor, such as the first convolutional layer of DNNs. That is, we check every nonzero weight in this layer and evaluate every possible valid bit flip on the coordinates of the nonzero weights. We find that even a single bit flip in the coordinate list can be detrimental. For example, an exhaustive search can find a single bit in the coordinate list of the first convolutional layer of a ResNet50 [60] model, which is trained to classify the TinyImageNet dataset with top-1 accuracy of 60%, that results in an accuracy drop to almost random-guess level ($< 1\%$) when it is flipped.

While this experiment can be a proof-of-concept for the effectiveness of bit flip attacks on the coordinates, an exhaustive search cannot be extended to more general layers with larger weight tensor sizes due to its prohibitive computational complexity. Suppose the weight tensor of the ith layer in a victim DNN, W_i, is a d-dimensional tensor with the size $N_1 \times N_2 \times \cdots \times N_d$. Then, an exhaustive search needs to check all $r \cdot \prod_{k=1}^{d} N_k$ nonzero elements, where $0 \leq r \leq 1$ represents the ratio of nonzero elements in the weight tensor. For each nonzero element, we need to check all bits in the coordinate representing its location, meaning $\sum_{k=1}^{d} \log(N_k)$ bits need to be checked since the minimum number of bits to represent a coordinate is proportional to $\log(N_k)$ for each dimension k. As a result, total $r \cdot (\prod_{k=1}^{d} N_k)(\sum_{k=1}^{d} \log(N_k))$ forward-propagations are required for an exhaustive search. This motivates us to develop an approximation method that chooses S_j to be a small subset of indices such that the complexity can be reduced to $|S_j| \cdot \sum_{k=1}^{d} \log(N_k)$, making the computation scales logarithmically with the size of a tensor.

Approximation Methods

A simple approximation method can randomly sample S_j so that different subsets of nonzero coordinates can be checked at each iteration (SparseBFA-RandomSubset). If the number of iterations is sufficiently large, then this random sampling will visit almost all nonzero coordinates, approximating the behavior of an exhaustive search. However, there is a trade-off between the size of a subset S_j to be visited and the number of iterations, and this method might not find the smallest number of bit flips.

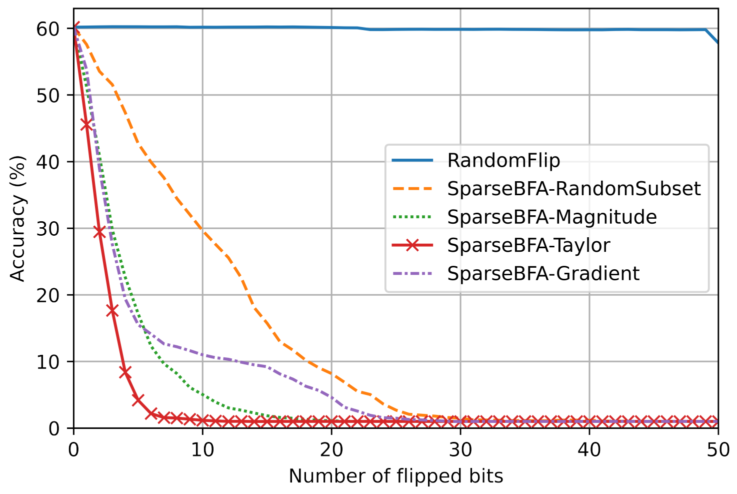


Figure 3.3: Accuracy of the ResNet50 model as bits in the coordinate list are flipped using different approximation methods for SparseBFA. RandomFlip, which induces random bit flips similar to soft errors, is reported as a baseline. All experiments are repeated five times, and we report means.

To overcome this challenge, we investigate other approximation methods that utilize the characteristics of bit flips in the coordinate lists. Recall that bit flips in the coordinate result in the rewiring of connections (Fig. 3.2). The rewiring can be broken down into two separate steps: first, the previously nonzero weight corresponding to the flipped coordinate will be set to zero, similar to that weight being 'pruned' away; second, the connection is reintroduced elsewhere. Therefore, the impact of the rewiring will be the combination of pruning away an existing connection, which is determined by choosing S_j (Line 6 in Algorithm 1) and reintroducing the connection to elsewhere, which is determined by the inner-loop in the search algorithm (Line 8-16).

We want to approximate the process of identifying S_j and considering the characteristic that it is similar to pruning, one approach can be estimating the importance of each nonzero weight similar to weight pruning and choosing S_j as indices of the coordinates with high importance. We examine two importance metrics from pruning literature: 1) the magnitude of values $|w|$ [69] (SparseBFA-Magnitude), and 2) the first-order term in Taylor's expansion $|w \cdot \frac{\partial l}{\partial w}|$ [81] for each nonzero weight w (SparseBFA-Taylor). The first approach is widely used for inducing unstructured sparsity in weight tensors and was shown to be effective for removing less important weights when combined with fine tuning [69]. While the second approach requires back-propagation for evaluating the importance, it is more mathematically intuitive to interpret based on Taylor's expansion, since it directly measures the first-order difference between the perturbed function and the original function:

$$f(x;\Theta+\Delta\Theta) \approx f(x;\Theta) + \Delta\Theta\frac{\partial f(x;\Theta)}{\partial\Theta} + \cdots \tag{3.4}$$

Finally, we also investigate using the gradients $|\frac{\partial l}{\partial w}|$ to estimate importance as in [17] (SparseBFA-Gradient). Here the assumption is that the importance of nonzero weights can be measured similarly to the weights in dense DNNs where the previous bit flip attacks succeeded by using this gradient information.

Fig. 3.3 shows the effectiveness of four approximation methods. For the ResNet50 model, which we analyzed for the exhaustive search method in Section 3.2.3, we measure how its accuracy changes as the bits in the coordinate lists are flipped using different approximation methods. For this experiment, we set $S_L = \{2, \ldots, L-1\}$, excluding the first and the last layer from the attack (otherwise a single bit flip at the first layer alone can degrade accuracy to the random-guess level), and set $|S_j| = 5$.

First, observe that random bit flips without SparseBFA, similar to bit flips that can be induced by soft errors, do not significantly affect the accuracy. However, SparseBFA with any of the approximation methods successfully identifies < 30 bit flips that result in the complete degradation of the performance of the victim DNN, with an accuracy of less than 1%. Second, SparseBFA-RandomSubset requires the most number of bit flips to reach the 1% accuracy, showing that the other three approximation methods estimating the importance of weights are effective compared to this random sampling method. Finally, we find that SparseBFA-Taylor provides the best attack with an average of 11.8 bit flips to reach less than 1% accuracy. Since the goal of using these approximation methods is to identify the most important nonzero weights, without the consideration for fine-tuning or retraining as

Dataset	Model	Sparsity	Total Bits	Original Accuracy	Target Accuracy	# Bits Flipped to Reach Target	
						All	Except First, Last
CIFAR-10 (Classification)	Conv4FC2 [1]	98.66%	14.26M	86.83%	$< 11\%$	1.4 ($\pm$0.55)	3.0 ($\pm$1.87)
	WideResNet [93]	94.37%	8.34M	93.14%		1.0 ($\pm$0)	12.8 ($\pm$2.17)
	MobileNetV2 [61]	94.37%	12.98M	90.06%		6.8 ($\pm$2.68)	8.8 ($\pm$2.17)
TinyImageNet (Classification)	ResNet50 [60]	94.37%	36.50M	60.18%	$< 1\%$ (Top-1)	3.2 ($\pm$4.38)	11.8 ($\pm$3.11)
	MobileNetV2 [61]	92.50%	17.73M	43.88%		1.4 ($\pm$0.89)	2.8 ($\pm$0.45)
PASCAL-VOC (Detection)	SSD300 [89]	82.20%	137.38M	0.70 mAP [2]	< 0.10mAP	34.2 ($\pm$3.42)	-

[1] 4 convolutional layers followed by 2 linear layers. Batch normalization is used for the convolutional layers.
[2] Note that the SSD300 model is not finetuned after quantization of the weight values. We report mean average precision (mAP) by randomly sampling 1000 images from PASCAL-VOC 2007 test dataset.

Table 3.1: Summary of the characteristics of DNNs and the number of bit flips found by SparseBFA-Taylor required to reach the target accuracy level. We report sparsity, the total number of bits (when the model parameters are stored using COO, assuming that the values are nonlinearly-quantized using 4-bits for CIFAR-10/TinyImageNet and 6-bits for PASCAL-VOC), and the original accuracy of the models, along with the number of bit flips. All SparseBFA experiments are repeated 5 times with different random seeds, and we report means and standard deviations for the number of bit flips.

in the pruning literature, the first-order Taylor expansion term can be more effective than other proxies like the magnitude of the gradient.

3.3 Attack Results

In this section, we demonstrate the vulnerability of sparse DNNs with diverse architectures and datasets to SparseBFA. Table 3.1 shows the number of bit flips required to degrade the performance of each sparse DNN to the target accuracy level (e.g., random guess level for classification tasks). We use SparseBFA-Taylor as the attack algorithm with $|S_j| = 5$, meaning that at each iteration SparseBFA chooses top-5 nonzero weights with the largest first-order Taylor expansion term to search for the bit flip. Also, SparseBFA uses randomly sampled inputs $(\mathbf{x}, \mathbf{t})$ from the training dataset to run forward and backward propagation required to evaluate the bit flip candidates. The size of these input samples is 128 for CIFAR-10 [91] and TinyImageNet [92] dataset, and 32 for PASCAL-VOC dataset [90]. We evaluate 6 different DNN models with different model architectures, levels of sparsity, and tasks. We also note the size of the weight tensors for each DNN model when they are stored using the COO format. We repeat all experiments 5 times to account for the stochasticity of SparseBFA due to random sampling of inputs and report the mean and the variance in Table 3.1.

First, we find that flipping only 0.00005% of total bits (1 in 2 million bits) can be detrimental to various DNNs. For example, the accuracy of the five DNNs for the image classification task decreases to the random-guess level with $1 \sim 15$ bit flips in all trials. Therefore, SparseBFA can be a practical threat considering that a similar number of bit flips can be successfully induced using Rowhammer in prior work [80]. In particular, the coordinate information in sparse tensors can be a valid target for an adversary, and the defense schemes for sparse DNNs should consider that the vulnerability against bit flip attacks is not limited to the values of the weights themselves.

Second, we observe that the first and the last layers are more sensitive to bit flips compared to intermediate layers. When the first and the last layers are not attacked, the number of bit flips to reach the target performance level increases by $1.3 \sim 12.8$ times compared to when all layers are attacked. Also, for the SSD300 model trained on the PASCAL-VOC dataset, SparseBFA cannot reach the target performance level within 50 bit flips when excluding the first and the last layers from the attack. Thus, a simple scheme to improve the robustness of sparse DNNs can protect at least the first and the last layers with security measures, such as more frequent refresh of DRAM rows to prevent RowHammer [29], or using a secure processor [94] during inference for those layers.

3.3.1 Sensitivity to Sparsity

We examine the relationship between sparsity and the vulnerability of DNNs to SparseBFA. Fig. 3.4 shows how the accuracy of the Conv4FC2 models with different sparsity levels changes as the number of bit flips increases. SparseBFA-Taylor is applied to all layers except the first and the last layers with $|S_j| = 5$, the same as our previous setup. We find that

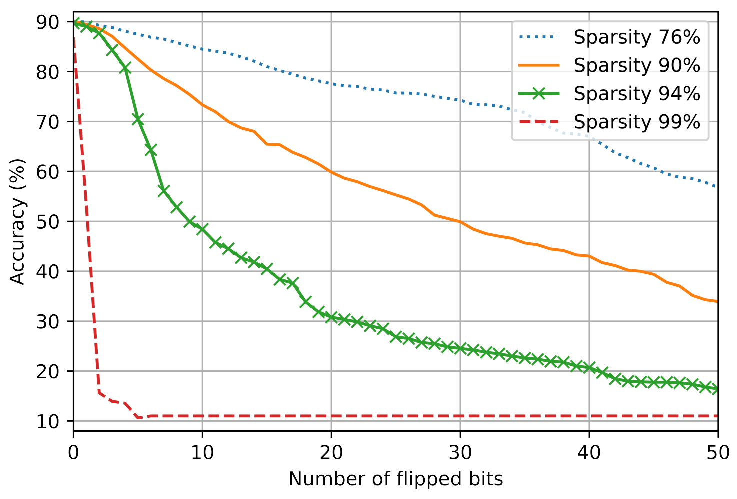


Figure 3.4: Accuracy of the Conv4FC2 models trained on CIFAR-10 with different sparsity as the number of bit flips found by SparseBFA-Taylor increases. We report the mean of five trials.

the models with higher sparsity are more vulnerable to bit flips: the accuracy of the model with 99% sparsity drops to below 11% with fewer than 5 bit flips, whereas the model with 76% sparsity still retains 55% accuracy even after 50 bit flips. One explanation for this trend is that there are fewer bit flips satisfying the `Is_Valid` function for denser models, thus limiting the feasible rewiring. Furthermore, as denser models have more redundancy, a single rewiring might not be as detrimental as in sparser models. Overall, this relationship manifests the trade-off between efficiency achieved with high sparsity and vulnerability to SparseBFA.

3.4 Summary

In this chapter, we present SparseBFA, an algorithm that identifies the most vulnerable bits in the coordinates of a sparse DNN whose weight tensors are stored using sparse matrix formats. SparseBFA can efficiently and effectively identify a small number of bits, less than 0.00005% of the total memory footprint, that results in a huge degradation of performance when perturbed. These vulnerable bits can be the target of fault injection attacks [18], [29], [35], [80] when they are stored in the off-chip DRAM. SparseBFA illustrates that the algorithmic characteristics of a DNN can be exploited to enable hardware-level attacks.

Specifically, SparseBFA presents three key contributions. First, we show that sparse DNNs that store weight tensors using sparse matrix formats can be vulnerable to bit flips in their coordinates (encoding the location of nonzero weights). This new source of vulnerability implies that the tensors using sparse matrix formats should protect the coordinate information for security, in addition to the nonzero values. Second, we address the challenge of identifying the most vulnerable bits in the coordinates by combining the exhaustive search with the approximation techniques to identify the most important nonzero weights in a sparse DNN. Bit flips on the coordinates rewire the connections between the neurons, hence they have a combinatorial nature that cannot be easily modeled with prior work on bit flips on the values of dense models [17], [80]. Finally, SparseBFA affects diverse sparse DNNs trained on different tasks. In particular, DNNs with a high level of sparsity can be heavily affected by bit flips induced by SparseBFA, showing the trade-off between the efficiency achieved by pruning and the robustness against fault injection attacks.

Chapter 4

Design Space Exploration of Secure Deep Neural Network Accelerators

From Chapter 3, we discussed how hardware-level vulnerabilities in the off-chip memory could be exploited to undermine the trustworthiness of DNNs. Providing a defense for the vulnerable off-chip memory can be critical for applications that process sensitive user information or make high-stakes decisions. In particular, a hardware solution can be beneficial in multiple ways. First, a hardware solution does not require algorithms and software stack to be modified, thus providing general protection regardless of algorithms. Second, a hardware solution reduces the root-of-trust to the secure hardware itself, not relying on any software-level details to provide security.

A trusted execution environment (TEE) is an appealing solution to provide security to the vulnerable off-chip memory. With a TEE, the off-chip data accesses are protected with cryptographic encryption and authentication, such that an adversary cannot recover the actual data and tamper with the data without being noticed. For general-purpose computing, there have been extensive research efforts in supporting TEEs [94]–[99], including the commercialized solutions by major chip vendors and various open-source solutions from academia, such as Intel SGX [100] and Keystone [39].

However, supporting a TEE for custom-designed DNN accelerators differs from general-purpose computing. First, DNNs are data-intensive applications, where the tensors associated with DNN models and their intermediate tensors can exceed gigabytes of data. Second, DNN accelerators have different off-chip data access patterns from general-purpose processors. Finally, there is a rich diversity of custom designs for DNN accelerators, ranging from those targeting high-performance data centers to those deployed in resource-constrained edge devcies [40]–[44], and the overhead associated with supporting a TEE in one design cannot be easily transferred to another design.

In this chapter, we provide a systematic approach to understanding the cost of supporting a TEE in custom-designed DNN accelerators. We address the unique challenges associated with combining cryptographic operations with the data movement pattern of DNN accelerators. Then, we propose SecureLoop, a design space exploration framework for *secure* DNN accelerators equipped with a hardware engine for cryptography and supporting a TEE. This chapter provides insights into the architecture design of secure DNN accelerators with protection for vulnerable off-chip memory.

4.1 Background

4.1.1 TEE for DNN Accelerators

Security Assumptions

We consider both the confidentiality and integrity of data stored in the off-chip DRAM. That is, the data stored in the off-chip DRAM should not be available to an unauthorized adversary, and bit flips in the data should be detected before the data is supplied for the computation. A TEE assumes that the on-chip structure, such as the on-chip memory hierarchy including SRAM and registers and the on-chip computation operations, is secure and trusted. However, it does not trust any off-chip components, and every communication

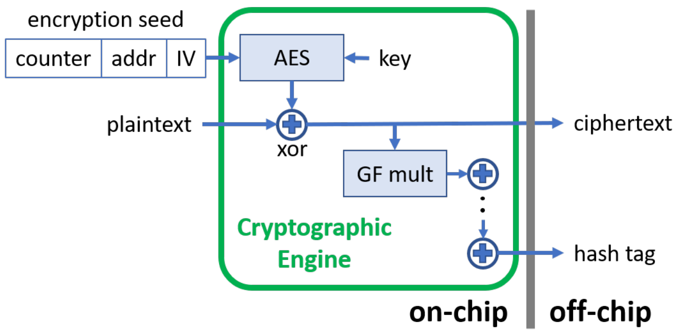


Figure 4.1: A cryptographic engine supporting AES-GCM, a widely used authenticated encryption protocol.

with the off-chip memory will be encrypted and authenticated using cryptographic primitives. Note that this assumption excludes attacks on the on-chip structure, such as the physical side-channel and fault injection attacks [25]–[27], [73], [77].

Memory Encryption and Authentication

Authenticated encryption is a cryptographic primitive that simultaneously encrypts the data for confidentiality and generates the hash for a block of data for integrity. An authenticated encryption scheme takes a plaintext, a secret key, and an encryption seed as inputs, and computes a ciphertext and a hash. Fig. 4.1 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 typically use explicit data orchestration [101] and the accelerators have full knowledge of the version number, recent works [102]–[104] 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. Finally, note that the security of authenticated encryption relies on the secret key used for cryptographic operations. If an adversary knows the secret key, an adversary can recover the plaintext and compute fake yet valid hashes for the perturbed data. Thus, the secret key is stored on-chip, typically using a special register for holding the key throughout the session until the key has to be updated.

All datatypes in a DNN (i.e., weights, input feature maps (`ifmap`), and output feature maps (`ofmap`)) 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 [96], [97]. As shown in Fig. 4.1, an AES-GCM block is primarily composed of an AES engine and a Galois-field multiplier. In the counter mode encryption, the encryption seed is fed into the AES engine to generate a one-time pad. Then, this one-time pad is XOR-ed with plaintext to obtain ciphertext, and vice versa. Note that AES is a block cipher, meaning that it operates over a fixed datapath size (e.g., 128 bits). The Galois-field multiplication is applied over multiple ciphertexts to generate a fixed-length hash for integrity verification.

There are several hardware implementations of AES-GCM with diverse performance and area characteristics. For example, Fig. 4.2 compares the AES hardware accelerator implementations published between 2001-2018 in circuits literature [45]–[50]. 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). Note that AES-

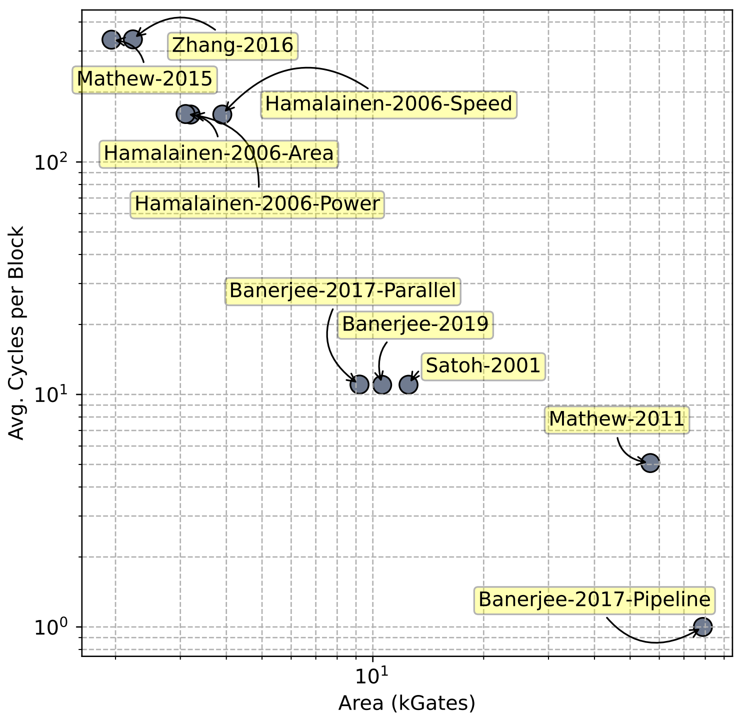


Figure 4.2: The tradeoff space for AES implementations.

GCM can be supported without a dedicated cryptographic engine in general-purpose CPUs. However, we focus on the hardware engine for AES-GCM here as DNN accelerators might not be equipped with a CPU that can compute cryptographic operations with sufficiently high throughput.

Replay Attack in DNN Accelerators

While cryptographic authenticated encryption can guarantee data confidentiality and integrity, one intricate problem is with replay attacks. An adversary can dump both the encrypted data and their hashes and replay this dumped data in later cycles. Since the replayed data has valid hashes, authenticated encryption alone cannot detect this attack. Thus, counter (i.e., timestamp of the data) information is added to the encryption seed for AES, and the correctness of counters guarantees protection against replay attacks. This counter management has been the primary focus of optimizing TEEs for general-purpose computing in prior work [95]–[99], and the major source of overhead for supporting TEEs.

However, DNN accelerators can leverage their structured and pre-determined data access patterns to *compute* the counters [102], [104], instead of storing them and managing their correctness. Therefore, as we discussed in Section 4.1.1, we assume that off-chip data accesses in secure DNN accelerators do not incur complicated memory accesses.

4.1.2 Design Space Exploration

DNN accelerators are designed to exploit substantial data reuse within tensor algebra operations. Note that while the maximum performance is determined by the accelerator design (e.g., the roofline model), the *mapping* of a DNN workload to the accelerator design determines the actual performance and energy efficiency. A mapping describes how the computation and data movement are temporally and spatially mapped to hardware resources and can be formulated using a nested for-loop called "loopnest" [52].

Prior work acknowledges that the search space for mappings is large, and efficiently searching for the optimal mapping (i.e., small latency with low energy consumption) presents a research challenge [52], [105]–[109]. Several methodologies have been proposed. For example, Timeloop [52] used brute-force search over all possible loopnests and supported approximate methods like random pruning to reduce the search time. CoSA [105], on the other hand, formulated the search problem as a constrained-optimization problem that can be solved using integer programming techniques. Furthermore, other search algorithms proposed to use machine learning, such as [108], [109], to identify the optimal mapping.

