S^2Loop: Explore Optimal Authentication Block Strategy for ML

Jan Strzeszynski*
Massachusetts Institute of Technology
jstrz@mit.edu

Jianming Tong*
Georgia Institute of Technology/MIT
jianming.tong@gatech.edu

Kyungmi Lee*
Massachusetts Institute of Technology
kyungmi@mit.edu

Nathan Xiong
Massachusetts Institute of Technology
nxiong@mit.edu

Angshuman Parashar
NVIDIA
aparashar@nvidia.com

Joel Emer
MIT/NVIDIA
jsemer@mit.edu

Tushar Krishna
Georgia Institute of Technology
tushar@ece.gatech.edu

Mengjia Yan
Massachusetts Institute of Technology
mengjiay@mit.edu

Abstract

Off-chip memory in ML accelerators is vulnerable to both hardware and software attack, which needs encryption and authentication. Precise performance modeling of it requires (1) representation of authentication blocks (AuthBlock) to cover full design space of multi-dimensional tiles based authentication,and (2) precise memory behavior modeling as encryption and authentication mainly add up memory traffic. This paper introduces S^2Loop, a framework that resolves these challenges by introducing (1) flexible, all-level partitioning based AuthBlocks for ensuring full coverage of the entire design space, (2) a realistic layout-based memory model, and (3) an Authentication-Layout-Mapping co-search algorithm to explore the drastic combinatorial design space to figure out optimal mapping, layout, and authentication choice for multi-layer workloads. SquareLoop's detailed memory model helps find better mapping to achieve 1.32× speedup on ResNet18 compared to the SotA SecureLoop, and our latency predictions are validated to within 7.3% of an RTL implementation. *S^2Loop* also achieve up-to 1.08×/1.82× overall speedup for authenticated ResNet18/MobileNet-V3 on various accelerators with AuthBlock and Mapping co-searching. We open-source S^2Loop to provide a powerful and validated tool for designing efficient, secure accelerators at https://github.com/maeri-project/squareloop.

CCS Concepts

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

Keywords

Neural networks, accelerator scheduling

1 Introduction

The slowdown of Moore's Law has pushed modern Systems-on-Chip (SoCs) toward heterogeneous designs in which general-purpose CPUs offload compute-intensive tasks (e.g., AI inference) to domain-specific accelerators. These engines share *untrusted* off-chip memory, exposing confidentiality and integrity risks such as cold-boot [8] and Rowhammer [25] attacks. Consequently, every datum crossing the memory boundary must be protected using authenticated encryption (AE)—encryption for confidentiality and a Message Authentication Code (MAC) for integrity. The cost is non-trivial: AE injects per-access latency and bandwidth overhead.

A central lever in overhead is the *authentication block* (*AuthBlock*, Fig. 1): the granularity at which ciphertext is tagged and verified. Before an off-chip write, output data generated on-chip is partitioned into AuthBlocks, encrypted, and hashed. The follow-on off-chip read must fetch and verify the entire AuthBlock even if only a subset is needed. Thus, any mismatch between the compute tile shape and the AuthBlock shape forces *redundant* traffic purely for integrity, inflating off-chip memory latency and bandwidth usage.

AuthBlock presents a large design space (e.g. 10^7 for the first layer of ResNet18). The state-of-the-art AuthBlock search tool, SecureLoop [17], effectively explores a *subset of the entire design space*. SecureLoop restricts the choice of AuthBlock: flatten multi-dimensional tensor and partitions flattened 1-D tensor into equally-sized AuthBlocks, termed as ***"single level partitioning"***. By carefully selecting the flattening orientation (row-wise or column-wise) and the size of AuthBlock, SecureLoop aims to find a strategy that minimizes the cryptographic overhead. However, such restricted design choice might not cover the optimal AuthBlock to resolve the conflicting usage of the same data in different layers, leading to redundant off-chip memory traffic, as detailed in §3.1.

Further, SecureLoop adopts an ideal bandwidth-based memory model, which assumes an ideal on-chip memory system where data is always available at maximum bandwidth. It ignores the fact that data residing in different lines of memory cannot be accessed together in the same read, even if there is available bandwidth. This oversimplification leads to an inaccurate underestimation of memory latency and, therefore, a flawed assessment of authentication costs. This might lead to up to 128× sim-to-real gap [24].

This paper introduces **SquareLoop**, a novel framework that overcomes the two fundamental limitations. Our core insight is that authentication blocks should not be constrained to a single-level chunking of compute tiles. Instead, they should be flexible, all-dimensional partitioning of each compute tile for covering the optimal design choice of AuthBlock to minimize authentication overhead. While prior work like LayoutLoop [24] introduced a formalism for multi-dimensional data organization in on-chip buffers, it is unsuitable for this task. LayoutLoop imposes a unified layout on

*These authors contributed equally to this research.

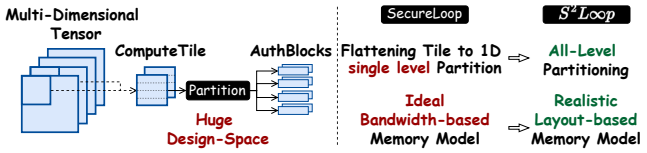


Figure 1: AuthBlock and S^2Loop contributions. A tensor is decomposed into compute tiles for accelerator execution, which is further partitioned into uniformly shaped authentication blocks (AuthBlocks) for encryption and integrity. S^2Loop extends SecureLoop by (1) enabling all-level partitioning based AuthBlock representation to cover the full design space, and (2) integrating a realistic, layout-aware memory model for accurate security overhead estimation.

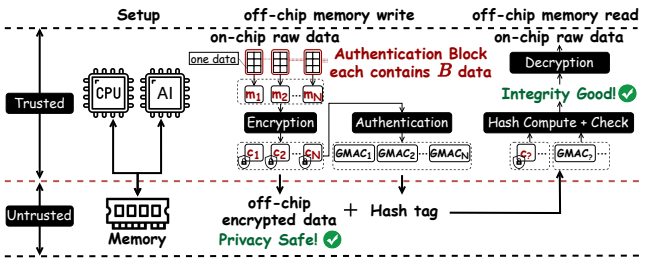


Figure 2: Overview of the system architecture for secure data processing, where compute units like the CPU and AI accelerator are in a trusted environment, but the off-chip memory is considered untrusted [12]. Each off-chip write is protected with authenticated encryption (e.g., AES-GCM [4]): data are grouped into *authentication blocks* (AuthBlocks), encrypted, and tagged with a GMAC hash. A off-chip read fetches the whole AuthBlock and its tag, re-computes the hash, and decrypts only if the tag matches.

all data and, more importantly, lacks a cost model for the encryption and hashing overheads central to secure memory systems.

SquareLoop synthesizes the security-aware cost modeling of SecureLoop with a powerful multi-dimensional tiling-based layout formalism inspired by LayoutLoop, creating a unified and validated simulation tool. Our contributions are:

- **A formalism for multi-dimensional authentication blocks.** We replace the restrictive single-level tiling-based AuthBlock of SecureLoop with a flexible, multi-dimensional layout-based AuthBlock representation, enhancing the granularity of authenticating tensor data that significantly reduces security overheads.
- **A realistic, crypto-aware memory cost model.** We develop an analytical model that accounts for layout, line granularity, bank conflicts, and authenticated encryption pipelines (encryption and GMAC), yielding accurate latency estimates for reads/writes under AuthBlocks.
- **A fast mapping-layout-authentication co-search algorithm.**
- **The release of SquareLoop as an open-source tool.** We provide our validated and extensible simulator to the community to facilitate further research in secure accelerator design and evaluation. We improved SecureLoop significantly by validating this new tool against a realistic silicon implementation with 7% error, caused by control costs that are infeasible to be modeled.

Table 1: Terminology

Symbol	Definition
D	Total number of data without authenticated encryption.
N	Number of Authentication Blocks (AuthBlock).
B	Number of data in each AuthBlock.
M_{cipher}	hash bit length of a cipher, typically 128 bits (an attacker has $\frac{1}{2^{128}}$ chance to create a valid forgery).
N_{AES}, T_{AES}	Number of hardware AES engines, and its latency
N_{GF}, T_{GF}	Number of hardware GF engines, and its latency

2 Background

2.1 Memory Encryption and Integrity Check

2.1.1 System Setup. We consider a common system where AI accelerator and host CPU share the same off-chip memory [14, 23].