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

4.2 Challenges of Secure DNN Accelerators

Prior work on customizing TEE solutions for DNN accelerators [102]–[104] leveraged the structured and predetermined data access data access patterns of DNN accelerators and derived 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.

Recall that 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.

There is an important yet unexplored research challenge in the search space for mappings considering the impact of cryptographic operations. 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 mapping, 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 mappings of consecutive layers are intertwined, exponentially increasing the search space for mappings.

In addition to the challenges associated with optimal mapping, there is another challenge due to the diversity of DNN accelerator designs. The overhead of cryptographic engines, such as their area, energy, and power, can be significant for certain designs targeting edge and embedded platforms. Moreover, cryptographic engines have rich design space of their own as we discussed in Section 4.1.1.

In this section, we analyze these challenges in detail. First, we point out that a cryptographic engine, which is often considered a low-cost add-on to a predefined DNN accelerator in prior work [102]–[104], can pose significant overhead to different designs. Second, we point out that authentication block assignments introduce a significant amount of complexity to our scheduling search space that our tool needs to navigate.

4.2.1 Overhead of Cryptographic Engines

Existing work on designing secure DNN accelerators [102]–[104] 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. 4.2, 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 [40] and other designs [42], [43]. 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 [104]. When each

AES-GCM engine is composed of a fully-pipelined AES engine and a single-cycle Galois-field multiplier [49], this configuration requires 416.7kGates in area, approximately 35% of the logic gates in Eyeriss [40], incurring extensive area overhead.

We note that prior work [102]–[104] only considered solutions for power-hungry accelerators, such as TPU [44], 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 a 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.

4.2.2 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 4.1.1, 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 [102], [103], 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 reuse. The "tile-as-an-AuthBlock" strategy assigns authentication blocks to exactly match each datatype's tile organization.

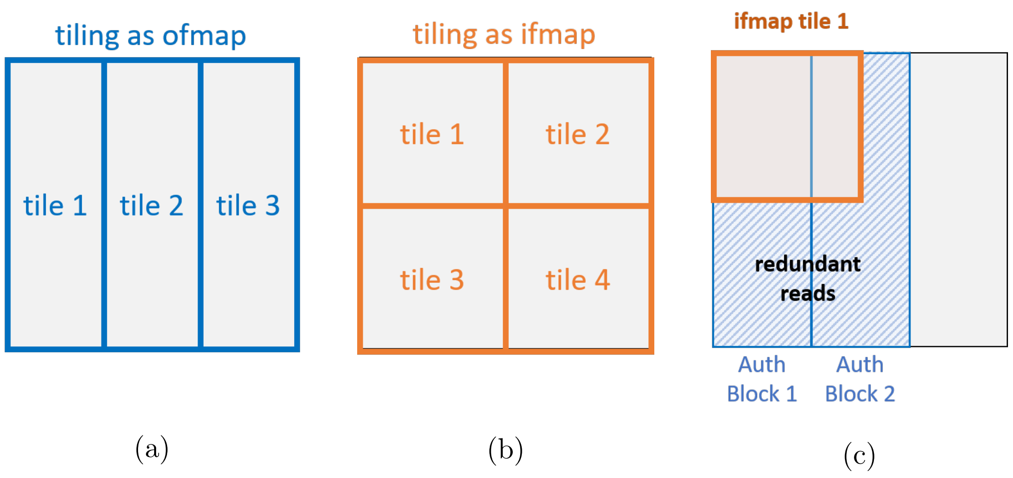
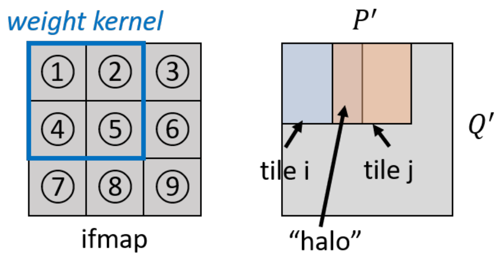


Figure 4.3: The same piece of data is used as `ofmap` of one layer (a) and as `ifmap` of the next layer (b) and the two layers use different tiling configurations. (c) shows that redundant reads are introduced when using the data as `ifmap` but assigning Authblock following `ofmap`'s tiling organization.

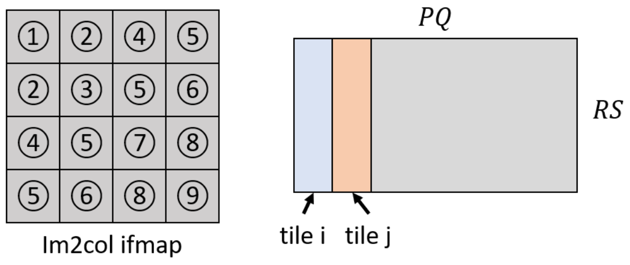
Cross-layer Dependency

"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. 4.3 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. 4.3(c), when the accelerator fetches the first `ifmap` tile for DNN computation, it is forced to fetch the whole AuthBlock 1 and 2,



(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 4.4: 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).

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, 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`.

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. 4.4 compares how tiles are organized for the two different types of accelerators. Fig. 4.4(a) shows that, due to coarse-grained tiling, the accelerators dedicated to 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. 4.4(b), in the matrix multiplication case, each element exclusively belongs to one tile and there does not exist 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.

Goal of AuthBlock Assignment

To summarize, the AuthBlock assignment poses a critical challenge in identifying the optimal mapping 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 of two layers becomes fundamentally intertwined. There might be a loopnest mapping for one layer that is not optimal on its own, but results in better overall performance when it is

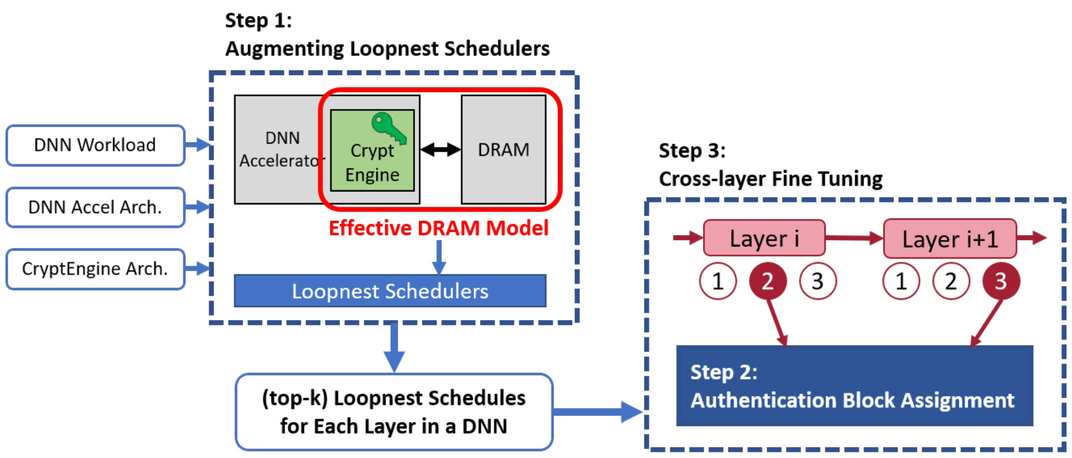


Figure 4.5: Overview of the search engine for mappings in SecureLoop.

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 mapping level and search for mappings that optimize for global performance rather than a single-layer performance. In Section 4.4, 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.

4.3 Mapping for Secure Accelerators

We present SecureLoop, a design space exploration tool that is equipped with a search engine for optimal mappings (Fig. 4.5) 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 mapping algorithms. This approach is general enough to be compatible with a broad range of existing loopenst mapping algorithms, such as Timeloop [52] and CoSA [105].

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 loopnest mapping 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 [61]. We use simulated annealing by heuristically defining neighboring loopnest configurations and search for the final schedule.

4.3.1 A Model for Cryptographic Operations

In the first step, we aim to identify the loopnest for secure DNN accelerators by leveraging the existing DNN loopnest mapping algorithms. When we exclude the complexity that arises from the authentication block assignment (Section 4.2.2), such as the additional off-chip data traffic due to the misalignment of tiles, the difference between a secure DNN accelerator and a traditional accelerator is the extra cryptographic operations performed by the augmented cryptographic engine. Thus, the DNN loopnest mappers 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 and access energy, instead of intrusively modifying the internals of the existing mappers.

Given that each piece of off-chip data access needs to go through both the DRAM interface and the cryptographic engine, the slower component among the two limits the bandwidth.

Thus, the effective data supply rate from the accelerator's perspective will be the minimum of the memory bandwidth and the cryptographic engine bandwidth. We replace the original DRAM memory bandwidth with this effective bandwidth for loopnest mappers. For example, Timeloop [52] directly uses the memory bandwidth when determining the number of cycles required for data transfer, and CoSA [105] can adjust the weighting parameter of its objective function reflecting the cost of data traffic. When the effective bandwidth is lower than the original DRAM bandwidth (i.e., the cryptographic engine has lower bandwidth), the mappers will prefer loopnests with a lower amount of off-chip data traffic (i.e., with higher computational intensity) although they might have lower PE utilization rate compared to the originally optimal loopnest. Such an approach is highly compatible with loopnest mappers 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 [101]).

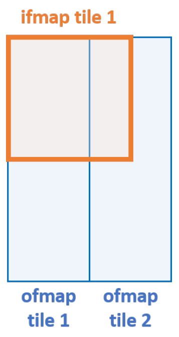
4.3.2 Optimal Authentication Block Assignment

In the second step of our mapping algorithm, 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.

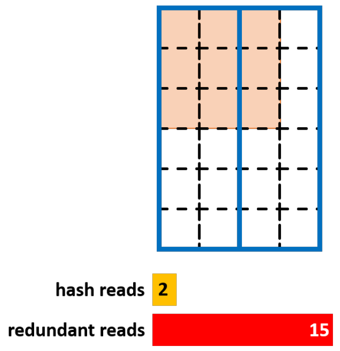
The Search Space of Authentication Blocks

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

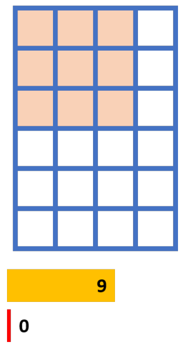
Given the nature of the memory authentication operation, it introduces additional off-chip memory accesses, classified into two categories. First, extra accesses to fetch the hashes.



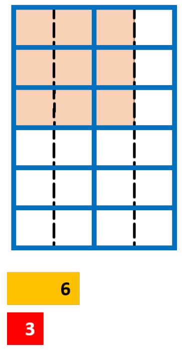
(a) Cross-layer Dependency



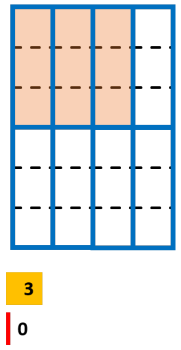
(b) tile-as-an-AuthBlock



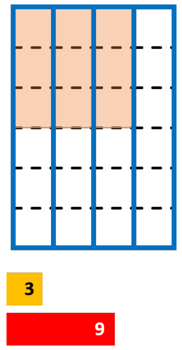
(c) horizontal, size: 1



(d) horizontal, size: 2



(e) vertical, size: 3



(f) vertical, size: 6

Figure 4.6: Examples of different AuthBlock assignments and their corresponding hash reads and redundant reads overhead. (a) reassembles the cross-layer dependency example discussed in Section 4.2.2. (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.

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. 4.6 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. 4.6(a) and (b), the example reassembles the cross-layer dependency case described in Section 4.2.2, 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. 4.6(c) and (d) compare 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 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. 4.6(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. 4.6(c). However, if we increase the size of the vertical AuthBlock to 6 in Fig. 4.6(f), the amount of redundant reads increases.

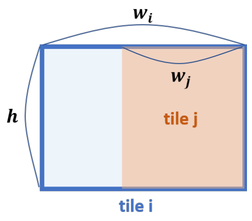
In summary, both the orientation and size of an AuthBlock affect the off-chip traffic overhead. To identify the optimal AuthBlock assignment, we can perform an exhaustive search over all feasible AuthBlock orientations and sizes. However, evaluating the additional off-chip traffic with a cycle-accurate simulation with the exact off-chip request sequence is unscalable considering the large search space for AuthBlocks. To overcome this issue, we present an analytical approach using a mathematical formulation of counting the additional off-chip traffic.

Also, we note that an AuthBlock assignment also affects the total number of cryptographic operations. The number of encryptions (for data write) and decryptions (for data read) will be proportional to the sum of baseline reads and redundant reads. When authentication requires different computations (e.g., Galois-field multiplication) for each 128-bit data (or the datapath size of a cryptographic engine), the number of such computations will be also proportional to the sum of baseline reads and redundant reads. If there are additional computations required to generate a hash tag at the start or end of authentication process (e.g., initialization or finalization step in some authenticated encryption algorithms [110]), the number of this additional computation will be proportional to the number of hash reads. Thus, minimizing the additional off-chip traffic can also generally reduce the total number of cryptographic operations.

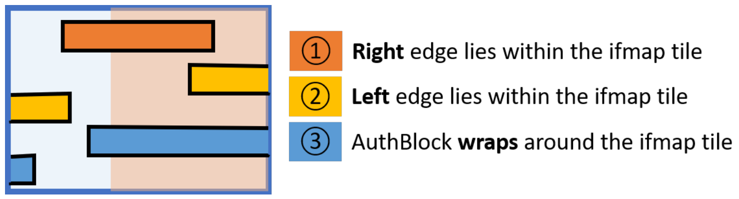
Mathematical Formulation

We now describe the mathematical formulation of the AuthBlock search process. Note that we discussed two cases where redundant reads occur in Section 4.2.2: 1) cross-layer dependency, and 2) halos. This mathematical formulation can be applied to both cases.

Problem In both cases, we are given a piece of tensor data and tile organizations associated with this data. For example, in the case of cross-layer dependency, we are given two tile organizations, one for the `ofmap` tiles and the other for the `ifmap` tiles, and there are overlaps between two different tile organizations. In the case of halos, we are given the `ifmap` tile organizations, and there are overlaps between different tiles in the same tile organization. We want to calculate the number of redundant reads and hash reads for each AuthBlock assignment, specified by the orientation and the size of AuthBlocks. Since each time an AuthBlock is accessed, all the elements in that AuthBlock need to be fetched together along with the hash associated with this AuthBlock, we can reduce the problem to counting *the number of AuthBlocks that overlap with each tile* loaded to the accelerator.



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


① Right edge lies within the ifmap tile
② Left edge lies within the ifmap tile
③ AuthBlock wraps around the ifmap tile

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

Figure 4.7: Mathematical formulation of counting redundant reads for a given AuthBlock assignment.

Conditions to Determine Overlap Consider an example shown in Fig. 4.7(a). This example shows a 2-dimensional tensor (i.e., matrix) 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$_i$. Note that this assumption is a natural scenario when tile$_i$ is an `ofmap` tile since hashes will be computed as the `ofmap` is generated. These AuthBlocks may not fully align with the boundary of tile$_j$ (e.g., an `ifmap` tile corresponding to the same tensor). Thus, if we want to calculate the additional off-chip traffic incurred when loading tile$_j$, we need to handle the case when the AuthBlocks cross the boundary of tile$_j$ to count all overlapping AuthBlocks.

The AuthBlocks can overlap with tile$_j$ in three conditions, as shown in Fig. 4.7(b). The first and second conditions are straightforward, where either the right edge or the left edge of an AuthBlock lies within tile$_j$. The third condition is when an AuthBlock wraps around tile$_j$, and both of its edges are in tile$_i$.

We can translate these three conditions into mathematical formulas. We label the left edge of an AuthBlock as (L_x, L_y), where L_x is the row (height) index of an AuthBlock in tile$_i$, and L_y is the column (width) index. Similarly, we can label the right edge of an AuthBlock as (R_x, R_y). Then, the first two conditions can be expressed as:

$$w_i - w_j \leq R_x < w_i \tag{4.1}$$

$$w_i - w_j \leq L_x < w_i \tag{4.2}$$

When the size of the AuthBlock is smaller than w_i, we can express the third condition as:

$$L_x < w_i - w_j \quad \wedge \quad R_x < w_i - w_j \quad \wedge \quad R_x < L_x \tag{4.3}$$

Note that if the size of the AuthBlock is equal to or larger than w_i, then it results in entire tile$_i$ to be fetched when loading tile$_j$, with the same amount of redundant reads and a potentially larger amount of hash reads as the tile-as-an-AuthBlock strategy. Thus, for the search process, we only have to consider the AuthBlock size smaller than w_i.

Linear Congruence For horizontal AuthBlocks in Fig. 4.7, each AuthBlock has a height of 1 and a width of u ($u < w_i$). Then, the left and right edges of the k-th AuthBlock in tile$_i$ can be derived as:

$$L_x^k = (u \times k) \bmod w_i \tag{4.4}$$

$$L_y^k = \lfloor \frac{u \times k}{w_i} \rfloor \tag{4.5}$$

$$R_x^k = (u \times k + (u - 1)) \bmod w_i \tag{4.6}$$

$$R_y^k = \lfloor \frac{u \times k + (u - 1)}{w_i} \rfloor \tag{4.7}$$

Then, we can plug these equations into the three conditions described above. For example, plugging Eq. (4.4) into Eq. (4.2) gives:

$$w_i - w_j \leq u \times k < w_i \bmod w_i \tag{4.8}$$

$$u \times k \equiv w_i - w_j, w_i - w_j + 1, \cdots, w_i - 1 \bmod w_i \tag{4.9}$$

Suppose we want to count how many AuthBlocks in tile$_i$ satisfy the second condition. This problem is then equivalent to counting how many ks ($0 \leq k < \lceil \frac{h \times w_i}{u} \rceil$) satisfy the linear congruence equation in Eq. (4.9). Note that this equation uses modular arithmetic, where $a \equiv b (\text{mod } c)$ means that the remainder of a divided by c and the remainder of b divided by c are equal.

Similarly, we can solve Eq. (4.1) and Eq. (4.3) for u and k, and the final linear congruence problem summarizing all three conditions is:

$$u \times k \equiv \min(w_i - w_j - u + 1, 0), \cdots, w_i - 1 \pmod{w_i} \tag{4.10}$$

Then the number of AuthBlocks overlapping with tile$_j$ is the number of ks satisfying the above equation. Therefore, we can convert the problem of counting overlapping AuthBlocks with a tile of our interest into a mathematical linear congruence problem that can be solved analytically without any cycle-level simulations.

A linear congruence problem in a format $a \times x \equiv b \bmod c$ can be solved efficiently using the extended Euclidean algorithm. The solver needs to find the greatest common denominator of a and c, which can be done in the logarithmic time of $\min(a, c)$. Eq. (4.10) requires at most w_i linear congruences to be solved. Thus, counting the overlapping AuthBlocks for a single tile can be done in a log-linear time of w_i. This analytical approach enables an exhaustive search over all feasible orientations and sizes, resolving the scalability issue with the simulation approach.

Generalization While we used a horizontal AuthBlock assignment and 2-dimensional tiles with the same height to illustrate the algorithm, this methodology is general enough to be applicable to vertical AuthBlock assignments and for higher-dimensional tiles with arbitrary overlapping patterns. First, consider a vertical AuthBlock assignment. We can apply the same analysis but with a different relationship between u and L_x, L_y, R_x, R_y. Essentially, the x-y coordinates will be transposed:

$$L_x^k = \lfloor \frac{u \times k}{h} \rfloor \tag{4.11}$$

$$L_y^k = (u \times k) \bmod h \tag{4.12}$$

$$R_x^k = \lfloor \frac{u \times k + (u-1)}{h} \rfloor \tag{4.13}$$

$$R_y^k = (u \times k + (u-1)) \bmod h \tag{4.14}$$

Consider the second condition in Eq. (4.2). Plugging the new L_x^k results in:

$$w_i - w_j \leq \lfloor \frac{u \times k}{h} \rfloor < w_i \tag{4.15}$$

$$(w_i - w_j) \times h \leq \lfloor \frac{u \times k}{h} \rfloor \times h < w_i \times h \tag{4.16}$$

Since $\lfloor \frac{u \times k}{h} \rfloor \times h = u \times k - (u \cdot k \bmod h) \leq u \times k$, the left-hand inequality becomes

$$(w_i - w_j) \times h \leq u \times k \tag{4.17}$$

Also, we know that $u \times k < h \times w_i$, hence the second condition boils down to:

$$(w_i - w_j) \times h \leq u \times k < w_i \times h \tag{4.18}$$

Note that this can be thought of as using Eq. (4.8) for a *reshaped* matrix, where the original 2-dimensional matrix is flattened in a column-major manner resulting in a matrix with a height of 1 and a width of $h \times w_i$.

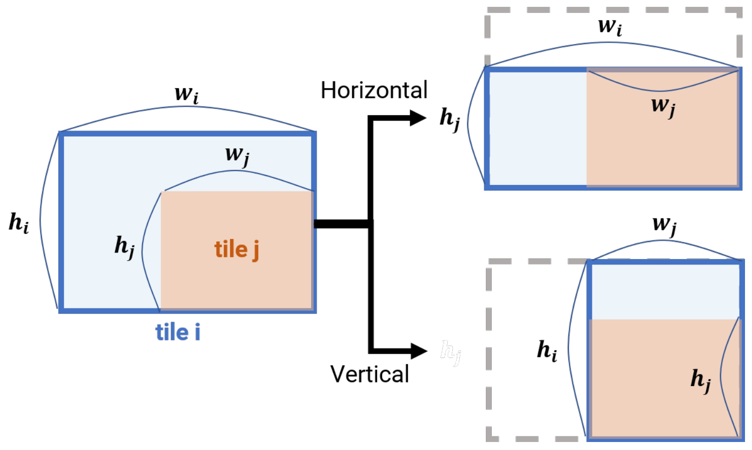


Figure 4.8: Even when two tiles have mismatches in multiple dimensions (two here), they can be boiled down to the example in Fig. 4.7 with maximum one authentication block miscounted.

More generally, consider n-dimensional tiles with a mismatch across a single dimension, $n^\star$ (e.g., width dimension in Fig. 4.7(a)). For simplicity of notation, we will assign numbers to denote dimensions: for a 2-dimensional matrix, dimension 0 refers to the height (row) dimension, and dimension 1 refers to the width (column) dimension. Similarly, for a 3-dimensional tensor, dimensions 0, 1, and 2 refer to the channel, height, and width dimensions, respectively. The AuthBlock orientation can be thought of as a permutation of these dimensions: in a 2-dimensional example, the horizontal orientation can be expressed as $(1, 0)$ (traverse through the width first, then, move to the next row), while the vertical orientation is $(0, 1)$. Suppose the AuthBlock orientation for n-dimensional tiles is $(p_1, \cdots, p_n)$ where $p_i \in [0, \cdots, n-1]$ for all is. There will be an index j with $p_j = n^\star$ that corresponds to the mismatching dimension. Then, for dimensions $i \leq j$, we can flatten them together and treat them as the equivalence of the 'width' in Fig. 4.7(a). Dimensions $i > j$ can be also flattened together and treated as the equivalence of the 'height'. Therefore, this mathematical approach can be generalized to higher-dimensional tiles.

Finally, we can also apply this approach when the tiles have mismatches in multiple dimensions, by reducing the tiles such that only a single mismatching dimension has to be considered. Fig. 4.8 shows 2-dimensional tiles that have mismatches in both the height and width dimensions. For a horizontal AuthBlock orientation, we can ignore the upper $h_i - h_j$ rows and only consider a smaller tile with the height of h_j and the width of w_i. This reduction only results in at most one AuthBlock miscounting at the horizontal boundary of tile$_j$. Similarly, we can ignore the left $w_i - w_j$ columns for a vertical AuthBlock orientation.

Intuitive Optimizations to Reduce Search Time In the case of a vertical AuthBlock orientation for Fig. 4.7(a), observe that there is at most a single AuthBlock that *partially* intersects with tile$_j$. This AuthBlock will be at the boundary of where tile$_j$ starts. Thus, the problem of counting redundant reads and hash reads can be further reduced using this property. The first $\lfloor \frac{h \times (w_i - w_j)}{u} \rfloor$ AuthBlocks in tile$_i$ will not overlap with tile$_j$ at all. The number of additional reads when loading tile$_j$ can be broken down into:

$$\text{redundant reads } = (h \times (w_i - w_j)) \bmod u \tag{4.19}$$

$$\text{hash reads } = \lceil \frac{(h \times (w_i - w_j)) \bmod u + h \times w_j}{u} \rceil \tag{4.20}$$

Also, when there is only a single dimension where tiles mismatch, generally it is optimal to set the AuthBlock orientation to visit that dimension lastly (e.g., use vertical orientation in Fig. 4.7(a)). Then, combined with the above-mentioned simplification, the optimal AuthBlock assignment can be boiled down to solving which value of u will minimize the sum of Eq. (4.19) and Eq. (4.20).

Example of Analysis Results

In Fig. 4.9, we visualize the search space of the AuthBlock assignment. The example follows the setup in Fig. 4.7(a) 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.

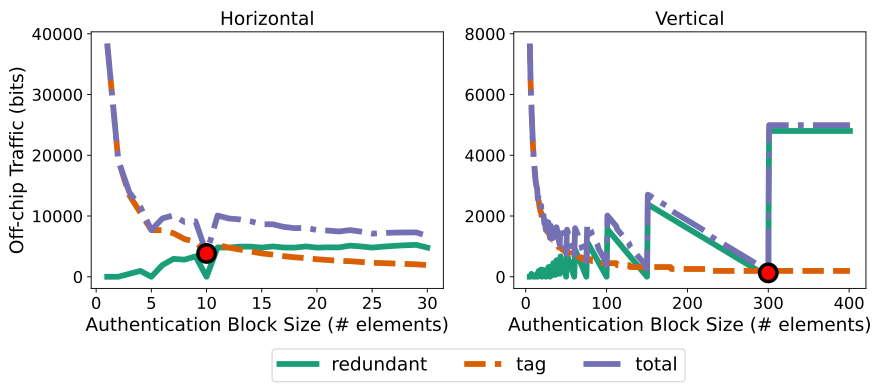


Figure 4.9: The amount of off-chip traffic incurred for accessing tile$_j$ in Fig. 4.7(a) when varying the AuthBlock orientation and size.

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. 4.7(a) 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.

4.3.3 Cross-layer Fine Tuning

So far, we derived the loopnest mappings in the first step for each individual layer in a DNN, then identified the optimal AuthBlock assignment based on the tiling organization defined by the loopnest mappings in the second step. However, the resulting mapping for secure DNN accelerators (i.e., loopnest mappings and AuthBlock assignments) may not be the *global* optimum considering the cross-layer dependency. For example, there can be loopnests for two dependent layers that are not optimal for each individual layer without considering the AuthBlocks, but result in better overall performance when both layers are considered together with the AuthBlock assignment. In the final step in our search algorithm, we directly consider this cross-layer dependency from the loopnest mapping level to find the globally optimal solution.

Challenges

Conventional mapping algorithms for DNN accelerators usually search for the optimal mapping for each layer independently. This is a natural assumption in the conventional setting, as each layer is executed sequentially and in-order in many cases, except for recent fused-layer processing techniques [111]. Furthermore, since each layer is independent of other layers for the purpose of mapping, the search space for mapping algorithms can be simplified. However, these mapping algorithms cannot be easily adapted to consider the influence of cryptographic operations with cross-layer dependency.

The main challenge is the exponential increase in the search space when multiple layers have to be searched jointly. Suppose the ith layer in a DNN has total s_i candidate loopnests. For brute-force search algorithms [52], [106], [107] that want to jointly search n layers (layers $1, 2, \cdots, n$), this means that the search space becomes $\prod_{i=1}^{n} s_i$. The computational complexity imposed by joint search can be prohibitive, especially for deep models with a large number of layers [60], [61].

Alternatively, optimization-based techniques are difficult to adapt due to the AuthBlock assignment. For example, the mathematics behind optimization techniques often require the constraints and the objective functions to be convex or linear [105], while the mathematical formulation for the AuthBlock assignment (Section 4.3.2) cannot be easily reduced to a closed-form nor be guaranteed to be convex.

Orthogonal to our case, there is recent work on jointly searching the mappings for multiple layers in the context of fused-layer processing [111]. These efforts tackle an orthogonal problem of fused-layer processing that aims to buffer the intermediate tensors to reduce the off-chip memory traffic for conventional DNN accelerators. In our case, the goal is to combine the impact of AuthBlock assignments with the mapping algorithm. Therefore, while these efforts are promising, they are orthogonal to the unique challenges of secure accelerators.

Search Using Simulated Annealing

We propose to use simulated annealing, a metaheuristic algorithm, to overcome the challenge of the large search space of cross-layer dependency. Simulated annealing is a probabilistic method for solving an optimization problem over a large search space. Although it does not guarantee the solution is optimal, it can efficiently search for a solution to a complex problem where the objective function and the constraints are not well-defined (or even unknown). Thus, simulated annealing can be used on top of the loopnest mapper and the AuthBlock assignment, without having to formulate these two previous steps into mathematically well-defined closed-form formulas.

Simulated annealing starts from an initial state, then iteratively searches for the *neighbors* of the current state and probabilistically decides whether to move on to the neighboring new state. The probability of moving is determined by a parameter called *temperature*, and the difference in the cost of the current and the new state. The temperature is gradually decreased throughout the iterations, such that suboptimal yet diverse states can be explored in the earlier iterations, while the best solutions can be fine-tuned in the later iterations.

Algorithm 2 describes our adaptation of a simulated annealing algorithm for the cross-layer fine-tuning. We denote L_i° as the optimal loopnest mapping of the i-th layer found from the first step without considering cross-layer dependency. Our algorithm attempts to identify a set of loopnests $(L_1, \cdots, L_n)$ that results in better performance compared to $(L_1^\circ, \cdots, L_n^\circ)$ when n layers are considered altogether with the AuthBlock assignment.

The algorithm starts by initializing the current set of loopnests 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

Algorithm 2 Pseudocode for step 3: simulated annealing

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

neighbor loopnest 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). Generically, neighbors can be defined as states with a small distance from the original state. However, there is no natural metric for measuring the 'distance' between two loopnests. We use the per-layer performance as the similarity metric. Specifically, we obtain top-k best loopnest mappings per layer from the first stage loopnest mapper, and the `GetNeighbor` function randomly samples among these k different loopnests to get a neighbor. When searching among k possible mappings for each layer, the search space has k^n distinct combinations to be explored in a limited number of simulated annealing iterations.

This choice has two key benefits. First, although the cross-layer dependency affects the global performance, still the individual layer performance is a key contributor to the global performance. For example, a per-layer loopnest mapping that results in low utilization of the PE array or excessive off-chip memory traffic is less likely to be a globally optimal solution anyway. Therefore, the top-k best neighbor set is an intuitive choice considering the importance of the per-layer performance. Second, obtaining top-k best loopnests only requires minor modifications to the conventional mappers like Timeloop [52]. This choice is thus highly compatible with the existing conventional mappers.

However, we also note this choice can result in less diverse candidate sets for the loopnest

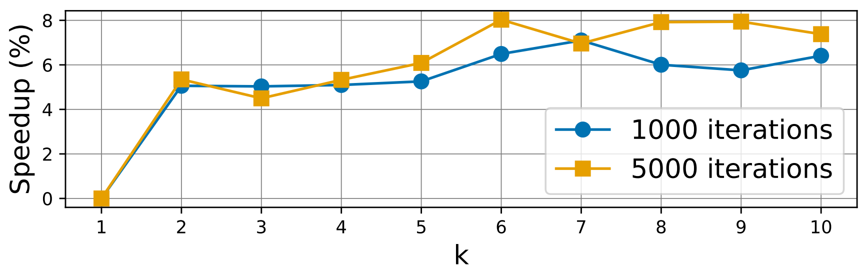


Figure 4.10: Improvement in latency (speedup) when using simulated annealing for different values of k, compared to when only the top-1 loopnest mapping for each layer.

mappings since the top-k best loopnests can have similar tiling strategies. Increasing k can be one potential method to consider more diverse loopnest candidates ($k \gg 10$), although at the cost of more iterations for simulated annealing to be converged.

The new mapping that replaces the loopnest mapping 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 Hyperparameters There are two key hyperparameters in Algorithm 2, the total number of iterations N and the size of the neighbor set k. We examine how these hyperparameters affect the search results. We demonstrate the performance improvement when the simulated annealing method is used with different values of k and N in Fig. 4.10. The numbers are for an architecture derived from Eyeriss [40] with a cryptographic engine with an energy-efficient AES-GCM implementation from [49] (detailed specifications in Table 4.2) running a MobilenetV2 [61] workload.

Increasing k from 1 to 2 can improve the overall performance by 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 We also need to consider post-processing operations that follow the intensive tensor algebra operations. Post-processing operations include non-linear activation functions, normalization, and pooling operations. Cross-layer dependency due to the intermediate tensors has to consider these post-processing operations. There are two types of post-processing operations.

First, some post-processing operations can be performed on-the-fly while the `ofmap` is being generated or while the `ifmap` is being loaded. These operations can be 'folded' with the tensor algebra operations, and we do not have to consider them separately. As a result, we should consider cross-layer dependency the same as when these operations do not exist. Examples of these operations are Batch Normalization [112], ReLU activation function, and adding zero pads. Note that if an accelerator design supports more complex operations on-the-fly with special functional units, those operations will also fall into this category.

Meanwhile, some other post-processing operations cannot be performed on-the-fly. For example, pooling operations and concatenation/addition of several feature maps for residual connections can be in this category depending on an accelerator design. These operations require a separate computation step and inevitably trigger rehashing. Thus, the cross-layer dependency problem due to AuthBlock assignment is not applicable for layers with these post-processing operations. As such, given a full DNN workload, we divide them into multiple segments based on the existence of the second type of post-processing operations and apply the AuthBlock assignment and the cross-layer fine-tuning within each segment.

4.4 Evaluation of the Mapping Algorithm

In this section, we present the effect of the mapping algorithms on the performance of a secure DNN accelerator. As a loopnest mapper for the first step, we use Timeloop [52]. We add an extension for Timeloop such that the top-k loopnests can be returned for each layer. The second and third steps, the AuthBlock assignment and cross-layer fine-tuning are

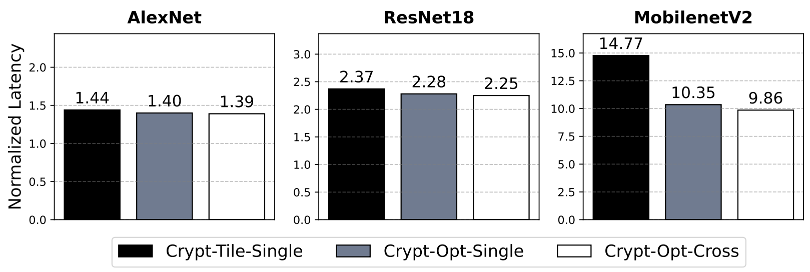
Table 4.1: Summary of different mapping algorithms.

Mapping Algorithm	Loopnest Mapper	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

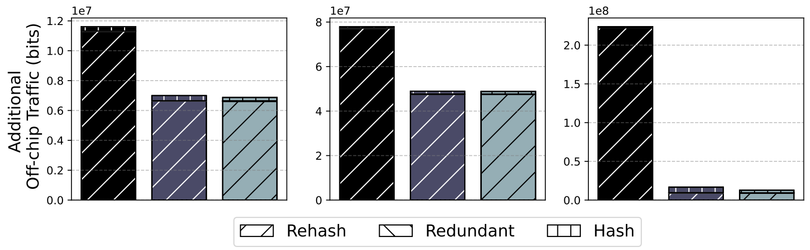
implemented to accept the loopnest mappings from the first step as inputs and generate the final loopnest mapping for each layer and the AuthBlock assignments. The performance is estimated using a simulator with an analytical approach that uses the action counts for each hardware component to estimate the number of cycles and the energy consumption. For a DNN accelerator, we can use Timeloop's `model` mode with Accelergy [113] to estimate the performance. For cryptographic engines, we calculate action counts for AES engines (for encryption) and Galois-field multiplier (for authentication) and estimate the performance using the reported throughput and energy consumption of AES-GCM designs in the literature [48], [49].

As a baseline algorithm, we consider *Crypt-Tile-Single*, which indicates that it uses Timeloop with the effective bandwidth and energy for the off-chip accesses reflecting the **crypt**ographic 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 sub-optimal loopnests degrading the baseline performance, especially when the cryptographic engine has lower throughput compared to the off-chip memory. We then add the second and third steps one by one, with the most optimized version denoted as *Crypt-Opt-Cross* with both the **opt**imal AuthBlock assignment and the **cross**-layer search enabled.

Base Configuration for Evaluation 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



(a) Performance overhead using different mapping 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 mapping algorithms.

Figure 4.11: Impacts of mapping algorithms on performance and off-chip traffic.

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 [40], 14×12 PEs, and 131kB on-chip global buffer.

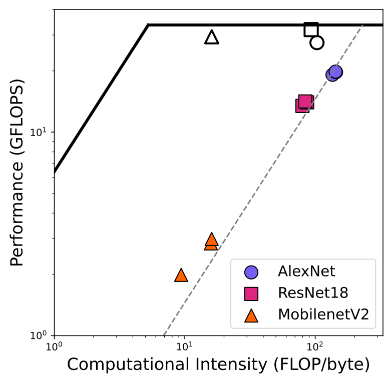
Furthermore, the secure accelerator uses an area-efficient parallel AES-GCM Implementation [49], [50] as its cryptographic engine, with one AES-GCM engine per each datatype. For the off-chip DRAM access, we assume LPDDR4 with a bandwidth of 64B per cycle. Finally, Accelergy [113] is used to estimate the energy and area of each component on the DNN accelerator, assuming 40/45nm technology it natively supports. For the cryptographic engines, we normalize the gate-equivalent area reported in the literature to 40nm technology to get their area [48], [49], and use the power and energy reported in [49].

Results Fig. 4.11(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. 4.11(b) shows the additional off-chip traffic incurred by cryptographic operations for each mapping algorithm. We examine three workloads with varying numbers of layers and characteristics: AlexNet [57], ResNet18 [60], and MobilenetV2 [61]. These workloads are mainly composed of 2-dimensional convolutions, and note that we only consider the first 5 layers of AlexNet that are convolutions.

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 an 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. 4.11(a) reports the mean value. This step does not significantly affect the performance on a shallower workload like AlexNet, where the opportunity for cross-layer optimization is limited. Nevertheless, it is worth noting 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 the baseline *Crypt-Tile-Single*.

Roofline Model We can also use the roofline model [67] to intuitively reason about the impact of scheduling algorithms (Fig. 4.12). In the left of Fig. 4.12, the roofline model de-



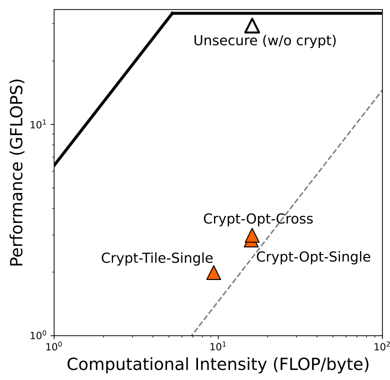


Figure 4.12: **Left:** Roofline model for accelerators using different mapping algorithms. White markers represent the unsecure baseline, and colored markers represent secure accelerators. **Right:** Roofline model zoomed in to show different mapping algorithms for the MobilenetV2 workload.

scribes 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-bound region for the unsecure baseline accelerator, but throttling from the cryptographic engine pushes the workloads to be in the effectively memory-bound region in secure accelerators. The right part of Fig. 4.12 zooms in to show the different mapping algorithms for the MobilenetV2 workload, and shows that each step in our mapper improves the performance by finding mappings with higher computational intensity.

Table 4.2: 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

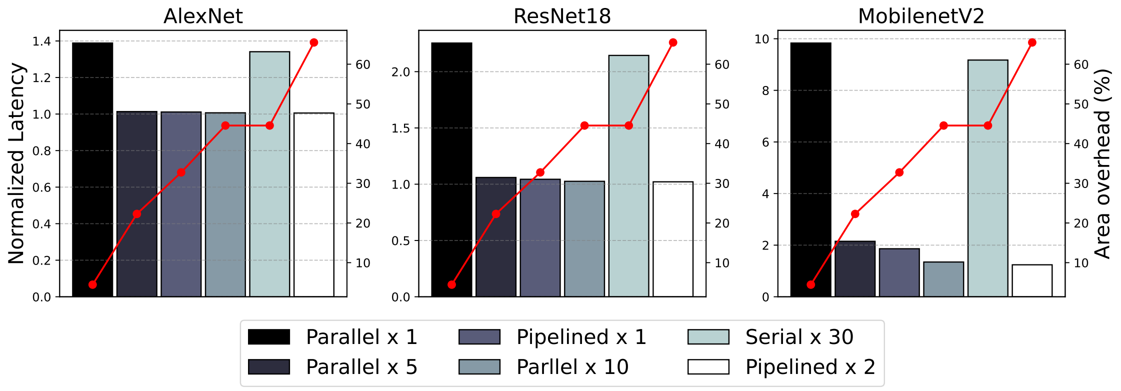


Figure 4.13: Slowdown over the unsecure baseline design and the area overhead of secure accelerators varying in their cryptographic engine configurations.

4.5 Impacts of Architecture Configurations

In this section, 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.

4.5.1 Cryptographic Engine

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 4.2. 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

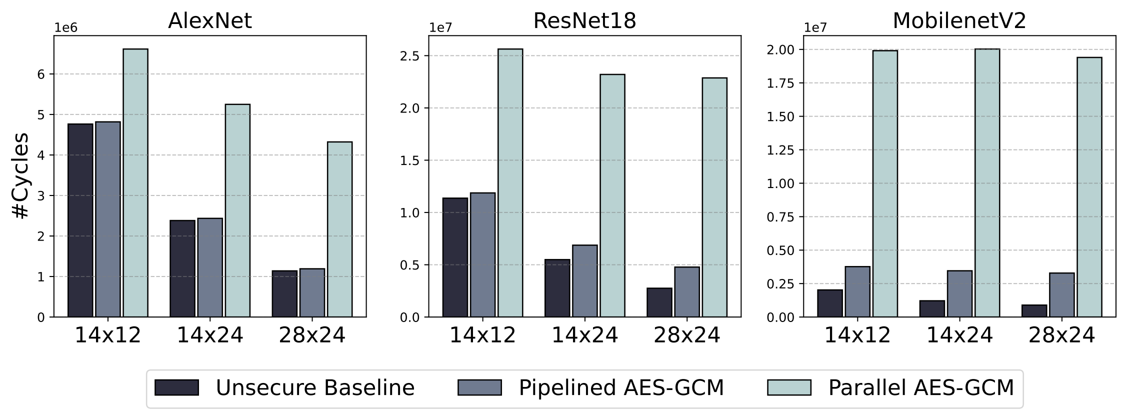


Figure 4.14: Latency for secure accelerator designs varying in their number of PEs.

low area overhead and low throughput. The parallel design is in between two other designs, with medium throughput and area overhead.

We use the same accelerator architecture as in Section 4.4 and use the *Crypt-Opt-Cross* mapping algorithm. Fig. 4.13 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 often using a moderate number of higher-throughput AES-GCM engines is a better design choice.

4.5.2 Processing Elements Array

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. 4.14 shows the evaluation result for different PE organizations $14 \times 12, 14 \times 24$, 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 manifest for the unsecure baseline accelerators (the latency decreases almost by half as the

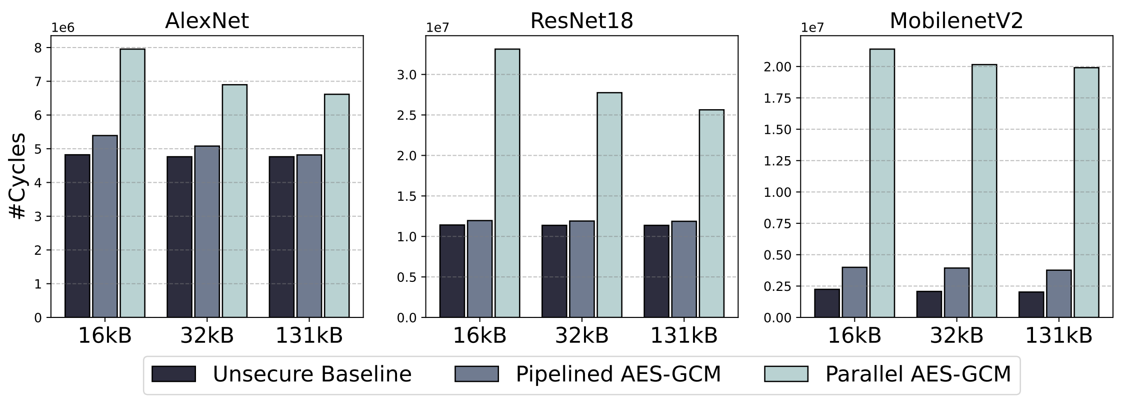


Figure 4.15: Latency of designs varying in the size of on-chip SRAM buffer.

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.

4.5.3 On-chip SRAM

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. 4.15, we examine the effect of different buffer sizes (131kB, 32kB, and 16kB) on secure accelerators while other design parameters 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 with limited encryption/decryption bandwidth, thus leading to longer latency for small buffer sizes.

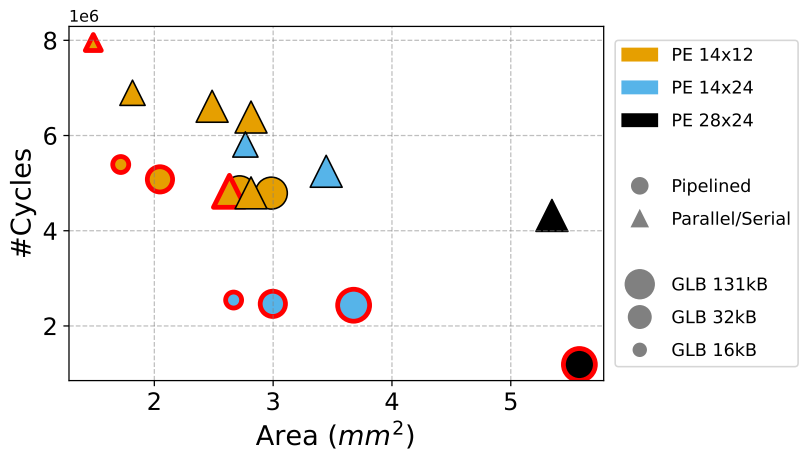


Figure 4.16: The area vs. performance trade-off of secure accelerator designs. Points highlighted with red edges indicate the Pareto front of this trade-off curve.