2.1.2 Threat Model. We assume no physical side-channels (power/EM side-channels [**?**]) for on-chip structures. From the software side channels, the host CPU is equipped with memory security solutions (e.g., memory encryption engine in Intel SGX [21]) so that off-chip access from CPU-side is secured. The accelerator implements static in-order control without speculation and hence being robust against Meltdown [19] and Spectre [16]. However, the off-chip data originated from an accelerator is vulnerable to cold-boot [8], Rowhammer [15] and replay-attack [6], unless we protect it with cryptographic encryption and authentication.

2.1.3 Adversary. We assume an attacker who has access to the physical off-chip memory for mounting a hardware attack [8] and controls software to probe or tamper data in off-chip memory [2, 15]. The attacker's goals are to (i) learn confidential secrets and (ii) modify malicious values that corrupt model execution.

[Jianming: Corrected the threat model.][Kyungmi: Thanks! Can we also add that we consider on-chip structures of accelerators to be 'secure', in that they are not vulnerable to side-channel attacks?] [Jianming: Fixed]

2.1.4 Secure Mechanisms. For *confidentiality* and *integrity*, data needs to go through encryption and hash calculation before it gets sent back to off-chip memory. Extra protections are introduced as detailed in Fig. 2.

Off-chip memory write: A write has following steps.

- *Partition:* raw data is divided into *authentication blocks* (AuthBlock). Extra zeros are padded if the shape of AuthBlock does not perfectly divide the shape of the data.
- *Encrypt:* Each AuthBlock m_k is XOR-masked with a keystream generated by AES in CTR mode to generate encrypted ciphertext c_k. Encryption of each different m_k has no dependency, and could be parallel accelerated.
- *Authenticate:* Each c_k is multiplied by a shared secret subkey H in $GF(2^{128})$ and XOR-reduced (GMAC) with a pseudo-random number to produce a 128-bit tag.
- *Store:* Ciphertext c_k and tag $GMAC_k$ are stored to DRAM.

Off-chip memory read: A read performs the inverse: fetch $\langle c_k, GMAC_k \rangle$, recompute the tag, and then decrypt c_k only if computed tag matches fetched $GMAC_k$ (Fig. 2).

2.2 Data Orchestration in DNN Accelerators

Data orchestration within DNN accelerators is the process of managing data movement to leverage reuse opportunities across a memory

hierarchy, which typically includes off-chip main memory (DRAM), several levels of on-chip buffers, and an array of processing elements. The execution of a workload on the accelerator is dictated by its *mapping* [1], which defines both the temporal and spatial aspects of memory access and computation. This involves dividing the overall data into smaller blocks, or "tiles," to be computed and moved throughout the memory hierarchy. The primary goals of a chosen mapping are to minimize computation latency and maximize data reuse, thereby enhancing energy efficiency.

Complementing the mapping is the data *layout*, which specifies how data is organized within each level of the multi-bank on-chip buffers. Specifically, the layout determines which data elements are grouped together into a single buffer line for parallel access. An optimal layout ensures two key performance optimizations. (1) Avoid Bank Conflict: Data requested concurrently by the processing elements resides in the same buffer line, allowing it to be accessed in a single cycle. If concurrently required data is spread across different lines within the same memory bank, it creates a "bank conflict," which stalls computation and degrades performance. (2) Avoid Excessive data access: Data not being used together to be separated into different lines such that we don't access data that we don't need. Therefore, mapping and layout must be explored in tandem to achieve performance targets like minimal latency and energy consumption for a given workload and architecture.

2.3 Overhead Breakdown

Authenticated encryption has following overheads (Tab. 2)

Table 2: Encryption and Integrity Verification Overhead

Overhead	Encryption/Decryption	Authentication
Ideal Latency	$N \cdot \frac{B}{N_{AES}} \cdot T_{AES}$	$N \cdot \frac{B}{N_{GF}} \cdot T_{GF}$
Storage/Memory	0	$N \cdot M_{cipher}$ bit

2.3.1 Area Overhead (Chip Fabrication). AES-GCM encryption uses a vectorized XOR operation, while authentication relies on Galois Field (GF) multiplication. Since both are not natively supported by AI accelerators, dedicated *AES engines* and *GF engines* are required, introducing area overhead.