4.5.4 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. 4.16. We also derive the Pareto front of this trade-off curve and observe the 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. 4.15, 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 for the row-stationary design we considered in this section.

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. 4.14 that the benefit of having higher parallelism cannot be achieved when cryptographic engines are the bottleneck.

4.5.5 Discussions on Other Design Aspects

DRAM Technologies Higher off-chip memory bandwidth does not necessarily improve the performance of secure DNN accelerators when the effective off-chip bandwidth is limited by the cryptographic engine. For example, increasing the DRAM throughput to 128B/cycle does not change the latency and energy of secure DNN accelerators for the AlexNet workload. Thus, in order for secure DNN accelerators to fully utilize high memory bandwidth, the throughput of cryptographic engines should be improved as well.

However, the energy for the off-chip access can be affected by the DRAM technology even when secure DNN accelerators are throttled by the cryptographic engine. For example, using the HBM2 technology that has lower energy per access compared to LPDDR4, the energy for both the unsecured baseline and the secure accelerators decreases, although the latency is not affected.

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 [104] 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 DNN accelerators.

4.6 Summary

In this chapter, we present SecureLoop, a design space exploration framework for secure DNN accelerators supporting a TEE. Due to cryptographic authentication, a secure DNN accelerator accesses the off-chip memory using authentication blocks as the basic granularity. As a result, the authentication block assignment determines the off-chip memory access pattern and the overhead associated with cryptographic operations.

We first analyze the challenges for determining the optimal authentication block assignment. We note that the entire tile cannot be simply assigned as one authentication block, due to the cross-layer dependency in DNNs. While one layer's input feature map is generated as its previous layer's output feature map, and they are the same tensor data, the tiling strategy for each layer can be different. Thus, assigning the entire output feature map tile as an authentication block can result in significant additional off-chip traffic and more cryptographic operations.

To overcome this challenge, we propose a search algorithm for determining the optimal authentication block size and orientation. We show the problem of measuring the additional off-chip traffic depending on the authentication block assignment can be formulated as a mathematical problem, which can be efficiently solved. With this analytical approach, we can run an exhaustive search over all possible sizes and orientations of authentication blocks to identify the optimal one. We show that the optimal authentication block assignment can provide up to 30% performance improvement over assigning tiles as authentication blocks.

Next, we present an approach reflecting the impact of the authentication block assignment from the loopnest mapping stage. We generate the top-k best loopnest mappings for each layer in a DNN and use simulated annealing over those mappings to identify the global optimum considering the authentication block assignment. This cross-layer fine-tuning step provides up to 3% additional performance improvement.

Using the mapping algorithm of SecureLoop, we can evaluate the performance of a secure

DNN accelerator design more fairly. We present case studies on the design space exploration of secure DNN accelerators, varying the size of the PE array, the on-chip buffer, and the cryptographic engine design.

SecureLoop is the first to integrate security into the design space exploration of DNN accelerators and addresses several challenges with developing a mapping algorithm considering cryptographic operations. Furthermore, SecureLoop can provide a systematic evaluation of the cost associated with supporting a TEE in diverse DNN accelerator designs.

Chapter 5

Secure Off-chip Memory Interface for Deep Neural Network Inference Accelerators

In this chapter, we present an implementation of a secure DNN accelerator supporting a trusted execution environment (TEE) in silicon. From this implementation, we strive to achieve three objectives. First, we can verify the performance simulation result provided by SecureLoop (Chapter 4) using the cycle-accurate RTL simulation developed for this implementation. Second, we can understand the impact of implementation-level decisions, such as the control logic for cryptographic engines, which are difficult to capture from the high-level simulations. Finally, we demonstrate specific design choices tailored for resource-constrained edge devices, such as lightweight cryptography and layer fusion, showing that a TEE can be supported at a reasonable cost of $< 4\%$ performance slowdown, 16.5% more energy consumption, and 8.1% of the accelerator core area.

5.1 Secure DNN Accelerator Architecture Overview

Recall that secure DNN accelerators require every off-chip data traffic for input, weight, and output tensors to be encrypted and authenticated using a cryptographic primitive. This feature translated into two architecture-level differences from conventional DNN accelerators. First, the accelerator has to be equipped with hardware support for cryptographic operations. Second, the granularity of off-chip data access becomes authentication blocks, not tiles, changing the amount of off-chip traffic and mapping considerations. These two differences were key motivations for SecureLoop, a design space exploration framework for secure accelerators we presented in Chapter 4.

However, as a high-level performance simulator, SecureLoop evaluates the performance of a secure DNN accelerator design relying on key assumptions on its behavior. For the latency evaluation, SecureLoop assumes that the off-chip data transfer, cryptographic operations, and the actual DNN accelerator computations work in parallel with proper pipelining between the hardware units and double-buffering for the data access. Also, the latency for each hardware unit is evaluated based on the action counts, such as how many encryption operations are required for a DNN workload and how many bytes of data have to be loaded into the buffer. While these assumptions are in line with prior work on DNN accelerators and simulators [40], [52], the implementation of these functionalities ultimately determines the actual performance. Therefore, the implementation of a secure DNN accelerator can validate the assumptions of the performance simulator and demonstrate the actual overhead of secure designs.

Furthermore, there are unique challenges in the implementation in order to minimize the overhead of supporting a TEE. In this section, we outline the implementation-level challenges and design choices for secure off-chip memory interfaces in DNN accelerators.

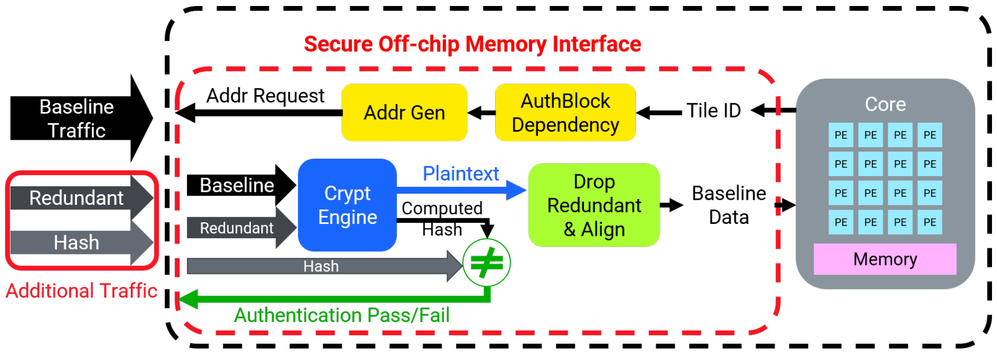


Figure 5.1: Overview of secure DNN accelerators supporting a TEE showing the necessary functionalities for a off-chip memory interface.

5.1.1 Challenges

Cryptographic Engine for Supporting a TEE

The first type of challenge arises from the difference in the control logic for the off-chip memory interface in secure DNN accelerators. Fig. 5.1 shows the block diagram of the off-chip memory control unit of a secure DNN accelerator. First, note that the granularity of the off-chip data request is authentication blocks, not regular tiles, since all data in one authentication block has to be fetched to compute the hash tag and verify the integrity. The memory control logic has to first translate the tile information to the dependent authentication blocks, and then track the addresses based on the authentication blocks. Then, when the data is fetched and decrypted, the control logic has to identify which elements are 'redundant' (i.e., data only required for authentication, not the actual computation) and drop those elements when writing to the global buffer.

Compared to conventional DNN accelerators where the off-chip memory control can be easily done with counters (e.g., increment the counter registers by a fixed amount until the counter reaches the tile size), the authentication-block-based control logic has to support additional operations described above. Our implementation aims to minimize the overhead from these additional operations to the off-chip memory control logic, such that the authentication-block-based control does not defeat the benefit of minimizing the additional off-chip traffic by optimizing the authentication block assignment.

Second, the global buffer memory map (i.e., the bank, row, and column information representing how each element in a tensor is stored in the buffer) and the authentication block orientation can misalign. For example, suppose a global buffer memory map requires one row in a 2-dimensional matrix to be stored in the same address so that all data in one row can be supplied in parallel for the computation. However, if the authentication blocks for this matrix are assigned in a column-major manner, the decrypted data cannot be written to the global buffer in a single cycle. Instead, the data has to be written in a transposed manner across multiple cycles.

This transpose problem increases the latency of data transfer in secure DNN accelerators. While one solution can be only allowing the authentication block orientation that matches the memory map constraints, it can result in a sub-optimal authentication block assignment increasing the redundant reads. Thus, our implementation aims to 'hide' this additional latency by leveraging the characteristics of the cryptographic operations, enabling a flexible authentication block assignment and good performance.

Resource-constrained Environment

The second type of challenge for our implementation is related to the resource-constrained environment for edge and mobile devices. Edge and mobile devices have limited area, power, and energy budget, hence the secure DNN accelerator has to be more conscious of the cost of security.

The large source of area overhead in secure accelerators is the cryptographic engine. For example, a widely-used authenticated encryption algorithm like AES-GCM can pose a significant area vs. performance trade-off for small accelerator designs as we described in Chapter 4. While AES is a standardized and well-validated cipher, an AES engine with sufficiently high throughput that does not throttle the off-chip data supply to a DNN accelerator can have a large area overhead. Thus, the overhead from the cryptographic engine itself can be challenging for a resource-constrained environment.

In our implementation, we reduce the overhead from cryptographic engines by adopting a recent lightweight cryptography algorithm [110] developed for edge and mobile devices, instead of using the conventional AES-GCM algorithm. This choice enables a better area vs. performance trade-off for secure accelerators while providing a sufficient level of security for DNN workloads.

Another challenge is different cryptographic engine throughput requirements depending on the workload characteristics. Compute-bound workloads, such as convolutions popularly used for image processing applications, require a relatively low throughput for off-chip data transfer, and using an area-efficient cryptographic engine can be acceptable. However, memory-bound workloads, such as matrix multiplications and self-attention modules in Transformer models [63], [64], [114], require much higher off-chip data transfer throughput and a low throughput cryptographic engine can significantly slow down the accelerator performance. Therefore, a cryptographic engine design tailored for one type of workload can be suboptimal for the other. For a resource-constrained accelerator, over-provisioning cryptographic engines for memory-bound workloads can incur large area overhead not acceptable for the overall design cost.

We aim to overcome this challenge by leveraging layer fusion [115], [116] that can convert memory-bound workloads to effectively compute-bound workloads. A key observation is that the widely-used memory-bound workloads like self-attention modules [63], [64], [114] have large memory footprints due to the intermediate activation tensors whose data traffic can be eliminated with layer fusion.

5.2 Secure Off-chip Memory Interface

In this section, we describe the implementation of the secure off-chip memory interface for DNN accelerators supporting a TEE. The configuration for a secure DNN accelerator

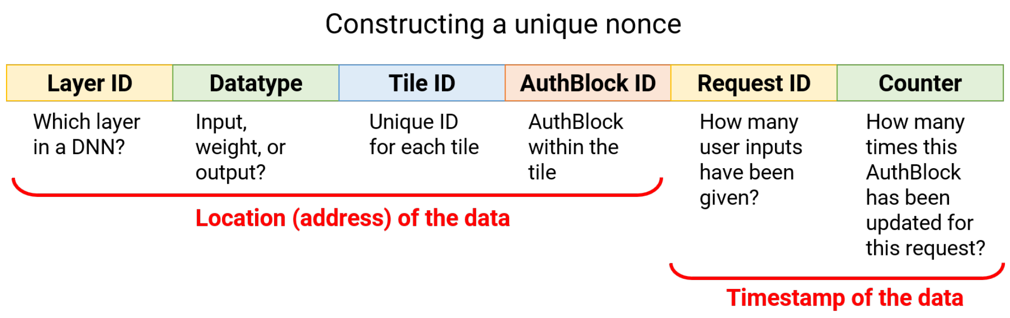

Constructing a unique nonce
Layer ID
Datatype
Tile ID
AuthBlock ID
Request ID
Counter
Which layer in a DNN?
Input, weight, or output?
Unique ID for each tile
AuthBlock within the tile
How many user inputs have been given?
How many times this AuthBlock has been updated for this request?
Location (address) of the data
Timestamp of the data

Figure 5.2: Constructing a unique nonce requires the address and the timestamp.

includes information for a DNN workload, a loopnest mapping specifying the tiling strategy and the computation order, and authentication block assignments specifying the size and orientation of authentication blocks for each datatype. Given this configuration, the main controller of an accelerator tracks the current and next tile information for each datatype. The tiles at the DRAM level are used for the off-chip data requests. From the tile information provided by the main controller, the off-chip memory interface handles the address tracking and generation for data requests and transfers the data between the accelerator and the off-chip memory.

Our secure off-chip memory interface implementation has four key contributions, addressing the challenges described in Section 5.1.1 and providing support for a TEE. First, we describe the nonce and counter management, which are essential to guarantee the security of cryptographic operations. Second, the authentication-block-based control logic is efficiently implemented with a small area and latency overhead compared to the conventional tile-based control logic. Third, we show that the transpose problem can be resolved by interleaving the data write operations with the cryptographic engine's internal operations and by adding a small transpose buffer. Finally, we adopt a recent NIST-standardized lightweight authenticated encryption algorithm, ASCON [110], as a core cryptographic engine to reduce the area overhead.

5.2.1 Managing Nonces

In many widely-used authenticated encryption algorithms, such as AES-GCM and ASCON, a *key* and a cryptographic *nonce* (i.e., numbers that are only used once) are required for the initialization. A secret key is always kept on-chip in a special register that other non-cryptographic operations cannot access, and the same key is used throughout the entire session (i.e., a user requests processing of a certain DNN workload). A cryptographic nonce has to be unique for every different authenticated encryption to guarantee the security of cryptographic primitives. The same nonce should not be repeated to encrypt different data using the same key. The secrecy of the key and the uniqueness of the (key, nonce) pair are crucial for security.

A nonce can be designed to be unique for every encryption by tracking the address and the timestamp for the data (Fig. 5.2). The timestamp is updated when the data in a particular address has to be updated and the corresponding nonce is different from the previous version. In secure DNN accelerators, the address can simply be the logical DRAM address or determined by the tile and authentication block information. For example, if the initialization has to be performed for each authentication block, the address field of the nonce can be uniquely determined by a layer id (e.g., which layer in a DNN workload is being processed), a datatype (e.g., input, weight, or output tensor), a tile id (e.g., which tile in a tensor), and an authentication block id (e.g., which authentication block within a tile).

The timestamp has to be updated when the data in a specific address is updated. Weight tensors are read-only and never updated during the inference unless the run-time fine-tuning is used, and the timestamp can be fixed to a single value when entering the TEE mode. Input and output tensors change for each request (e.g., the number of input data a user has requested for processing so far). Output tensors can be updated multiple times if partial intermediate tensors have to be written back to the off-chip memory when processing a single layer. While input tensors are read-only for a single layer, they are dependent on the

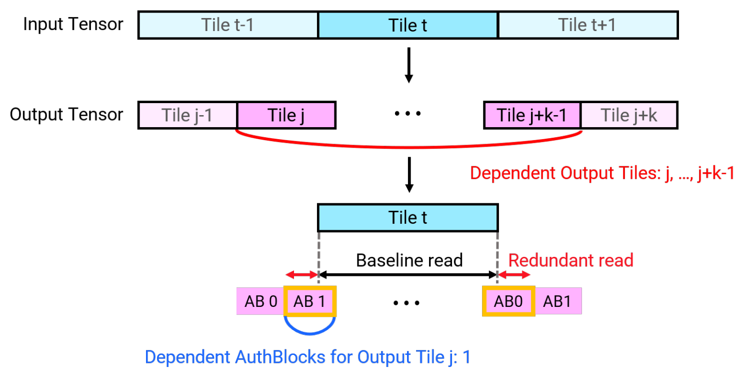


Figure 5.3: A simple 1-dimensional example to visualize the authentication-block-based off-chip memory access control.

previous layer's output tensors. Thus, the timestamp for input and output tensors has to keep two fields: a request id and a counter.

A request id can be easily tracked with a counter that is only updated when the processing of the entire DNN workload on the current input data is completed. A counter tracks the number of updates for a specific authentication block within an output tile. Note that in DNN accelerators, this update can be pre-determined by the loopnest mapping, without having to store counters separately [102], [104]. Overall, managing nonces can be achieved with a single counter tracking a request id, and the tile and authentication block information computed from the loopnest mapping and the cryptographic configuration.

5.2.2 Authentication Blocks

As we described earlier, the previous layer's output tensor is used as the current layer's input tensor. This cross-layer dependency results in the misalignment of authentication blocks, which are assigned when generating the output tensor, and the tiles in the input tensor (Chapter 4). Therefore, when loading an input tile, the off-chip memory interface has to identify all dependent authentication blocks of this tile, and load all data in those authentication blocks.

This authentication-block-based control is described in Algorithm 3 and Fig. 5.3. For simplicity, we explain this algorithm for a 1-dimensional tensor (i.e., a vector) here. The

Algorithm 3 Tracking off-chip data requests based on authentication blocks

```
Input: Input tile id $t$, Input tile size $s_i$
Input: Output tile size $s_o$, Output tile authentication block size $s_a \leq s_o$
    ▷ Determine the start and end indices of the requested input tile
 1: Input tile start index $i_{\text{start}} \leftarrow t \times s_i$
 2: Input tile end index $i_{\text{end}} \leftarrow (t+1) \times s_i$
    ▷ Determine the dependent output tiles
 3: $x_1 \leftarrow \lfloor \frac{i_{\text{start}}}{s_o} \rfloor$          ▷ The leftmost dependent output tile
 4: $y_1 \leftarrow i_{\text{start}} \% s_o$
 5: $x_2 \leftarrow \lfloor \frac{i_{\text{end}}}{s_o} \rfloor$          ▷ The rightmost dependent output tile
 6: $y_2 \leftarrow i_{\text{end}} \% s_o$
    ▷ Determine the dependent authentication blocks
 7: $a_1 \leftarrow \lfloor \frac{y_1}{s_a} \rfloor$     ▷ The leftmost authentication block in the leftmost dependent output tile
 8: $a_2 \leftarrow \lfloor \frac{y_2}{s_a} \rfloor$     ▷ The rightmost authentication block in the rightmost dependent output tile
    ▷ Tracking addresses
 9: for output tile id $t_o \in [x_1, \cdots, x_2]$ do          ▷ Iterate through the dependent output tiles
    ▷ Determine the dependent authentication blocks in this output tile
10:     if $t_o = x_1$ then
11:         $a_{\text{start}} \leftarrow a_1$
12:     else
13:         $a_{\text{start}} \leftarrow 0$
14:     if $t_o = x_2$ then
15:         $a_{\text{end}} \leftarrow a_2$
16:     else
17:         $a_{\text{end}} \leftarrow \lceil \frac{s_o}{s_a} \rceil$
    ▷ Iterate through the dependent authentication blocks
18:     for authentication block id $a \in [a_{\text{start}}, \cdots, a_{\text{end}}]$ do
19:         for $k \in [0, s_a)$ do          ▷ Iterate through each element in an authentication block
20:             $i_{\text{curr}} \leftarrow t_o \times s_o + a \times s_a + k$
21:             if $i_{\text{start}} \leq i_{\text{curr}} < i_{\text{end}}$ then
22:                 WRITE_DATA_TO_SRAM
23:             else
24:                 DROP_REDUNDANT_READ
```

authentication blocks are assigned as a 'sub-tile' of an output tensor tile. However, note that this algorithm can be extended to higher-dimensional tensors.

The control logic takes an input tensor tile id, an input tensor tile size, the previous layer's output tensor tile size, and the authentication block size as input, and returns the address sequence, which is used for the off-chip memory requests and tracking the received data, and the data that will be written to the accelerator's memory hierarchy. First, the start and the end indices of an input tensor tile is computed using an input tensor tile id and its size (Algorithm 3 line 1-2). Next, for this input tensor tile, we identify all its dependent output tensor tiles by dividing the start and the end indices by the size of the output tensor tile (line 3-6). Since tiles are continuous (i.e., all elements between their start and end indices belong to a tile), only computing for the two edges of a tile is sufficient. The remainder of the previous division is further used to identify the authentication blocks within the tiles. In the first tile (leftmost), the remainder is divided by the authentication block size, and the quotient indicates the starting authentication block id within this output tensor tile (line 7). For the last tile (rightmost), the quotient indicates the last authentication block id within the tile (line 8).

Using the dependent output tile ids and the authentication block ids with each tile, we can generate the address sequence with for-loops. The first counter tracks the dependent tile ids (line 9) and the second counter tracks the authentication blocks within each tile (line 10). Note that the authentication block start and end indices can be different for the first and the last dependent output tensor tile. When the off-chip memory returns the data for the requested address and the cryptographic engine completes decryption, we determine if this data is 'redundant read' (i.e., data only needed for authentication, not for the actual computation), we drop the data (line 20-24). This last step can be determined by comparing the start index of an input tensor tile with the start index of the first authentication block, and similarly for the last authentication block.

Compared to the conventional off-chip memory control, the authentication-block-based control requires some additional cycles and hardware resources to compute the dependent output tiles and authentication blocks. For example, using two integer dividers, line 3-6 will take a single cycle. For the address sequence generation, this control logic requires one additional for-loop to track the authentication blocks within each output tensor tile, resulting in an additional counter. However, the area overhead is limited to a few dividers, counters, and comparators, and configuring the dependent authentication blocks (line 1-8) has to be performed only once per each input tensor tile, limiting the latency overhead to just a few additional cycles. Therefore, the authentication-block-based control can be efficiently supported for DNN accelerators, and the flexible authentication block assignment (i.e., not simply assigning the entire tile as an authentication block) can reduce the overhead for the off-chip data access and cryptographic operations without incurring a significant overhead to the control logic.

5.2.3 Memory Map and Transpose Problem

In the high-level abstraction, the transpose problem can pose a significant overhead as it can limit the on-chip global buffer data write to just one word per cycle in the worst case. Consider a block cipher that performs encryption and decryption on a 128-bit block of data. For a word size of 16-bit, the worst-case latency for writing this cipher block to the global buffer is 8 cycles.

A closer look at the implementation of the cryptographic engine and the data write operations provides opportunities to reduce this overhead by ‘hiding’ the latency and using a small buffer. When an encrypted cipher block is received, a cryptographic engine has to update its internal state before accepting the next cipher block. For example, a counter-mode AES will require several cycles to generate the one-time pad for the next cipher block. Interleaving the writing of the decrypted data with this internal state update can hide the worst-case latency of the transpose problem. In our example, if the internal state update requires more than 8 cycles, the latency can be completely hidden by interleaving.