2.3.2 Memory Overhead. Hash tags are the only source of extra memory overhead, with M_{cipher}-bit overhead per authentication block, the overall extra memory overhead is

$$N \cdot M_{cipher} + datawidth \cdot \#redundant\_data \tag{1}$$

Redundant Data Reads. Compute operates on *tiles*. When the tile shape mismatch the shape of AuthBlock, integrity verification forces the memory system to fetch the entire AuthBlock even though only a subset of data is needed, leading to redundant data shown in red hash area in Fig. 3. Avoiding such overhead requires carefully picking AuthBlock to minimize redundant data reads.

Insight: For a specific cipher, *fewer large authentication blocks with less redundant reads are favored to have less extra storage overhead. This requires a careful selection of the AuthBlock shape.* Note that nonces can embed the address and version numbers of authentication blocks to prevent splicing and replay attacks. For AI accelerators, both addresses and version numbers can be easily calculated on-chip [12, 18], and thus nonces do not add any storage overhead.

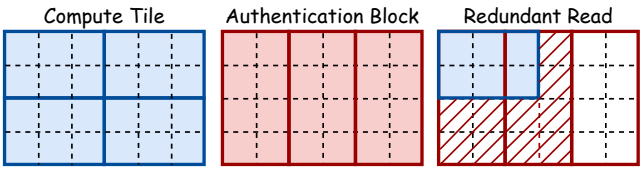


Figure 3: Illustration of *Redundant Read* from the mismatch between compute tiles (blue) and authentication blocks (red). Fetching one compute tile requires loading all overlapping AuthBlocks, causing unused data (red-shaded) to be read, adding latency and memory bandwidth overhead.

2.3.3 Latency Overhead. With N_{AES} AES dedicated engines and N_{GF} dedicated engines for GF multiplication, ideal encryption/decryption latency, assuming pipelined encryption and integrity is

$$T_{\text{ideal}} = N \cdot \max\left(\frac{B}{N_{\text{AES}}} \cdot T_{AES}, \frac{B}{N_{\text{GF}}} \cdot T_{GF}\right) \tag{2}$$

We note the latency T_{ideal} as *crypto latency*. However, data might be fetched from off-chip memory many times because of **repeated data access needed by mapping.** A mapping might need the same data at a different time, resulting in extra latency overhead. Reducing such overhead requires carefully selecting the mapping.

In summary, to minimize latency and memory overhead, the key research question is to *find an optimal mapping and its authentication block shape to minimize (1) the number of AuthBlock N, (2) redundant reads, (3) repeated data access, and (4) compute latency.*

3 Motivation and Contributions

While SecureLoop represents a step towards securing DNN accelerators, its methodology has two critical limitations (C1 and C2) that hinder its effectiveness in fully resolving security redundancy. These challenges motivate our new framework, SquareLoop.

3.1 C1: SubOptimal 1-level Tiled AuthBlock

Limitation. The core limitation of SecureLoop is its rigid, single-level partitioning-based AuthBlocks, which only cover a subset of the entire design space. Specifically, SecureLoop first flattens a multi-dimensional tile of data into a 1D vector and then divides that vector into fixed-size authentication blocks (AuthBlocks), i.e. single-level partitioning. Albeit such a strategy is effective for row-stationary dataflow accelerators like Eyeriss [1], it fundamentally mismatches with the multi-dimensional nature of tensor computations. For example, in Fig. 4, workloads often involve consecutive layers that produce and consume data using differently shaped multi-dimensional tiles. For instance, a producer layer outputs 4×4 tiles, while the consumer layer ingests 6×6 tiles. SecureLoop's 1D partitioning forces the creation of AuthBlocks (e.g., 2×4) that do not align with the consumer's tile shape. Consequently, when the consumer layer fetches a single 6×6 tile, it must read six separate AuthBlocks, leading to significant redundant data traffic (48 redundant reads) and hash computations (24 hash reads).