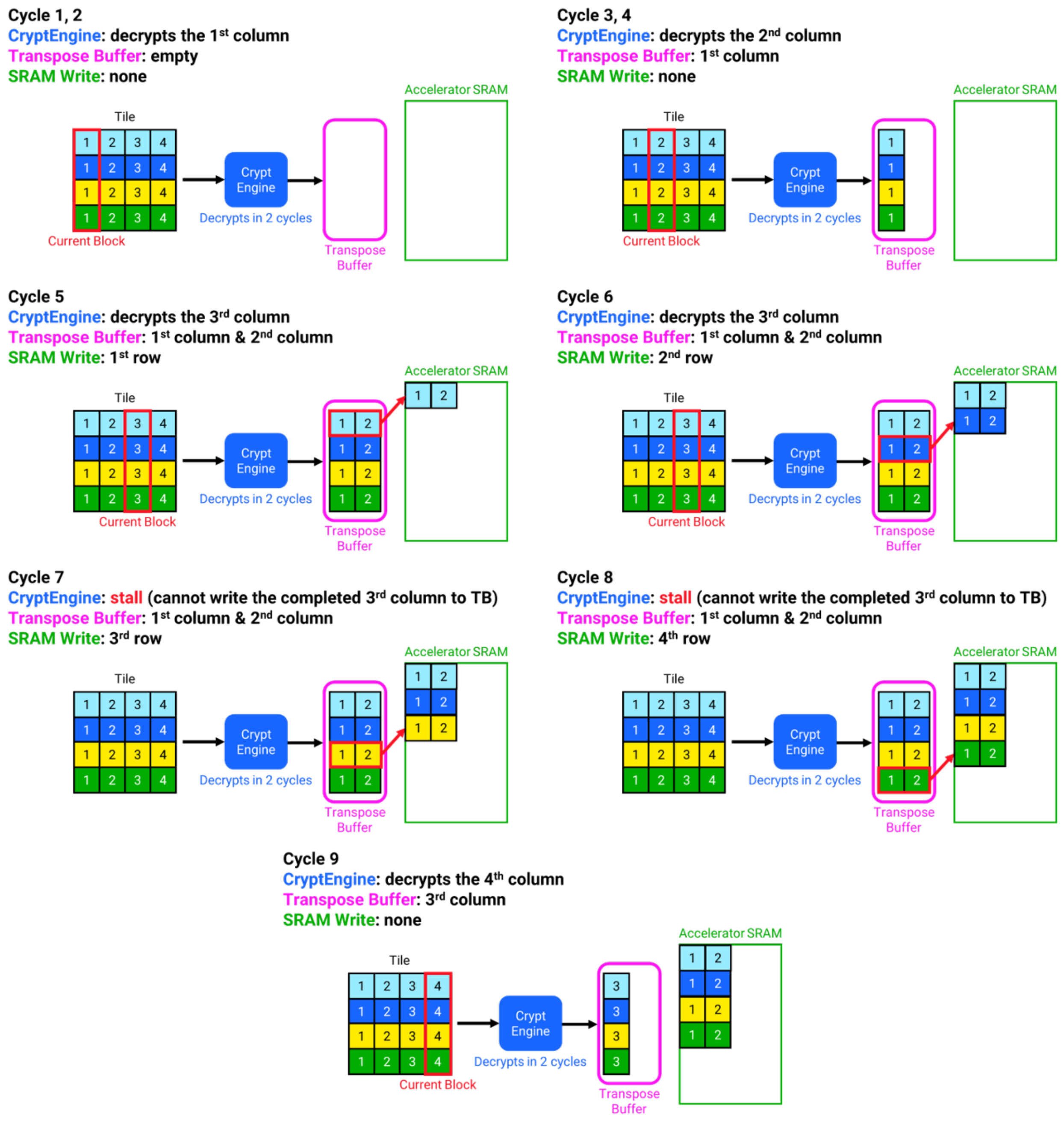

Cycle 1, 2
CryptEngine: decrypts the 1st column
Transpose Buffer: empty
SRAM Write: none
Cycle 3, 4
CryptEngine: decrypts the 2nd column
Transpose Buffer: 1st column
SRAM Write: none
Cycle 5
CryptEngine: decrypts the 3rd column
Transpose Buffer: 1st column & 2nd column
SRAM Write: 1st row
Cycle 6
CryptEngine: decrypts the 3rd column
Transpose Buffer: 1st column & 2nd column
SRAM Write: 2nd row
Cycle 7
CryptEngine: stall (cannot write the completed 3rd column to TB)
Transpose Buffer: 1st column & 2nd column
SRAM Write: 3rd row
Cycle 8
CryptEngine: stall (cannot write the completed 3rd column to TB)
Transpose Buffer: 1st column & 2nd column
SRAM Write: 4th row
Cycle 9
CryptEngine: decrypts the 4th column
Transpose Buffer: 3rd column
SRAM Write: none
Tile
Current Block
Crypt Engine
Decrypts in 2 cycles
Transpose Buffer
Accelerator SRAM

Figure 5.4: Introducing a transpose buffer can reduce the stalls due to the SRAM write operation. In this example, a transpose buffer that can hold two columns can initiate SRAM write for every two decryptions, reducing the stall cycles to two cycles per two decryptions (instead of only one decryption).

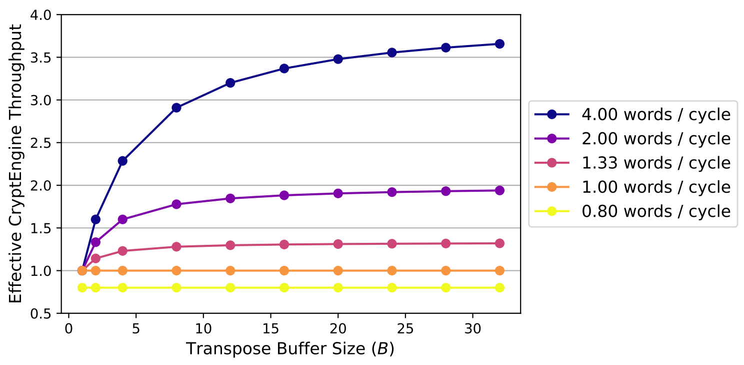


Figure 5.5: The trade-off between the effective cryptographic engine bandwidth considering the stalls from transposed writes (y-axis) and the size of the transpose buffer (B) representing the number of words that can be written in parallel (x-axis). Different lines represent cryptographic engines with the baseline throughput stated in the legend.

However, if the cryptographic engine throughput is higher and the internal state update can be done in less than 8 cycles, the latency cannot be fully hidden and the next data decryption has to be stalled until the current data is fully transferred to the on-chip buffer. Here the transpose problem limits the effective throughput of cryptographic engines.

To alleviate this problem, we introduce a small transpose buffer that temporarily stores decrypted data before initiating write to the global buffer. Suppose a buffer can store two cipher blocks (256-bits or 32B). Since two elements can be written to the same row in the global buffer, the data write can be initiated for every two decryptions instead of one. Then, the stall will only happen every two rounds of decryptions, improving the cryptographic engine throughput (Fig. 5.4).

More generally, suppose one cipher block has N_w words (e.g., 8 words for a 128-bit block and a 16-bit word size) and the internal state update requires k cycles. If a transpose buffer can store $N_w \times B$ words where B words can be written in parallel to the global buffer, then the effective cryptographic engine throughput becomes:

$$\text{throughput} = \begin{cases} \frac{N_w}{k} & \text{for} \quad k \geq N_w \\ \frac{N_w}{k + \frac{N_w - k}{B}} & \text{for} \quad k < N_w \end{cases} \tag{5.1}$$

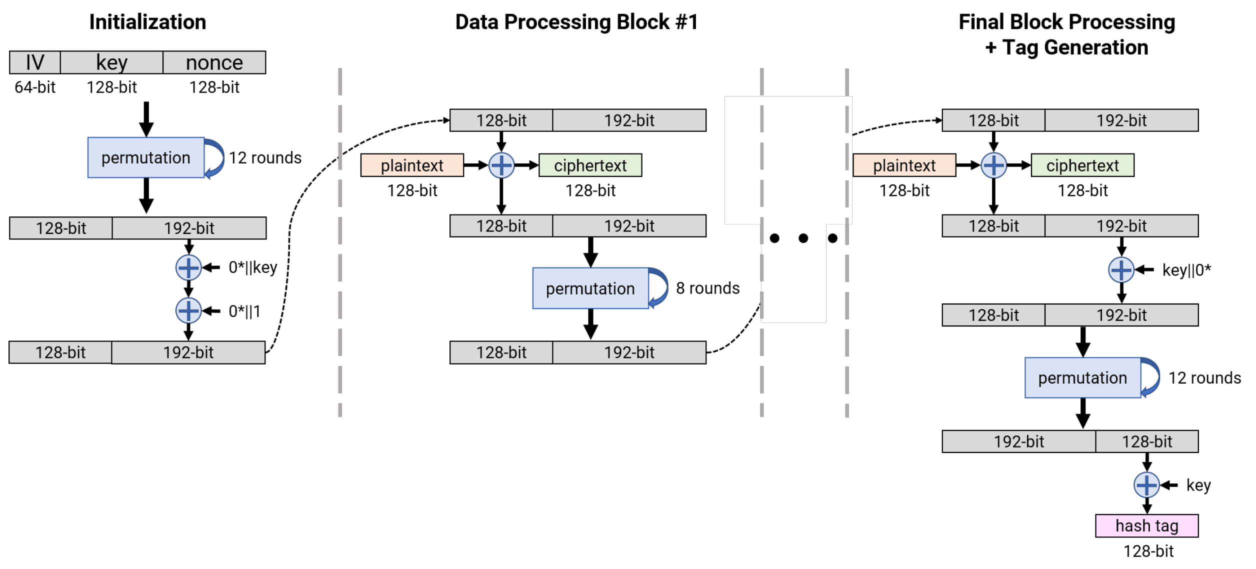


Figure 5.6: Cryptographic operations for each authentication block using ASCON-128a. Here we depict the encryption of plaintext data to generate ciphertext data. The decryption of ciphertext data can be done similarly. IV refers to an initialization vector as defined in [110].

For the case when $N_w = 8$, Fig. 5.5 shows the relationship between the throughput of a cryptographic engine and the size of a transpose buffer. We can observe that as a transpose buffer capacity increases, the throughput converges to the ideal cryptographic engine throughput when the transpose problem does not exist. In practical implementation, a designer can choose the desired throughput and area trade-off point. Thus, the transpose problem of the cryptographic engine can be solved with a small transpose buffer and careful interleaving of operations, without having to provide any specialized memory design for transpose [117].

5.2.4 Lightweight Cryptography

In many prior work for a TEE, Advanced Encryption Standard (AES) [118] is used as a cryptographic encryption algorithm [94], [96], [97]. AES can be a good fit for general-purpose processors especially when an instruction set architecture provides dedicated instructions for AES [119] and the latency of encryptions/decryptions can be hidden with the counter mode operation [96].

However, domain-specific DNN accelerators have differences from those of general-purpose processors. First, hardware support for cryptographic operations should be implemented for a TEE in DNN accelerators, and the cost (i.e., area, energy, power) of this hardware cryptographic engine can be significant for edge accelerator designs. In AES, the non-linear substitution operations involving Galois-field arithmetic can be expensive in hardware and pose a significant performance vs. area trade-off for a small DNN accelerator design as we discussed in Chapter 4. Thus, a cryptographic primitive with better area, power, and energy efficiency can be a better alternative for DNN accelerators. Second, the off-chip data access for DNN accelerators is generally throughput-sensitive not necessarily latency-sensitive, as the latency can be hidden with double-buffering and pre-determined explicit data orchestration, reducing the benefit of the counter-mode AES.

In this implementation, we adopt ASCON (more specifically, ASCON-128a variant) as a cryptographic primitive for authenticated encryption [110], which has a simple design (e.g., bitwise operations and rotations) and comparable security guarantee to AES. Furthermore, ASCON has been recently selected as a standard for NIST Lightweight Cryptography [120], showing its future potential to be widely adopted for edge applications.

Fig. 5.6 shows the operations required to encrypt and generate a hash tag for each authentication block. For each authentication block, ASCON takes a unique (key, nonce) pair to initialize its state. A key is fixed throughout one session and a nonce is determined based on the authentication block information (Section 5.2.1). Once the initialization step is completed (note that we do not use associated data field), ASCON processes each 128-bit plaintext (for encryption) or ciphertext (for decryption) block by XOR-ing the given data with its internal state and updating its own internal state afterward. When it finishes updating its own state, ASCON can accept the next 128-bit block of data. When all data in an authentication block has been processed, ASCON generates a hash tag.

The initialization, data processing, and tag generation in ASCON all rely on the same permutation operation, but with different numbers of rounds. Here, the latency of an ASCON

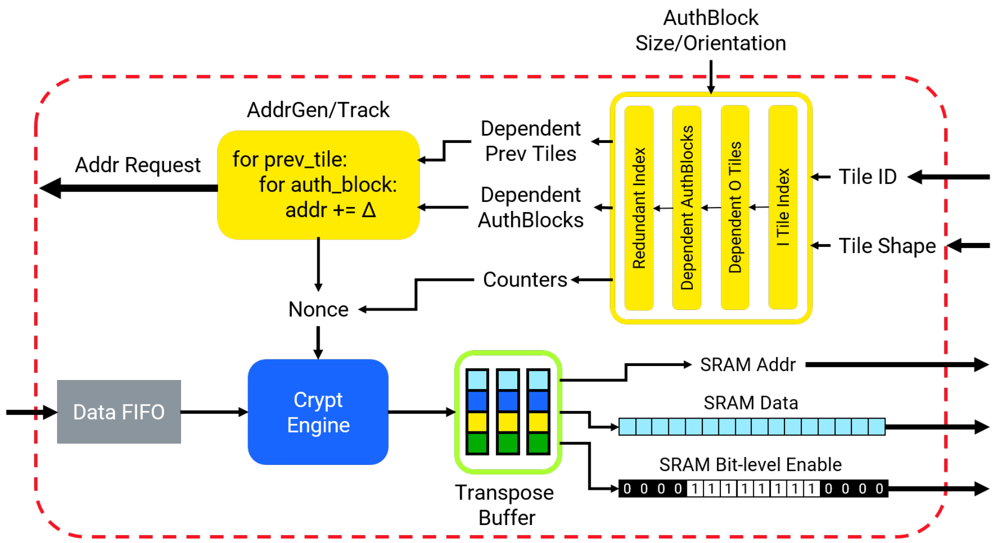


Figure 5.7: A block diagram of the secure off-chip memory interface supporting 1) the off-chip address tracking based on authentication blocks, 2) authenticated encryption on the off-chip data, and 3) alignment of the decrypted data with the on-chip SRAM address map.

cryptographic engine depends on how many permutation rounds can be processed in a single clock cycle. In order to reduce the latency and increase the throughput, the combinatorial logic should implement the unrolled permutation rounds with more logic gates.

5.2.5 Putting Together

Fig. 5.7 shows the design of our secure off-chip memory interface for input tensors. First, from the tile id requested by the main controller of a DNN accelerator, our interface computes the dependent previous layer tile information (i.e., each tile in an input tensor has dependent tiles in the previous layer's output tensor) and the authentication block information (i.e., which authentication blocks in each dependent output tile are relevant). The nonce can be generated using the tile and authentication block information (Section 5.2.1), and addresses for the off-chip accesses can be tracked using one additional counter for authentication block (Section 5.2.2).

The ASCON cryptographic engine decrypts the data that is available in the data FIFO only when 1) the data FIFO is not empty, 2) the initialization step is completed when the nonce is changed (i.e., moving to the next authentication block), 3) its internal state update is completed, and 4) the transpose buffer is not full. The decrypted data is pushed to the transpose buffer, and the transpose buffer initiates the on-chip SRAM write when it is full or when the current authentication block is finished. We compute the SRAM address and the bit-level write enable control signal for each write operation, and the redundant data can be dropped using this enable signal. When the entire authentication block has been processed, the cryptographic engine computes a hash tag. If the computed tag does not match the fetched tag from the off-chip memory, this interface module will flag the authentication failure and reset a DNN accelerator processing.

Note that the effective data supply rate to a DNN accelerator is determined by three factors: 1) the baseline off-chip data bandwidth determining how fast the data FIFO can be filled, 2) the cryptographic engine throughput affecting how fast decryption can be done, and 3) the transpose buffer size deciding the effective SRAM write bandwidth (only when the authentication block orientation requires transposed write). The slowest of these three components will throttle the data supply.

Finally, although we showed the interface module for input tensors, this design can be applied to weight tensors and output tensors with modifications. Since weight tensors do not have the cross-layer dependency problem, the dependency computation and the transpose buffer can be eliminated. For output tensors, the data direction has to be reversed (encryption instead of decryption). If we limit authentication blocks to be sub-tiles of each tensor in an output tensor, authentication blocks can be easily tracked with a counter in the address tracking module, and the transpose buffer can be replaced by a simple data FIFO between the interface module and a DNN accelerator.

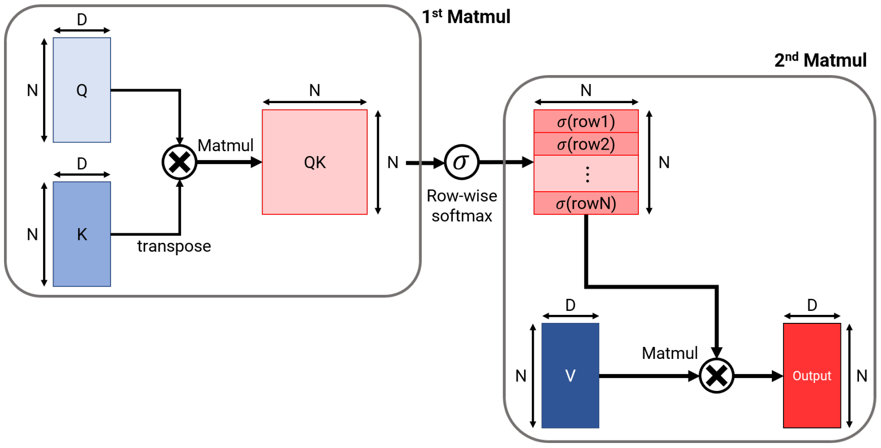


Figure 5.8: Two matrix multiplications in a self-attention module are often memory-bound due to the large intermediate tensor (QK). This QK tensor is the output of the first matrix multiplication, and serves as the input to the second matrix multiplication.

5.3 Cryptographic Engine Throughput Requirement and Fused-layer Processing

To reduce the overhead for memory-bound workloads, we adopt fused-layer processing as a workaround to convert memory-bound workloads to effectively compute-bound workloads. Fused-layer processing can help reduce (or completely eliminate) the off-chip data movement of intermediate activation tensors between two layers in a DNN [111], [115], [116], [121]. When a workload is memory-bound due to large intermediate tensors, as in the case of two batched matrix-matrix multiplications in a self-attention module (Fig. 5.8), fused-layer processing can change the workload to be effectively compute-bound. Fused-layer processing has been well investigated in prior work for accelerating a self-attention module in GPUs [115] and domain-specific accelerators [116]. For secure DNN accelerators, fused-layer processing can bring the additional benefit of reducing the overhead of cryptographic engines since low-throughput and area-efficient cryptographic engines can be used for compute-bound workloads.

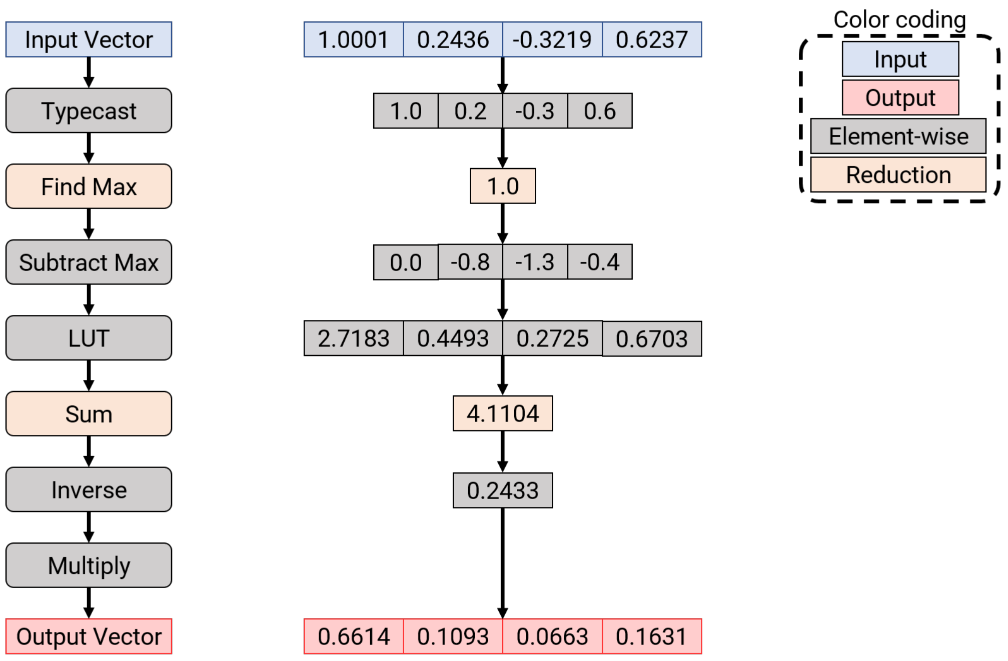


Figure 5.9: The hardware implementation of softmax requires several element-wise and reduction operations. We denote the input tensor to the softmax module in blue, the output tensor in red, element-wise operations in gray, and reduction operations across the entire vector in orange. The floating-point input tensor is first converted to fixed-point numbers. Then, the tensor is normalized by subtracting the maximum value. The exponentiation is performed using a look-up table (LUT). Finally, the inverse of the sum of the exponentials is multiplied with each element to generate the output tensor.

5.3.1 Hardware Support for Fused-layer Processing

Softmax Module

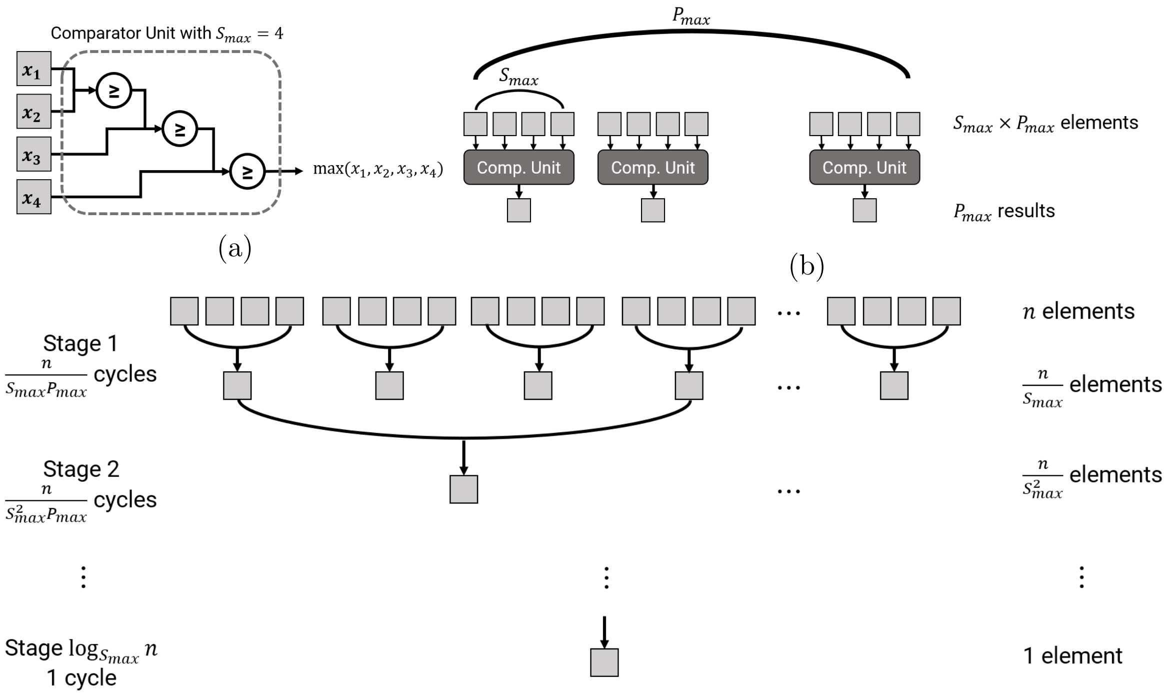
In order to support fused-layer processing targeting a self-attention module, an accelerator has to support on-chip softmax operations. For a one-dimensional vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]$, softmax generates a vector with the same size $\mathbf{y} = [y_1, y_2, \cdots, y_n]$ where

$$y_i = \frac{\exp x_i}{\sum_{j=1}^{n} \exp x_j} \tag{5.2}$$

Softmax involves exponentiation and summation over the entire vector, both of which can be costly in hardware. Thus, hardware support for softmax incurs non-trivial overhead [122]–[129].

We outline the hardware implementation of softmax in Fig. 5.9 when the input vector uses half-precision 16-bit floating point numbers. First, instead of computing the exponential of floating point numbers, we convert the input vector to fixed point numbers ('typecast'). Note that a floating point number has three fields to represent the actual numerical value: sign, exponent, and fraction. The fraction field of a floating point number can be simply shifted by the exponent field to generate a fixed point number, while the sign is preserved. Typecasting thus can be done efficiently with a linear shifter in hardware. Multiple type-casting modules can work in parallel to reduce the latency for this step, and we denote the number of parallelism as P_{type}. Then this typecasting step takes $\lceil \frac{n}{P_{\text{type}}} \rceil$ cycles.

Next, the vector is normalized by subtracting the maximum value from each element in the vector. Identifying the maximum value only requires the comparator. However, sequentially comparing each element will take a total of n cycles. Alternatively, multiple elements ($S_{\max}$) can be compared in a single cycle (serially with a combinatorial logic), although $S_{\max}$ will be limited by the critical path delay (Fig. 5.10a). We can use multiple comparators in parallel as well ($P_{\max}$), where each comparator unit takes $S_{\max}$ elements and produces one



Comparator Unit with $S_{max} = 4$
x_1
x_2
x_3
x_4
≥
$\max(x_1, x_2, x_3, x_4)$
(a)
P_{max}
S_{max}
Comp. Unit
$S_{max} \times P_{max}$ elements
P_{max} results
(b)
n elements
Stage 1
$\frac{n}{S_{max}P_{max}}$ cycles
$\frac{n}{S_{max}}$ elements
Stage 2
$\frac{n}{S_{max}^2 P_{max}}$ cycles
$\frac{n}{S_{max}^2}$ elements
Stage $\log_{S_{max}} n$
1 cycle
1 element
(c)

Figure 5.10: (a) A comparator unit can take multiple elements ($S_{\max} = 4$ in this example), and return the maximum value among those elements in a single cycle. However, the critical path delay increases as more elements have to be compared in a single cycle. (b) Multiple $P_{\max}$ comparator units can operate in parallel. Then, in a single cycle, $P_{\max} \times S_{\max}$ elements can be compared to generate $P_{\max}$ results. (c) The maximum value among n elements can be found across multiple stages.

element as an output (Fig. 5.10b). Since identifying the maximum value requires reduction over the entire vector, comparison operations require multiple stages. In the first stage, the comparator units operate on an input vector across $\lceil \frac{n}{S_{\max} \times P_{\max}} \rceil$ cycles to generate $\lceil \frac{n}{S_{\max}} \rceil$ intermediate elements (Fig. 5.10c). Then, in the second stage, the comparator units operate on these intermediate elements over $\lceil \frac{n}{S_{\max}^2 \times P_{\max}} \rceil$ cycles. The stages are repeated until we have a single maximum value. In this scheme, the maximum element of an input vector can be identified within (ignoring the ceiling for simplicity)

$$\frac{n}{S_{\max} \times P_{\max}} + \frac{n}{S_{\max}^2 \times P_{\max}} + \cdots + 1 \leq \frac{n}{P_{\max}} \frac{1}{S_{\max} - 1} \tag{5.3}$$

cycles.

Subtracting this maximum value from each element in a vector is an element-wise operation, which can be simply performed using a fixed-point subtractor. When P_{sub} subtractors can operate in parallel, this subtraction operation will simply take $\lceil \frac{n}{P_{\text{sub}}} \rceil$ cycles similar to the typecasting step.

Since we subtracted the maximum value in the normalization step, each element is now distributed in a range $(-\infty, 0]$. Now, we want to take an exponential of each element, and normalization helps to limit the output of the exponentiation to $(0, 1]$. A look-up table (LUT) is commonly used to efficiently implement the exponentiation in hardware [127], [129], and we also adopt this approach. A LUT stores a 16-bit floating point value of the exponentiation result for each fixed-point reference point. We limit the range of LUT reference points to $(-8, 0]$, and any input values lesser than -8 will result in 0 output for the exponentiation.

The approximation error depends on how fine-grained the fixed-point reference points are. A simple approach to building a LUT can use a fixed interval to generate the reference points. For example, a fixed interval of $\frac{1}{8}$ for a range of $(-8, 0]$ will result in 64 reference points. Using a smaller interval, such as $\frac{1}{64}$, can be more accurate but has a larger LUT size with 512 reference points. However, note that the error is larger for an input value closer

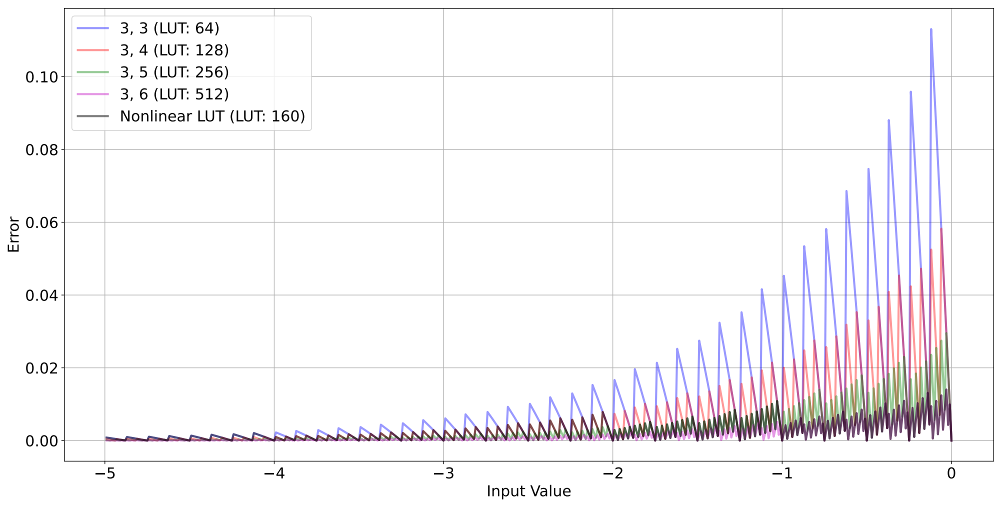


Figure 5.11: The error of approximating the exponential operation with a look-up table using fixed-point references.

to 0 in this fixed interval scheme, whereas smaller input values closer to -8 will only have a small error, due to the characteristic of an exponential function. Thus, instead of using a fixed interval for the LUT reference points, we use different interval values to balance the approximation error and the LUT size [127]. Our scheme uses an interval of $\frac{1}{64}$ for $(-1, 0]$, $\frac{1}{32}$ for $(-2, -1]$, $\frac{1}{16}$ for $(-4, -2]$, and $\frac{1}{8}$ for $(-8, -4]$. The total number of reference points is $64 + 32 + 16 \times 2 + 8 \times 4 = 160$. Fig. 5.11 shows the error of approximating an exponential function using a LUT with fixed-point reference points. Our approach to LUT achieves low approximation error, with a maximum of 0.014 for inputs in the $(-8, 0]$ range, matching that of a much larger LUT using a total 512 reference points with a fixed interval of $\frac{1}{64}$. When P_{lut} input elements can look up this LUT at the same time, then the number of cycles for the exponentiation step will be $\lceil \frac{n}{P_{\text{lut}}} \rceil$.

After we obtain the exponential values, we have to compute the sum across the entire vector. Similar to how we computed the maximum value across the vector, we can use P_{sum} parallel adders, where each adder takes two floating-point numbers. Then, the number of cycles to generate the sum is limited to $\frac{n}{P_{\text{sum}}}$. We then use a single floating-point inverter that computes the inverse of the sum in a single cycle.

Finally, this inverted sum will be multiplied with the exponential values using floating-point multipliers to generate the final output values. This multiplication is an element-wise operation, and the number of cycles for this step will be $\lceil \frac{n}{P_{\text{mult}}} \rceil$ when there are P_{mult} multipliers operating in parallel.

Combining all steps for softmax computation, we can obtain the total number of cycles required for a vector with n elements:

$$\#\text{cycles}_{\text{softmax}} = \lceil \frac{n}{P_{\text{type}}} \rceil + \frac{n}{P_{\text{max}}} \frac{1}{S_{\text{max}} - 1} + \lceil \frac{n}{P_{\text{sub}}} \rceil + \lceil \frac{n}{P_{\text{lut}}} \rceil + \frac{n}{P_{\text{sum}}} + 1 + \lceil \frac{n}{P_{\text{mult}}} \rceil \tag{5.4}$$

For fused-layer processing to be advantageous over writing the intermediate tensor to the off-chip and reading it back, this total number of cycles should be smaller than the latency for writing and reading a vector with n elements to the off-chip. If there are N_w words in a single encryption/decryption block and a cryptographic engine takes k cycles per block, then

$$\#\text{cycles}_{\text{softmax}} < 2 \times \frac{n}{N_w} \times k \tag{5.5}$$

Thus, a designer can choose the parallelization factors for each step in the softmax module to meet this latency constraint, while keeping the area overhead below the budget.

5.4 Results

In this section, we present the implementation result of a secure DNN accelerator equipped with a secure off-chip memory interface. First, we describe the baseline unsecure DNN accelerator architecture in Section 5.4.1 that will be used for evaluation throughout this section. Then, we validate the performance of a secure DNN accelerator estimated by SecureLoop, a design space exploration framework we presented in Chapter 4. Finally, we present a secure DNN accelerator design with a small overhead over the baseline design, demonstrating that off-chip memory security can be practically achieved even for edge accelerators.

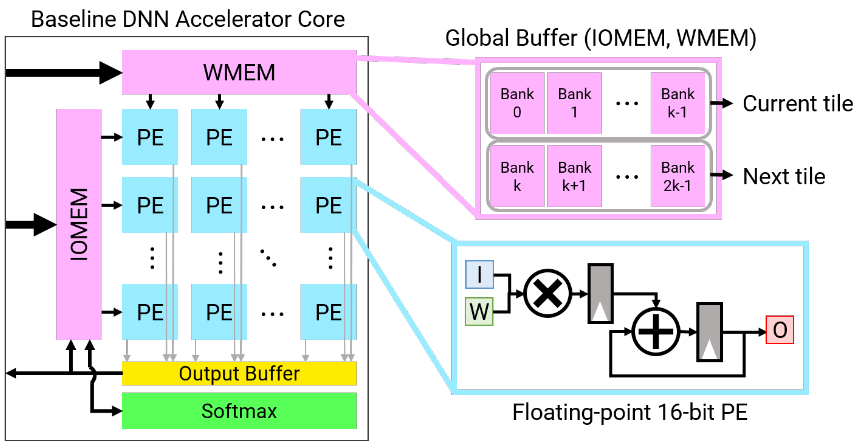


Figure 5.12: The baseline DNN accelerator design used for evaluation.

5.4.1 Baseline DNN Accelerator Architecture

Fig. 5.12 shows the baseline DNN accelerator architecture. It has a 2-dimensional spatial array of processing elements that perform multiply-and-accumulate operations for half-precision floating-point numbers. We adopt output-stationary (OS) dataflow [40], where each processing element (PE) holds an output (or a partial sum) element, and different pairs of an input element and a weight element are supplied to a PE each cycle. When the computation finishes for each output element (i.e., all multiply-and-accumulation operations required to generate an output or a partial sum element are completed), the outputs are written to the output buffer. Typically, the outputs are written back to the off-chip main memory, and the output buffer will request the off-chip writes. Note that the off-chip write can support both the row-major and column-major read-out directions of the PE array.

This baseline design has a simple three-level memory hierarchy of the off-chip main memory, the on-chip global buffers (IOMEM and WMEM), and the register of each PE. The off-chip main memory holds the entire input, weight, and output tensors for a workload. IOMEM and WMEM hold a tile of an input tensor and a weight tensor, respectively. They supply data to the PE array in a multi-cast manner, where one element in an input tile is multi-casted to one row of the PE array, and one element in a weight tile is multi-casted to one column of the PE array. IOMEM and WMEM are implemented as a bank of multiple

```
// DRAM
...
  // IOMEM
  ...
    // WMEM
    ...
    for c in [0, C)
      for r in [0, R)
        for s in [0, S)

          // Parallel
          for q in range [0, Q3)
            for m in range [0, M3)

              // PE
              ...
```

(a) Conv2D

```
// DRAM
...
  // IOMEM
  ...
    // WMEM
    ...
    for c in [0, C)

      // Parallel
      for n in range [0, N3)
        for m in range [0, M3)

          // PE
          ...
```

(b) Batched matrix-vector multiplication

```
// DRAM
for n in [0, N)
  ...
  // IOMEM
  ...
    // WMEM
    ...
    for c in [0, C)

      // Parallel
      for a in range [0, A3)
        for b in range [0, B3)

          // PE
          ...
```

(c) Batched matrix-matrix multiplication

Figure 5.13: Architecture constraints for each workload showing the parallel and temporal factors. These constraints are used to identify the mapping for the baseline architecture. Notations follow Fig. 2.1.

small SRAMs. We support an option to either double-buffer the current and the next tiles by allocating half the bank for each tile or simply single-buffer the current tile by using all the banks.

When fused-layer processing has to be supported, the output buffer writes the first layer outputs to IOMEM instead of the off-chip memory. If softmax operations have to be performed before moving on to the second layer, the softmax module reads the data from IOMEM, computes softmax, and re-writes the data back to IOMEM.

This baseline architecture can support diverse workloads with a high PE array utilization rate. Fig. 5.13 shows the architectural constraints for the loopnest, including the parallelization factors, for each workload type. For a 2-dimensional convolutional layer (Conv2D), the output channel is parallelized across the x-direction such that WMEM can supply weight elements for different output channels in parallel, and the width (column) dimension of the output tensor is parallelized across the y-direction. For a batched matrix-vector multiplication, the output dimension is parallelized across the x-direction similar to a Conv2D, whereas the batch is parallelized across the y-direction. Finally, for a batched matrix-matrix multiplication, the height (row) dimension and the width (column) dimension of the output tensor are parallelized across the y-direction and the x-direction, respectively.

Table 5.1: The specification of a workload used to compare SecureLoop and the RTL implementation

	N	M	C	P/H	Q/W	R	S	Stride	Padding
Size	1	64	64	32/32	32/32	3	3	1	1

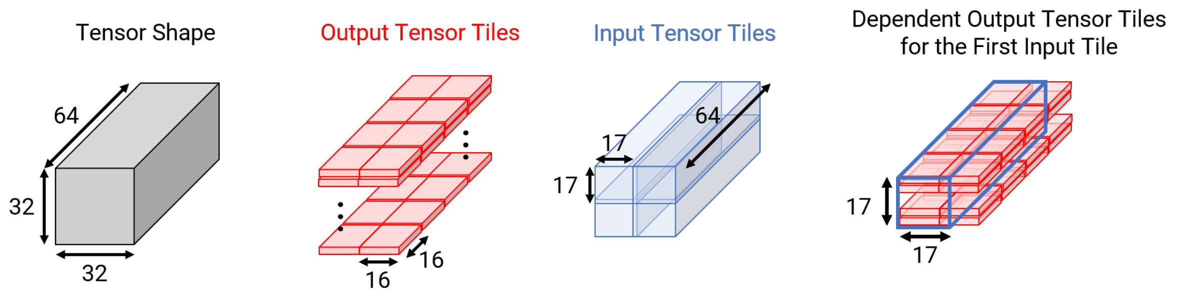


Figure 5.14: A workload used to evaluate the performance of SecureLoop and the cycle-accurate RTL simulation. We consider an input tensor that has dependency on the previous layer's output tensor.

The main controller of the accelerator core tracks the current and next input, weight, and output tile information following the loopnest mapping provided by the configuration. When a new input or weight tile is needed, IOMEM or WMEM will request off-chip data reads. When the computation for one output tile (the output elements computed in parallel in the PE array) is completed, the output buffer will request off-chip data writes as we explained above, unless fused-layer processing is used.

Overall, this baseline architecture is simple yet versatile for various workloads. However, we also note its downsides. In order to fully utilize the $X \times Y$ PE array, WMEM has the capacity to hold $Y \times C \times R \times S$ elements for a Conv2D workload, $Y \times C$ elements for a batched matrix-vector multiplication, and $Y \times C$ elements for a batched matrix-matrix multiplication. When WMEM has insufficient capacity and can only hold $M3 \times C \times R \times S$ elements where $M3 < Y$ for a Conv2D workload, then only $\frac{M3}{Y}$ of the PE array will be utilized. Alternatively, one can tile across C, R, or S dimension at the IOMEM level or the DRAM level and fully utilize the PE array with $M3 = Y$, at the cost of increased off-chip traffic for the partial sums. Thus, the size of WMEM has to be carefully chosen for the typical workloads this accelerator aims to support.

5.4.2 Comparison with SecureLoop

Our secure off-chip memory interface can be augmented on this baseline architecture, where the interface handles off-chip memory reads and writes with support for a TEE. We first compare the performance of a secure DNN accelerator estimated by SecureLoop (Chapter 4) and the actual RTL implementation result. This comparison helps validating the intuitions of SecureLoop that the authentication block assignment has a significant impact on the performance.

We set a baseline DNN accelerator design to use 16×16 PEs, 131kB IOMEM, 131kB WMEM, and DRAM bandwidth of a maximum 16B/cycle for reads and 8B/cycle for writes. For a cryptographic engine, we examine three designs with different throughputs: 'ASCON 1' which processes one permutation round per cycle with a 32B transpose buffer, 'ASCON 2' which processes two permutation rounds per cycle with a 128B transpose buffer, and 'ASCON 4' that processes four permutation rounds per cycle with a 128B transpose buffer. Note that a cryptographic engine design that processes a higher number of rounds per cycle has higher throughput. SecureLoop requires the throughput of a cryptographic engine to be given as an average number of cycles to encrypt/decrypt a 128-bit block and any additional cycles required per authentication block to generate (initialize and finalize) a hash tag. This information can be obtained from the RTL implementation of a cryptographic engine.

We consider a Conv2D workload specified in Table 5.1. We assume that the input tensor of this workload has a dependency on its previous layer's output tensor, where the output tensor tiles have a size of $16 \times 1 \times 16$ and equally divide the tensor (Fig. 5.14). Then, the authentication blocks will be assigned as 'sub-tiles' of the output tensor tiles. The input tensor tiles defined by the loopnest mapping on this workload have a size of $64 \times 17 \times 17$ excluding the padding.

The dependency between the input tensor tiles and the output tensor tiles can incur many redundant reads. For example, assigning the entire output tensor tile as an authentication

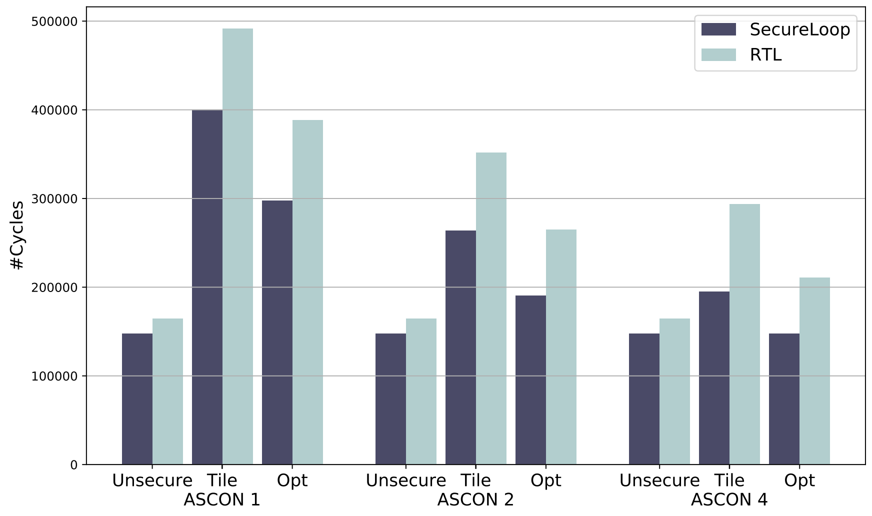


Figure 5.15: Comparing the performance estimated by SecureLoop and measured by the cycle-accurate RTL implementation of the secure accelerator designs.

block (the 'tile-as-an-AuthBlock' strategy in Section 4.2.2) requires fetching $4 \times 17 \times 2$ output tensor tiles in total, resulting in redundant reads of $(4 \times 17 \times 2) \times (16 \times 16) - (64 \times 17 \times 17) = 16320$ elements and $(4 \times 17 \times 2) = 136$ tag reads for every input tensor tile. The optimal assignment strategy identified by SecureLoop uses a smaller authentication block with a size of $16 \times 1 \times 4$. This smaller authentication block reduces the redundant reads to 3264 elements although it increases the tag reads to 340 for every input tensor tile.