Solution. In contrast, SquareLoop supports single-level tiling AuthBlock partitioning. By partitioning data across all dimensions, it can define an AuthBlock shape (e.g., 2×2) that is a common factor of both the producer and consumer tiles. This alignment completely

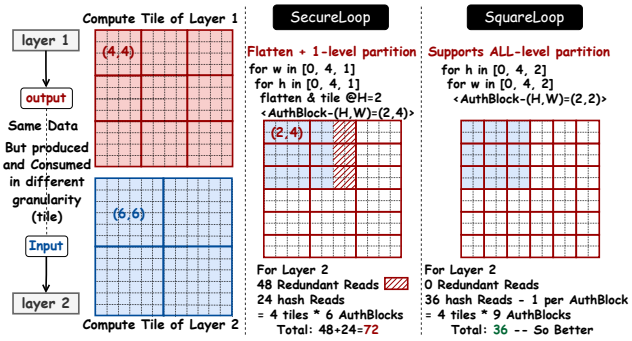


Figure 4: Illustration of difference in AuthBlock partitioning strategies between SecureLoop and SquareLoop. Toy workload for illustration purpose: layer 1 produces the output tensor of shape $H \times W = 12 \times 12$, which becomes the input of layer 2. Layer 1 produces the tensor tile-by-tile, each tile in red has 4×4 data. Layer 2 consumes the same tensor in 6×6 blue tile. SecureLoop's post-search optimal AuthBlock partitioning strategy flattens each red tile 4×4 in dimension order of $H \rightarrow W$, then chunks the post-flattened 1-D vector with 16 data into equal-length AuthBlock of 8 data of shape (2×4), which is high-level partitioning in a single W dimension. When such data gets read by layer 2 with 6×6-shape tile, 6 AuthBlocks need to be fetched per tile, resulting in a redundant read of 12 data shown in the red shaded area and 6 hash tags (one tag per AuthBlock). This gives an overall 48 redundant reads and 24 hash access for all tiles, in total 72 extra overhead, assuming the size of the hash tag equals to a single data element. By constrast, SquareLoop enables all-level partitioning, which uses the great common factor from both layer-1 tile 4×4 and layer-2 tile 6×6 as the AuthBlock per rank, i.e. (2×2). Such an AuthBlock shape eliminates all redundant reads, minimizing costs to 36 hash tags read.

eliminates redundant data reads, significantly reducing security overhead to only the necessary hash tag reads.

3.2 C2: Over-Ideal Memory Behavior

Limitation. DRAM often comes with a specific line size (e.g. 64 bytes as a single burst read [13]). Only data located on the same line could be concurrently accessed per burst. SecureLoop's performance model overlooks this complexity, assuming an ideal memory system where data is always available at maximum bandwidth. This oversimplification leads to an *underestimation* of memory latency and authentication costs.

Solution. To address this, SquareLoop integrates a sophisticated, nested-loop-based memory model considering data layout, inspired by LayoutLoop [24]. This allows SquareLoop to precisely model the data access patterns within the memory hierarchy, capturing realistic latency and providing a high-fidelity evaluation of the true performance impact of different AuthBlock strategies.

4 Method

4.1 Mapping, Layout and AuthBlock Modeling

We refine high-level concepts (§2.2) with detailed definitions:

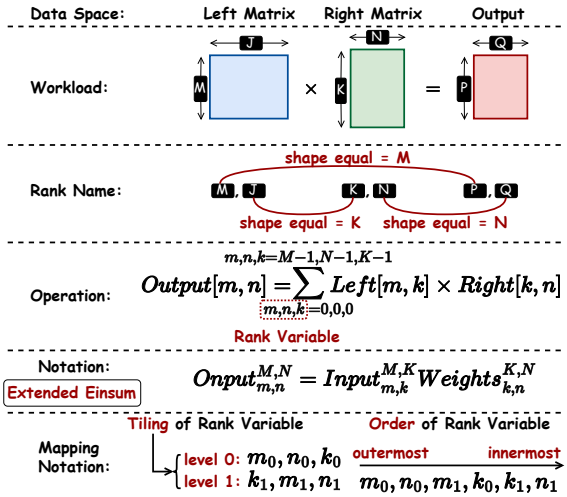


Figure 5: *Mapping* definition of convolution and matrix multiplication. Mapping is defined as extended Einsum over rank, while layout is defined as nested loop of ranks. SquareLoop assigns all shared dimensions an independent rank (shown by red arrow) to enable different partitioning strategies of the shared dimensions in each dataspace.