Fig. 5.15 shows the performance of the unsecure baseline architecture (normal mode operation without a TEE; 'Unsecure'), the secure mode using the 'tile-as-an-AuthBlock' strategy, ('Tile') and the secure mode using the optimal authentication block assignment ('Opt'). First, SecureLoop closely estimates the performance of the actual RTL implementation, although it generally underestimates the latency. As we discussed in Section 5.1, SecureLoop estimates the latency by taking the maximum of the cycles required for off-chip data transfer, cryptographic operations, and PE computations. However, in the actual implementation, the latency of loading the first input and weight tensor tile cannot be hidden even with the perfect double-buffering. For example, for the 'ASCON 1' configuration, this first tile transfer can take 12% of the latency for the 'Opt' scenario. Subtracting this latency, the

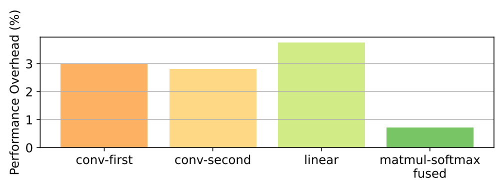
Table 5.2: Design parameters of a secure DNN accelerator implementation

Component	Design Parameters
PE	8×8, Floating-point 16-bit MAC
IOMEM	131kB: 16 512×128 SRAMs
WMEM	131kB: 8 1024×128 SRAMs
Softmax	$P_{\text{type}} = 16$ $S_{\max} = 4, P_{\max} = 4$ $P_{\text{sub}} = 4$ $P_{\text{lut}} = 16$ $P_{\text{sum}} = 16$ $P_{\text{mult}} = 16$
Output Buffer	128B
Input CryptEngine	ASCON-4, 128B TB
Weight CryptEngine	ASCON-4, No TB required
Output CryptEngine	ASCON-4, No TB required

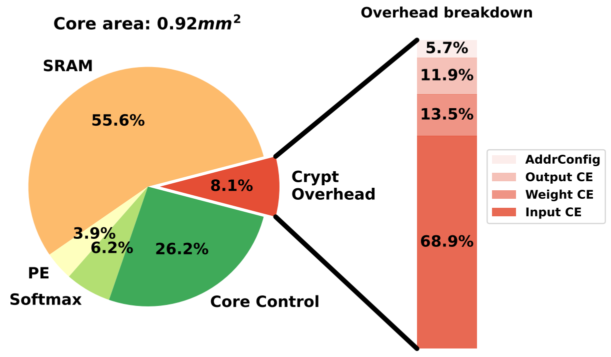
performance estimate of SecureLoop differs by less than 5% from the RTL implementation results.

The remaining errors in the performance estimation of SecureLoop can arise from multiple sources. For a high-throughput cryptographic engine, stalls from the transpose problem cannot be completely removed using a small transpose buffer. Those stall cycles are not modeled in SecureLoop, and they can contribute to this error. Also, we assume that the zero-padding to an input tensor is handled on-chip, requiring a small number of cycles to write zero-padded elements to the SRAM buffer in the actual implementation.

Second, the benefit of using the optimal authentication block assignment is well shown in the actual RTL implementation results. Compared to the 'tile-as-an-AuthBlock' strategy, the optimal authentication block assignment identified by the mapping algorithm of SecureLoop reduces the slowdown by 63%, 53%, and 51% for the 'ASCON 1', 'ASCON 2', and 'ASCON 4' configurations, respectively. Therefore, the authentication block assignment is critical to the performance of secure DNN accelerators, and supporting the off-chip memory interface with the authentication-block-based control is essential.



(a)



(b)

Figure 5.16: (a) The performance overhead of using the secure TEE mode. (b) The area breakdown of a secure DNN accelerator design.

Table 5.3: The power and energy per each MAC operation for the unsecure normal mode and the secure TEE mode.

	Power (mW)	Energy/MAC (pJ)	Overhead
Normal	35.29	2.76	-
TEE	41.13	3.21	16.5%

5.4.3 Implementation Results

We show the hardware implementation result of a secure DNN accelerator design specified in Table 5.2. Our design is synthesized using TSMC 28nm HPC+ CMOS technology and Cadence Genus Synthesis Solution. Fig. 5.16a shows the performance overhead measured as the number of cycles required for processing each workload when using the secure TEE mode normalized by that for when using the unseure normal mode. We measure the performance for four representative workloads: `conv-first` (Conv2D) assumes no cross-layer dependency for an input tensor and uses the 'tile-as-an-AuthBlock' strategy, `conv-second` (Conv2D) is the workload described in Section 5.4.2 and uses the optimal authentication block identified by SecureLoop, `linear` (batched matrix-vector multiplication) with a sufficiently large batch size to fully utilize the PE array, and `matmul-softmax-fused` that has two batched matrix-matrix multiplications with softmax between those two (Fig. 5.8) and uses fused-layer processing. The performance overhead is less than 4% for all these four workloads in the secure TEE mode.

Next, we measure the power and energy consumption when operating at 200MHz frequency and 0.81V voltage (Table 5.3). We measure the power when the accelerator is processing a workload with all PEs active, in order to understand its peak power consumption. For the normal mode, we report the power excluding the power consumed by the cryptographic engines. The secure TEE mode consumes 16.5% more power.

Finally, we report the area breakdown of the secure accelerator design (Fig. 5.16b). The core area of this design is 0.92mm^2. The overhead for supporting a TEE accounts for 8.1% of this core area. The largest source of this overhead (68.9%) is the input cryptographic engine since cross-layer dependency affects input tensors and the authentication-block-based control logic (Section 5.2.2) has to be supported. The weight and output cryptographic engine does not have to support cross-layer dependency (i.e., weight tensors are unique for each layer and authentication blocks can be assigned as 'sub-tiles' of output tensors), and they result in a smaller overhead of 13.5% and 11.9%. Lastly, the logic required for computing the dependent output tiles and authentication blocks takes less than 5.7% of this overhead.

Overall, this implementation results show that a TEE can be supported with a low overhead of $< 4\%$ performance slowdown, 16.5% more energy consumption per each MAC operation, and 8.1% of the core area for a DNN accelerator targeting resource-constrained edge devices.

Post-route Results

We also fabricate a simpler design of a secure DNN accelerator (without support for pipelining between the modules, including double-buffering) using TSMC 28nm HPC+ CMOS technology. Table 5.4 summarizes the post-route specification of the fabricated design using Cadence Innovus Implementation System. Table 5.4 reports power consumption when all PEs are active (full PE utilization), measured at the worst-case corner of 0.81V. The design operates at 167MHz (i.e., 6ns clock period) at this worst-case corner, verified with the SDF-annotated timing checks. As this simpler implementation does not pipeline the off-chip

Table 5.4: Performance and specification of a fabricated design of a secure DNN accelerator

Chip Specification	
Technology	TSMC 28nm HPC+ CMOS
Logic Area	1274k (2-input NAND equivalent)
SRAM	262kB
Power	33.74mW @ 167MHz, 0.81V
PE	64 Half-precision (16-bit) floating-point MAC
Cipher	Ascon-128a
Misc.	No double-buffering / pipeline support

memory access and the on-chip computation, it takes longer cycles to complete processing but has lower power consumption compared to the pipelined design.

5.5 Related Work

There has been a significant research effort in designing a TEE for general-purpose CPUs [95]–[99], [130] and GPUs [131]–[133]. However, extending support for a TEE to DNN accelerators only recently started to be actively investigated [102]–[104], [134]. An important optimization for DNN accelerators is that the timestamp (i.e., also known as a counter or a version number) management can be simplified using the structured and pre-determined data access pattern of DNN accelerators, removing the overhead for managing a tree-structure for the timestamps [102], [104]. Thus, the main source of overhead for supporting a TEE in DNN accelerators boils down to the cryptographic operations on all off-chip data traffic and different off-chip data access patterns arising from the authentication operations (Chapter 4).

Prior work on supporting a TEE in DNN accelerators demonstrated and benchmarked the design using a combination of high-level performance simulators (also Chapter 4), cycle-accurate simulators, and a partial RTL implementation of certain modules [103], [104], [134]. While these simulation and modeling approaches provide valuable insights into the high-level architecture and overhead of supporting a TEE, they lack in detailed understanding of implementation-level challenges (Section 5.1.1). Also, prior work mainly focused on a cloud environment where cryptographic operations required for a secure DNN accelerator can be performed by a onboard CPU [134] or a substantial number of cryptographic engines can be implemented in parallel without a significant overhead to the system [102], [104]. However, these assumptions on the compute capability for cryptographic operations cannot be applied to resource-constrained edge and mobile devices. This chapter provides several techniques and solutions for supporting a TEE in resource-constrained environments and demonstrates the design in silicon.

Recent circuit-level implementations of DNN accelerators support memory encryption although they do not support a full TEE functionality [135]–[139]. For example, [135]–[137] used lightweight cryptography algorithms or standard AES for encrypting weight tensors. Other work leverages a simple XOR operation to encrypt weight tensors [139]. While these implementations illustrate the model parameter encryption is feasible at reasonable hardware overhead for DNN accelerators, they do not extend their support for authenticated encryption on all tensors (including the intermediate tensors that are the core problem due to the cross-layer dependency).

Some implementations [136] also support the authentication of weight tensors to detect unintended bit flips. However, [136] relies on a non-cryptographic authentication algorithm to reduce the complexity of loading the hash tags, thus lacking a strong theoretical security guarantee of cryptographic primitives. Also, such authentication protocols are only supported for weight tensors, leaving out the intermediate tensors required for a full TEE functionality.

5.6 Summary

In this chapter, we present the hardware implementation of a secure DNN accelerator supporting a TEE, specifically targeting resource-constrained edge and mobile devices. We show that support for a TEE can be implemented with a low overhead of $< 4\%$ performance slowdown, 16.5% more energy consumption per multiply-and-accumulate operation, and 8.1% of the accelerator core area. We present both the post-synthesis and post-route simulation results in this chapter.

First, we address the implementation-level challenges of a TEE. Since the granularity of the off-chip data access is an authentication block for a TEE mode operation, the tile information that a DNN accelerator requests for data access should be translated to the authentication block information. The implementation of this translation module only requires basic algebra operations. Furthermore, the received data should be aligned before they can be written to the on-chip SRAM of a DNN accelerator, such that redundant reads can be dropped and the data can be written according to the SRAM address map requirements. We adopt a transpose buffer to resolve the bottleneck in the effective SRAM write bandwidth.

Second, we adopt two techniques for achieving a low overhead for resource-constrained devices. In order to reduce the overhead of a cryptographic engine, we adopt ASCON, a lightweight cryptography standard, to perform authenticated encryption. Moreover, we exploit fused-layer processing to convert memory-bound workloads like a self-attention module in Transformers to compute-bound workloads, such that a low-throughput yet resource-efficient cryptographic engine can be used without incurring the performance overhead. Fused-layer processing for a self-attention module requires on-chip support for softmax operations.

Finally, we validate the SecureLoop's performance estimation using the RTL simulation of this implementation. SecureLoop closely estimates the performance of the actual implementation, where the estimation error is due to the initial tile transfer and the transpose problem. This comparison validates SecureLoop's insight on the impact of authentication block assignment on the performance of a secure DNN accelerator.

Chapter 6

Conclusion

6.1 Summary of Contributions

As DNNs are increasingly adopted for critical applications, security concerns for DNNs are becoming important. This thesis investigates the hardware-level vulnerabilities associated with the off-chip DRAM that acts as a main memory for DNN accelerators. This thesis has the following contributions:

- **Illustrates the vulnerability of sparse DNNs to a small number of bit flips in their weight tensors.** In Chapter 3, we present SparseBFA, an algorithm that identifies the most critical bits in the coordinates of the weight tensors that are stored using sparse matrix formats. SparseBFA can identify less than 0.00005% of total bits in a sparse DNN that result in complete degradation of performance when they are flipped. It exploits the characteristic of a sparse matrix format that a bit flip in the coordinates 'rewires' the connection between neurons (i.e., changes the location of the affected nonzero weight). Using SparseBFA, an adversary can identify the target bits and wage fault injection attacks to the off-chip DRAM that stores those bits. Thus, we show that the algorithmic characteristics can enable exploiting the hardware-level vulnerabilities more easily.

- **Provides a design space exploration framework for secure DNN accelerators supporting a TEE.** As a defense solution against untrusted off-chip memory, a TEE can be adopted. However, a TEE requires authenticated encryption on all off-chip data traffic and changes the off-chip memory access pattern of a DNN accelerator. Furthermore, adding support for cryptographic operations adds hardware costs such as energy and area. In order to systematically analyze the impact of supporting a TEE in the performance, energy, and area of a DNN accelerator, we develop SecureLoop (Chapter 4), a design space exploration framework. The mapping algorithm of SecureLoop identifies the optimal authentication block assignment, such that the additional off-chip traffic due to authentication can be minimized. Also, it enables the joint search of loopnest mappings across multiple layers, such that the impact of the authentication block assignment can be considered from the loopnest mapping stage. This mapping algorithm allows up to 33% faster and 50% better in EDP compared to the baseline mapping algorithm that simply assigns the entire tile as an authentication block. Using SecureLoop, we showcase design space exploration examples for secure DNN accelerators, with sweeps over the PE array size, the on-chip buffer size, and the cryptographic engine configurations.

- **Demonstrates a secure DNN accelerator supporting a TEE can be implemented with a low overhead in silicon.** Our implementation of a secure DNN accelerator in Chapter 5 has $< 4\%$ performance slowdown and 16.5% more energy consumption per multiply-and-accumulate operation compared to the normal unsecure mode operation. Also, the area overhead of security support is 8.1% of the accelerator area. This result illustrates that a TEE that provides both confidentiality and integrity for the off-chip memory can be implemented at a low overhead even for accelerators targeting resource-constrained environments with limited energy and area budget.

- **Presents a comparison of the performance simulation result and the actual**

implementation. As a high-level performance simulator, SecureLoop estimates the performance relying on a few key assumptions. We compare the performance of a secure DNN accelerator estimated by SecureLoop and the actual RTL implementation result. We observe that SecureLoop closely estimates the actual performance when the first tile transfer is accounted for (i.e., the latency for transferring the first tile in each layer cannot be hidden even when using double-buffering), where the key source of difference is the impact of the transpose problem (Section 5.2.3). This comparison also validates the key motivation for SecureLoop that the authentication block assignment has a significant impact on the performance of a secure DNN accelerator. Furthermore, the difference between the simulation and the actual result illustrates how future performance simulators can improve their estimation.

6.2 Future Work

There are many potential future works related to security and machine learning acceleration.

- **Stronger attacks with fewer assumptions.** SparseBFA requires an adversary to have full knowledge about its victim DNN. While this white-box assumption can be applied to many scenarios using open-source DNN models or when an adversary has other capabilities to obtain model parameters [140]–[144], it cannot be applied to other scenarios where a victim uses closed-source proprietary DNN models. A stronger attack assuming black-box or gray-box scenarios can reveal vulnerabilities even for those closed-source models.

- **Extending design space exploration of secure DNN accelerators.** Current SecureLoop assumes that a DNN accelerator supports dense tensor operations and processes each layer individually. However, there is rich work on sparse tensor algebra acceleration [145]–[148]. Sparsity can create a unique challenge for the authentication block assignment since the tiles can have different shapes depending on the sparsity pattern [149]. Furthermore, depending on the tiling strategy employed by a sparse accelerator design [149], the assumption that counters can be computed using the regular and pre-determined access pattern can be challenged. Thus, supporting a TEE for sparse accelerators requires different challenges to be addressed.

Fused-layer processing can benefit secure DNN accelerators since it reduces the amount of off-chip traffic. This characteristic was exploited in Chapter 5 to support a TEE for memory-bound workloads efficiently. More generally, a design space exploration framework supporting fused-layer processing [111] can be used to generate tiling strategies, and SecureLoop can operate on top of those tiling strategies to determine the optimal authentication block assignment.

- **On-line mapping.** When determining the optimal authentication block assignment using SecureLoop, we assume that this mapping process is done off-line for a fixed DNN workload and a secure DNN accelerator architecture. This scenario is adequate for the off-line design space exploration when a designer is evaluating diverse designs or when a workload is fixed during the deployment. However, some deployment scenarios might require an ability to adapt to different resource allocations, such as when multiple tenants are sharing an accelerator [150]–[152]. On-line mapping algorithms for secure DNN accelerators can require faster optimization methods other than an exhaustive search and simulated annealing we used for SecureLoop. Alternatively, if there are a few representative usage patterns (e.g., resource allocation scenarios), one can generate multiple candidate mappings off-line and choose the most adequate one on-line.
- **"Caching" redundant reads.** In this thesis, redundant reads due to the misalignment of tiles and authentication blocks are considered to be dropped after decryption and authentication operations are completed. When the on-chip buffer utilization has to be maximized, this assumption is valid as we do not want to allocate the buffer space

for redundant reads. However, in some cases where the buffer has sufficient empty capacity and redundant reads belong to the tiles that will be used in the near future, it might be more beneficial to keep redundant reads on-chip. If this "caching" is allowed, then redundant reads are no longer a source of overhead and can be simply considered as baseline traffic. Supporting this caching requires modification to SecureLoop such that it can determine if caching is feasible given the loopnest mapping and the authentication block assignment. It also requires modification to the control logic of the secure off-chip memory interface to support caching.

- **Scaling support for a TEE.** DRAM technologies are advancing rapidly to provide high bandwidth for AI applications [153], [154]. Fully utilizing this high bandwidth while supporting a TEE requires cryptographic engines to provide sufficiently high throughput. However, such high-performance cryptographic engines can incur large hardware overhead. Thus, scaling support for a TEE for future DRAM technologies can be challenging yet important. Furthermore, advances in large foundational models [1], [64] show that the memory footprints of such models can exceed the DRAM capacity, requiring the data to be stored at the persistent storage. Thus, extending a TEE to consider the travel between the DRAM and the disk can be essential for those large models.

- **Heterogeneous system.** In this thesis, we considered a standalone DNN accelerator when discussing a TEE. However, in many cases, DNN accelerators will work together with a host CPU, GPU, and other accelerators in a heterogeneous system. Furthermore, some operations in a DNN can be more efficiently done at a CPU (e.g., special functions including sigmoid, exponentiation, etc.). Thus, designing a TEE solution considering that some operations might be performed by a CPU and other components is important.

- **Secret key and attestation.** The security guarantee of a secure DNN accelerator relies on the secrecy of a key used for authenticated encryption. While this thesis did not cover how these private secret keys are set up, there are several techniques to provide these secret keys, including physically unclonable functions [137], [155]–[159] and public-key cryptography for key exchange [160]. These techniques can be integrated with the secure off-chip memory interface proposed in this thesis to complete support for a TEE in the future.

- **Physical-layer security.** A key assumption for a TEE in this thesis is that the on-chip structures are trusted and secure. However, this assumption can be challenged when an adversary has physical access to a DNN accelerator. Then, an adversary can leverage physical side-channel attacks [25]–[27], [143], [161] and fault injection attacks [72], [73], [77] to undermine the on-chip structures. For a scenario where an adversary has physical access, providing defenses against these physical-layer attacks can be important.

Appendix A

Artifact for SecureLoop

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 Fig. 4.11 of SecureLoop (Chapter 4). We use a docker environment to manage all dependencies necessary to run our artifact. Our artifact requires an x86-64 machine and 15GB of disk space for docker support.

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

Description

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.

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.

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 Fig. 4.11 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 Fig. 4.11. The plots can be generated by a shorter Jupyter notebook `workspace/plot_figures.ipynb` once the script is finished.

Evaluation and expected results

For each DNN workload, the notebook generates two plots to compare different scheduling algorithms: 1) performance overhead in the normalized latency (Fig. 4.11(a)), and 2) additional off-chip traffic due to cryptographic operations (Fig. 4.11(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 Fig. 4.11. However, note that the scheduling algorithm involves random processes (e.g., simulated annealing randomly chooses which layer and loopnest mapping to use at each iteration), and the result might not exactly match the numbers in Fig. 4.11. 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 Fig. 4.11 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.

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 (Fig. 4.14).

References

[1] OpenAI, J. Achiam, S. Adler, *et al.*, *Gpt-4 technical report*, 2024. arXiv: 2303.08774 [cs.CL].

[2] *Github copilot.* URL: https://github.com/features/copilot.

[3] S. Ortiz, "The new ai-powered bing is now open to everyone - with some serious upgrades," *ZDNet*, May 2023.

[4] T. Brooks, B. Peebles, C. Holmes, *et al.*, "Video generation models as world simulators," 2024. URL: https://openai.com/research/video-generation-models-as-world-simulators.

[5] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[6] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of field robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[7] J. M. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, pp. 583–589, 2021. URL: https://api.semanticscholar.org/CorpusID:235959867.

[8] J. Cawthra, M. Ekstrom, L. Lusty, J. Sexton, J. Sweetnam, and A. Townsend, *Data integrity: Detecting and responding to ransomware and other destructive events*, 2020. URL: https://www.nccoe.nist.gov/data-integrity-detecting-and-responding-ransomware-and-other-destructive-events.

[9] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen, and K. H. Maier-Hein, "Nnu-net: A self-configuring method for deep learning-based biomedical image segmentation," *Nature methods*, vol. 18, no. 2, pp. 203–211, 2021.

[10] *The Health Insurance Portability and Accountability Act (HIPAA)*. U.S. Dept. of Labor, Employee Benefits Security Administration, 2004.

[11] *General Data Protection Regulation*. URL: https://gdpr-info.eu/.

[12] W. Knight, "Openai's ceo says the age of giant ai models is already over," *Wired*, Apr. 2023. URL: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/.

[13] E. Strickland, "15 graphs that explain the state of ai in 2024: The ai index tracks the generative ai boom, model costs, and responsible ai use," *IEEE Spectrum*, Apr. 2024. URL: https://spectrum.ieee.org/ai-index-2024.

[14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, 2013. arXiv: 1312.6199 `[cs.CV]`.

[15] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[16] K. Lee and A. P. Chandrakasan, "Understanding the energy vs. adversarial robustness trade-off in deep neural networks," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 843–855, 2021. DOI: 10.1109/OJCAS.2021.3116244.

[17] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220. DOI: 10.1109/ICCV.2019.00130.

[18] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "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*, ser. SEC'19, Santa Clara, CA, USA: USENIX Association, 2019, pp. 497–514, ISBN: 9781939133069.

[19] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17, Denver, Colorado: Association for Computing Machinery, 2017, ISBN: 9781450351140. DOI: 10.1145/3126908.3126964. URL: https://doi.org/10.1145/3126908.3126964.

[20] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, *T-bfa: Targeted bit-flip adversarial weight attack*, 2021. arXiv: 2007.12336 [cs.LG].

[21] A. Solender and I. Fried, "Scoop: Congress bans staff use of microsoft's ai copilot," *Axios*, Mar. 2024. URL: https://www.axios.com/2024/03/29/congress-house-strict-ban-microsoft-copilot-staffers.

[22] S. Crawford, S. Kesh, and M. M. Cangueiro, *Ai for security, and security for ai: Two aspects of a pivotal intersection*. URL: https://www.spglobal.com/en/research-insights/featured/special-editorial/ai-for-security-and-security-for-ai-two-aspects-of-a-pivotal-intersection.

[23] M. Lipp, M. Schwarz, D. Gruss, *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990, ISBN: 978-1-939133-04-5. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[24] P. Kocher, J. Horn, A. Fogh, *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002.

[25] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, Springer, 1999, pp. 388–397.

[26] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005, Web Security, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2005.01.010. URL: https://www.sciencedirect.com/science/article/pii/S1389128605000125.

[27] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–45, ISBN: 978-3-540-36400-9.

[28] L. Szekeres, M. Payer, T. Wei, and D. Song, "Sok: Eternal war in memory," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 48–62. DOI: 10.1109/SP.2013.13.

[29] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020, ISSN: 0278-0070. DOI: 10.1109/TCAD.2019.2915318. URL: https://doi.org/10.1109/TCAD.2019.2915318.

[30] A. Hastings, L. Chilton, and S. Sethumadhavan, "How much is performance worth to users?" In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, ser. CF '23, Bologna, Italy: Association for Computing Machinery, 2023, pp. 154–163, ISBN: 9798400701405. DOI: 10.1145/3587135.3592194. URL: https://doi.org/10.1145/3587135.3592194.

[31] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in *2020 57th ACM/IEEE*

Design Automation Conference (DAC), 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218665.

[32] Q. Liu, W. Wen, and Y. Wang, "Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380263. DOI: 10.1145/3400302.3415726. URL: https://doi.org/10.1145/3400302.3415726.

[33] J. Wang, Z. Zhang, M. Wang, H. Qiu, T. Zhang, Q. Li, Z. Li, T. Wei, and C. Zhang, "Aegis: Mitigating targeted bit-flip attacks against deep neural networks," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 2329–2346.

[34] Q. Liu, J. Yin, W. Wen, C. Yang, and S. Sha, "NeuroPots: Realtime proactive defense against Bit-Flip attacks in neural networks," in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 6347–6364, ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/liu-qi.

[35] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210.

[36] K. Lee and A. P. Chandrakasan, "Sparsebfa: Attacking sparse deep neural networks with the worst-case bit flips on coordinates," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 4208–4212. DOI: 10.1109/ICASSP43922.2022.9747337.

[37] "Software enabling for intel® tdx in support of tee-i/o," Intel Corporation, Tech. Rep., 2022.

[38] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," Jan. 2004.

[39] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20, Heraklion, Greece: Association for Computing Machinery, 2020, ISBN: 9781450368827. DOI: 10.1145/3342195.3387532. URL: https://doi.org/10.1145/3342195.3387532.

[40] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379. DOI: 10.1109/ISCA.2016.40.

[41] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "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*, ser. ASPLOS '14, Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, pp. 269–284, ISBN: 9781450323055. DOI: 10.1145/2541940.2541967. URL: https://doi.org/10.1145/2541940.2541967.

[42] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, Seoul, Republic of Korea: IEEE Press, 2016, pp. 243–254, ISBN: 9781467389471. DOI: 10.1109/ISCA.2016.30. URL: https://doi.org/10.1109/ISCA.2016.30.

[43] B. Moons and M. Verhelst, "An energy-efficient precision-scalable convnet processor in 40-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 903–914, 2017. DOI: 10.1109/JSSC.2016.2636225.

[44] N. P. Jouppi, C. Young, N. Patil, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, Toronto, ON, Canada: Association for Computing Machinery, 2017, pp. 1–12, ISBN: 9781450348928. DOI: 10.1145/3079856.3080246. URL: https://doi.org/10.1145/3079856.3080246.

[45] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact rijndael hardware architecture with s-box optimization," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 239–254, ISBN: 978-3-540-45682-7.

[46] P. Hamalainen, T. Alho, M. Hannikainen, and T. Hamalainen, "Design and implementation of low-area and low-power aes encryption hardware core," in *9th EUROMICRO Conference on Digital System Design (DSD'06)*, 2006, pp. 577–583. DOI: 10.1109/DSD.2006.40.

[47] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy, "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*, vol. 46, no. 4, pp. 767–776, 2011. DOI: 10.1109/JSSC.2011.2108131.

[48] Y. Zhang, K. Yang, M. Saligane, D. Blaauw, and D. Sylvester, "A compact 446 gbps/w aes accelerator for mobile soc and iot in 40nm," in *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, 2016, pp. 1–2. DOI: 10.1109/VLSIC.2016.7573553.

[49] U. Banerjee, "Energy-efficient protocols and hardware architectures for transport layer security," M.S. thesis, Massachusetts Institute of Technology, 2017.

[50] U. Banerjee, A. Wright, C. Juvekar, M. Waller, Arvind, and A. P. Chandrakasan, "An energy-efficient reconfigurable dtls cryptographic engine for securing internet-of-

things applications," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 8, pp. 2339–2352, 2019. DOI: 10.1109/JSSC.2019.2915203.

[51] K. Lee, M. Yan, J. Emer, and A. Chandrakasan, "Secureloop: Design space exploration of secure dnn accelerators," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23, Toronto, ON, Canada: Association for Computing Machinery, 2023, pp. 194–208, ISBN: 9798400703294. DOI: 10.1145/3613424.3614273. URL: https://doi.org/10.1145/3613424.3614273.

[52] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315. DOI: 10.1109/ISPASS.2019.00042.

[53] N. Nayak, T. O. Odemuyiwa, S. Ugare, C. Fletcher, M. Pellauer, and J. Emer, "Teaal: A declarative framework for modeling sparse tensor accelerators," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23, Toronto, ON, Canada: Association for Computing Machinery, 2023, pp. 1255–1270, ISBN: 9798400703294. DOI: 10.1145/3613424.3623791. URL: https://doi.org/10.1145/3613424.3623791.

[54] A. Paszke, S. Gross, F. Massa, *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 `[cs.LG]`.

[55] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[58] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[59] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016. DOI: 10.1109/cvpr.2016.90. URL: http://dx.doi.org/10.1109/CVPR.2016.90.

[61] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, 2019. arXiv: 1801.04381 [cs.CV].

[62] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].

[63] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[64] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[65] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, 2020. arXiv: 2005.14165 [cs.CL].

[66] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017. DOI: 10.1109/JPROC.2017.2761740.

[67] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, pp. 65–76, Apr. 2009. DOI: 10.1145/1498765.1498785.

[68] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: Cold boot attacks on encryption keys," in *17th USENIX Security Symposium (USENIX Security 08)*, San Jose, CA: USENIX Association, Jul. 2008. URL: https://www.usenix.org/conference/17th-usenix-security-symposium/lest-we-remember-cold-boot-attacks-encryption-keys.

[69] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, 2015. arXiv: 1506.02626 [cs.NE].

[70] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016. arXiv: 1510.00149 [cs.CV].

[71] M. Seaborn and T. Dullien, *Exploiting the dram rowhammer bug to gain kernel privileges*, Mar. 2015. URL: https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

[72] K. Gomina, J.-B. Rigaud, P. Gendrier, P. Candelier, and A. Tria, "Power supply glitch attacks: Design and evaluation of detection circuits," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 136–141. DOI: 10.1109/HST.2014.6855584.

[73] W. Liu, C.-H. Chang, F. Zhang, and X. Lou, "Imperceptible misclassification attack on deep learning accelerator by glitch injection," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218577.

[74] J.-M. Schmidt and M. Hutter, "Optical and em fault-attacks on crt-based rsa: Concrete results," English, in *Austrochip 2007, 15th Austrian Workhop on Microelectronics, 11 October 2007, Graz, Austria, Proceedings*, Austrochip 2007 ; Conference date: 11-10-2007 Through 11-10-2007, Verlag der Technischen Universität Graz, 2007, pp. 61–67, ISBN: 978-3-902465-87-0.

[75] M. Dumont, M. Lisart, and P. Maurine, "Electromagnetic fault injection : How faults occur," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2019, pp. 9–16. DOI: 10.1109/FDTC.2019.00010.

[76] Q. Liu, L. Guo, and H. Tang, "Fault model analysis of dram under electromagnetic fault injection attack," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137146.

[77] A. Cui and R. Housley, "BADFET: Defeating modern secure boot using Second-Order pulsed electromagnetic fault injection," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC: USENIX Association, Aug. 2017. URL: https://www.usenix.org/conference/woot17/workshop-program/presentation/cui.

[78] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Physical security of deep learning on edge devices: Comprehensive evaluation of fault injection attack vectors," *Microelectronics Reliability*, vol. 120, p. 114 116, 2021, ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2021.114116. URL: https://www.sciencedirect.com/science/article/pii/S0026271421000822.

[79] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational

Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[80] F. Yao, A. S. Rakin, and D. Fan, “Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips,” in *29th USENIX Security Symposium (USENIX Security 20)*, USENIX Association, Aug. 2020, pp. 1463–1480, ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/yao.

[81] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv preprint arXiv:1611.06440*, 2016.

[82] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[83] B. Hassibi, D. Stork, and G. Wolff, “Optimal brain surgeon and general network pruning,” in *IEEE International Conference on Neural Networks*, 1993, 293–299 vol.1. DOI: 10.1109/ICNN.1993.298572.

[84] E. Karnin, “A simple procedure for pruning back-propagation trained neural networks,” *IEEE Transactions on Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990. DOI: 10.1109/72.80236.

[85] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in *International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=rJqFGTslg.

[86] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft filter pruning for accelerating deep convolutional neural networks,” *arXiv preprint arXiv:1808.06866*, 2018.

[87] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, Oct. 2017. DOI: 10.1145/3133901. URL: https://doi.org/10.1145/3133901.

[88] W. Ahrens, T. F. Collin, R. Patel, K. Deeds, C. Hong, and S. Amarasinghe, *Finch: Sparse and structured array programming with control flow*, 2024. arXiv: 2404.16730 [cs.MS].

[89] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, pp. 21–37, 2016, ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[90] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[91] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[92] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[93] S. Zagoruyko and N. Komodakis, *Wide residual networks*, 2016. arXiv: 1605.07146 [cs.CV].

[94] S. Gueron, *A memory encryption engine suitable for general purpose processors*, Cryptology ePrint Archive, Report 2016/204, https://ia.cr/2016/204, 2016.

[95] B. Gassend, G. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003, pp. 295–306. DOI: 10.1109/HPCA.2003.1183547.

[96] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *33rd International Symposium on Computer Architecture (ISCA'06)*, 2006, pp. 179–190. DOI: 10.1109/ISCA.2006.22.

[97] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "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)*, 2007, pp. 183–196. DOI: 10.1109/MICRO.2007.16.

[98] M. Taassori, A. Shafiee, and R. Balasubramonian, "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*, ser. ASPLOS '18, Williamsburg, VA, USA: Association for Computing Machinery, 2018, pp. 665–678, ISBN: 9781450349116. DOI: 10.1145/3173162.3177155. URL: https://doi.org/10.1145/3173162.3177155.

[99] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 416–427. DOI: 10.1109/MICRO.2018.00041.

[100] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[101] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "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*, ser. ASPLOS '19, Providence, RI, USA: Association for Computing Machinery, 2019, pp. 137–151, ISBN: 9781450362405. DOI: 10.1145/3297858.3304025. URL: https://doi.org/10.1145/3297858.3304025.

[102] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Mgx: Near-zero overhead memory protection for data-intensive accelerators," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22, New York, New York: Association for Computing Machinery, 2022, pp. 726–741, ISBN: 9781450386104. DOI: 10.1145/3470496.3527418. URL: https://doi.org/10.1145/3470496.3527418.

[103] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Guardnn: Secure accelerator architecture for privacy-preserving deep learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, San Francisco, California: Association for Computing Machinery, 2022, pp. 349–354, ISBN: 9781450391429. DOI: 10.1145/3489517.3530439. URL: https://doi.org/10.1145/3489517.3530439.

[104] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "Tnpu: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 229–243. DOI: 10.1109/HPCA53966.2022.00025.

[105] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2021, pp. 554–566. DOI: 10.1109/ISCA52012.2021.00050. URL: https://doi.ieeecomputersociety.org/10.1109/ISCA52012.2021.00050.

[106] P. Chatarasi, H. Kwon, A. Parashar, M. Pellauer, T. Krishna, and V. Sarkar, "Marvel: A data-centric approach for mapping deep learning operators on spatial accelerators," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 1, Dec. 2021, ISSN: 1544-3566. DOI: 10.1145/3485137. URL: https://doi.org/10.1145/3485137.

[107] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Trans. Embed. Comput. Syst.*,

vol. 18, no. 5s, Oct. 2019, ISSN: 1539-9087. DOI: 10.1145/3358198. URL: https://doi.org/10.1145/3358198.

[108] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: Enabling efficient algorithm-accelerator mapping space search," ser. ASPLOS '21, Virtual, USA: Association for Computing Machinery, 2021, pp. 943–958, ISBN: 9781450383172. DOI: 10.1145/3445814.3446762. URL: https://doi.org/10.1145/3445814.3446762.

[109] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450380263. DOI: 10.1145/3400302.3415639. URL: https://doi.org/10.1145/3400302.3415639.

[110] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *J. Cryptol.*, vol. 34, no. 3, p. 33, 2021. DOI: 10.1007/S00145-021-09398-9. URL: https://doi.org/10.1007/s00145-021-09398-9.

[111] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, *Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks*, 2022. arXiv: 2212.10612 [cs.AR].

[112] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG].

[113] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8. DOI: 10.1109/ICCAD45719.2019.8942149.

[114] S. Mehta and M. Rastegari, "Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer," in *International Conference on Learning Representations*, 2022. URL: https://openreview.net/forum?id=vh-0sUt8HlG.

[115] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, *Flashattention: Fast and memory-efficient exact attention with io-awareness*, 2022. arXiv: 2205.14135 [cs.LG].

[116] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "Flat: An optimized dataflow for mitigating attention bottlenecks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023, Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 295–310, ISBN: 9781450399166. DOI: 10.1145/3575693.3575747. URL: https://doi.org/10.1145/3575693.3575747.

[117] J.-W. Su, X. Si, Y.-C. Chou, *et al.*, "15.2 a 28nm 64kb inference-training two-way transpose multibit 6t sram compute-in-memory macro for ai edge chips," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 240–242. DOI: 10.1109/ISSCC19947.2020.9062949.

[118] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, *Advanced encryption standard (aes)*, en, 2001-11-26 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197.

[119] S. Gueron, May 2010. URL: https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf.

[120] C. Boutin, *Nist selects 'lightweight cryptography' algorithms to protect small devices*, Feb. 2023. URL: https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices.

[121] K. Ueyoshi, I. A. Papistas, P. Houshmand, *et al.*, "Diana: An end-to-end energy-efficient digital and analog hybrid neural network soc," in *2022 IEEE International*

Solid-State Circuits Conference (ISSCC), vol. 65, 2022, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731716.

[122] I. Kouretas and V. Paliouras, “Simplified hardware implementation of the softmax activation function,” in *2019 8th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, 2019, pp. 1–4. DOI: 10.1109/MOCAST.2019.8741677.

[123] M. Wang, S. Lu, D. Zhu, J. Lin, and Z. Wang, “A high-speed and low-complexity architecture for softmax function in deep learning,” in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 223–226. DOI: 10.1109/APCCAS.2018.8605654.

[124] Y. Gao, W. Liu, and F. Lombardi, “Design and implementation of an approximate softmax layer for deep neural networks,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180870.

[125] H. Chen, Z. Yu, J. Xu, L. Jiang, Z. Lu, Y. Fu, and L. Li, “Huicore: A generalized hardware accelerator for complicated functions,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2463–2476, 2022. DOI: 10.1109/TCSI.2022.3152799.

[126] K. Chen, Y. Gao, H. Waris, W. Liu, and F. Lombardi, “Approximate softmax functions for energy-efficient deep neural networks,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 1, pp. 4–16, 2023. DOI: 10.1109/TVLSI.2022.3224011.

[127] X. Dong, X. Zhu, and D. Ma, “Hardware implementation of softmax function based on piecewise lut,” in *2019 IEEE International Workshop on Future Computing (IWOFC*, 2019, pp. 1–3. DOI: 10.1109/IWOFC48002.2019.9078446.

[128] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 469–474. DOI: 10.1109/DAC18074.2021.9586134.

[129] Z. Wei, A. Arora, P. Patel, and L. John, "Design space exploration for softmax implementations," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2020, pp. 45–52. DOI: 10.1109/ASAP49362.2020.00017.

[130] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 454–465. DOI: 10.1109/HPCA.2018.00046.

[131] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, Oct. 2018, pp. 681–696, ISBN: 978-1-939133-08-3. URL: https://www.usenix.org/conference/osdi18/presentation/volos.

[132] R. Abdullah, H. Zhou, and A. Awad, "Plutus: Bandwidth-efficient memory security for gpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 543–555. DOI: 10.1109/HPCA56546.2023.10071100.

[133] S. Na, J. Kim, S. Lee, and J. Huh, "Supporting secure multi-gpu computing with dynamic and batched metadata management," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2024, pp. 204–217. DOI: 10.1109/HPCA57654.2024.00025. URL: https://doi.ieeecomputersociety.org/10.1109/HPCA57654.2024.00025.

[134] A. Dhar, S. Sridhara, S. Shinde, S. Capkun, and R. Andri, *Empowering data centers for next generation trusted computing*, 2022. arXiv: 2211.00306 [cs.CR].

[135] S. Maji, U. Banerjee, S. H. Fuller, and A. P. Chandrakasan, "A threshold implementation-based neural network accelerator with power and electromagnetic side-channel countermeasures," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 141–154, 2023. DOI: 10.1109/JSSC.2022.3215670.

[136] S. Maji, K. Lee, C. Gongye, Y. Fei, and A. P. Chandrakasan, "An energy-efficient neural network accelerator with improved resilience against fault attacks," *IEEE Journal of Solid-State Circuits*, pp. 1–11, 2024. DOI: 10.1109/JSSC.2024.3374638.

[137] M. Ashok, S. Maji, X. Zhang, J. Cohn, and A. P. Chandrakasan, "A secure digital in-memory compute (imc) macro with protections for side-channel and bus probing attacks," in *2024 IEEE Custom Integrated Circuits Conference*, 2024.

[138] Q. Fang, L. Lin, H. Zhang, T. Wang, and M. Alioto, "Voltage scaling-agnostic counteraction of side-channel neural net reverse engineering via machine learning compensation and multi-level shuffling," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1–2. DOI: 10.23919/VLSITechnologyandCir57934.2023.10185228.

[139] Y.-C. Chiu, C.-S. Yang, S.-H. Teng, *et al.*, "A 22nm 4mb stt-mram data-encrypted near-memory computation macro with a 192gb/s read-and-decryption bandwidth and 25.1-55.1tops/w 8b mac for ai operations," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 178–180. DOI: 10.1109/ISSCC42614.2022.9731621.

[140] Y.-S. Won, S. Chatterjee, D. Jap, A. Basu, and S. Bhasin, "Deepfreeze: Cold boot attacks and high fidelity model recovery on commercial edgeml device," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9. DOI: 10.1109/ICCAD51958.2021.9643512.

[141] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532, ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

[142] C. Gongye, Y. Fei, and T. Wahl, "Reverse-engineering deep neural networks using floating-point timing side-channels," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218707.

[143] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—attacks and defenses," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12 079–12 092, 2021. DOI: 10.1109/JIOT.2021.3061314.

[144] X. Zhang, A. A. Ding, and Y. Fei, "Deep-learning model extraction through software-based power side-channel," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9. DOI: 10.1109/ICCAD57390.2023.10323806.

[145] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "Extensor: An accelerator for sparse tensor algebra," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 319–333, ISBN: 9781450369381. DOI: 10.1145/3352460.3358275. URL: https://doi.org/10.1145/3352460.3358275.

[146] G. Zhang, N. Attaluri, J. S. Emer, and D. Sanchez, "Gamma: Leveraging gustavson's algorithm to accelerate sparse matrix multiplication," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21, Virtual, USA: Association for Computing Ma-

chinery, 2021, pp. 687–701, ISBN: 9781450383172. DOI: 10.1145/3445814.3446702. URL: https://doi.org/10.1145/3445814.3446702.

[147] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 724–736. DOI: 10.1109/HPCA.2018.00067.

[148] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1377–1395, 2022. URL: https://api.semanticscholar.org/CorpusID:248721917.

[149] T. O. Odemuyiwa, H. Asghari-Moghaddam, M. Pellauer, *et al.*, "Accelerating sparse data orchestration via dynamic reflexive tiling," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023, Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 18–32, ISBN: 9781450399180. DOI: 10.1145/3582016.3582064. URL: https://doi.org/10.1145/3582016.3582064.

[150] S. Banerjee, S. Wei, P. Ramrakhyani, and M. Tiwari, "Triton: Software-defined threat model for secure multi-tenant ml inference accelerators," in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '23, Toronto, Canada: Association for Computing Machinery, 2023, pp. 19–28, ISBN: 9798400716232. DOI: 10.1145/3623652.3623672. URL: https://doi.org/10.1145/3623652.3623672.

[151] S. Kim, J. Zhao, K. Asanovic, B. Nikolic, and Y. S. Shao, "Aurora: Virtualized accelerator orchestration for multi-tenant workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23,

Toronto, ON, Canada: Association for Computing Machinery, 2023, pp. 62–76, ISBN: 9798400703294. DOI: 10.1145/3613424.3614280. URL: https://doi.org/10.1145/3613424.3614280.

[152] S. Ghodrati, B. H. Ahn, J. Kyung Kim, *et al.*, “Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 681–697. DOI: 10.1109/MICRO50266.2020.00062.

[153] SK, *Hbm technology: The silent catalyst of the ai revolution*, Jan. 2024. URL: https://eng.sk.com/perspectives/hbm-technology-the-silent-catalyst-of-the-ai-revolution.

[154] Micron, *Micron commences volume production of industry-leading hbm3e solution to accelerate the growth of ai*, Feb. 2024. URL: https://investors.micron.com/news-releases/news-release-details/micron-commences-volume-production-industry-leading-hbm3e.

[155] G. E. Suh and S. Devadas, “Physical unclonable functions for device authentication and secret key generation,” in *2007 44th ACM/IEEE Design Automation Conference*, 2007, pp. 9–14.

[156] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, “Silicon physical random functions,” in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS ’02, Washington, DC, USA: Association for Computing Machinery, 2002, pp. 148–160, ISBN: 1581136129. DOI: 10.1145/586110.586132. URL: https://doi.org/10.1145/586110.586132.

[157] R. Maes and I. Verbauwhede, “Physically unclonable functions: A study on the state of the art and future research directions,” in *Towards Hardware-Intrinsic Security: Foundations and Practice*, A.-R. Sadeghi and D. Naccache, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–37, ISBN: 978-3-642-14452-3. DOI: 10.1007/978-3-642-14452-3_1. URL: https://doi.org/10.1007/978-3-642-14452-3_1.

[158] Y. Gao, S. F. Al-Sarawi, and D. Abbott, “Physical unclonable functions,” *Nature Electronics*, vol. 3, no. 2, pp. 81–91, 2020.

[159] H. Sun, S. Maji, A. P. Chandrakasan, and B. Marelli, “Integrating biopolymer design with physical unclonable functions for anticounterfeiting and product traceability in agriculture,” *Science Advances*, vol. 9, no. 12, eadf1978, 2023. DOI: 10.1126/sciadv.adf1978. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.adf1978. URL: https://www.science.org/doi/abs/10.1126/sciadv.adf1978.

[160] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976. DOI: 10.1109/TIT.1976.1055638.

[161] S. Maji, K. Lee, and A. P. Chandrakasan, “Sparseleakynets: Classification prediction attack over sparsity-aware embedded neural networks using timing side-channel information,” *IEEE Computer Architecture Letters*, pp. 1–4, 2024. DOI: 10.1109/LCA.2024.3397730.