- **Operation**: The computation pattern applied to multi-dimensional tensors (dataspaces), e.g., convolution or matrix multiplication [7].
- **Mapping**: The temporal and spatial iteration order for executing the operation across all dataspaces [24].
- **Layout**: The assignment of tensor elements to physical memory locations [24]. Representations in Fig. 5 and illustration in Fig. 6.
- **AuthBlock**: The set of data elements over which a hash tag is computed for integrity verification [17]. Illustration in Fig. 6.

Limitation of prior work. FEATHER-style layouts [24] model a layer as nested loops over "dimensions" and require all dataspaces in that layer to share the *same* partitioning along any shared dimensions. This coupling is locally sensible—within a layer, shared dimensions are consumed in lock-step—but becomes globally *sub-optimal* when successive layers adopt different mappings. Different mappings traverse the same dataspace with different temporal/spatial orders and thus prefer conflicting physical layouts; the shared-dimension constraint prevents each dataspace from choosing its own layout to resolve these conflicts. To break such coupling and decouple a dataspace's *logical* indices from its **physical** storage: we introduce a **rank** that provides a per-dataspace physical view independent of the shared logical dimensions.

Rank abstraction. To decouple logical and physical organization, we introduce ***Rank***, which is defined as the name of a logical axis in a tensor, e.g., L, H, W, R, S, M, P, Q. Each rank has a **shape**, i.e., a range of allowed coordinates, e.g.,$[0 \cdots H)$. Shape is represented as rank name (superscript of Extended Einsum notation in Fig. 5) – By convention the name of a rank is also used as the upper (open) limit on the shape range. Coordinates are represented as rank variable (subscript of Extended Einsum notation in Fig. 5). Einsum already separates "rank name (shape)" from "rank variable (iterator)", which

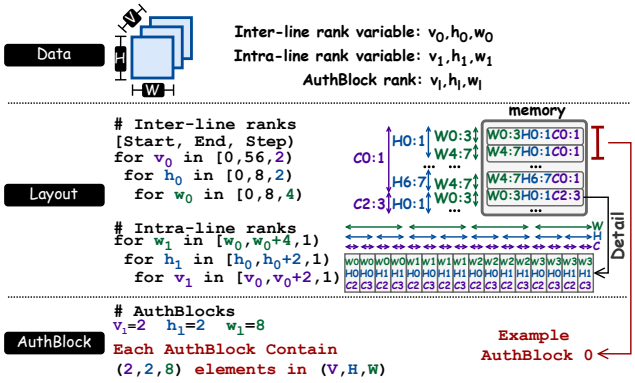


Figure 6: *Layout* and *AuthBlock* representation illustration. SquareLoop extends the nested loop of ranks to specify data belonging to each authentication block (AuthBlock).

decouples the mapping and layout such that different tensors could take its own choice of layouts.

SquareLoop refinement. We refine the layout from a nested loop of "dimensions" into a nested loop of "ranks" by assigning an *independent rank* to every shared dimension in every dataspace (red arrows in Figure 5). Such decoupling allows each data space to adopt its own partitioning shape independently and cover full design choices of layout for each dataspace. After such refinement, logical location (dimension) would be considered in mapping, while physical position (rank) is considered in layout.

4.1.1 AuthBlock Modeling. SquareLoop extends the layout formalism to model AuthBlocks. We also use a nested loop of ranks to specify data belonging to a single AuthBlock, as shown in Fig. 6. This model is capable of representing any hyper-rectangular AuthBlock shape. Each dataspace in each layer has a uniformly shaped AuthBlock. We note that memory line is always an evenly dividable subset of AuthBlock.

4.1.2 Cost Evaluation. The use of authenticated encryption introduces latency overhead to all off-chip memory reads and writes. SquareLoop employs a two-step evaluation process, first assessing the slowdown for individual data spaces and then combining these to determine the total latency. This two-step approach balances high fidelity with evaluation speed.

Dataspace-wise Evaluation. For each dataspace, SquareLoop enumerates all *memory-access cases* that arise from its mapping and AuthBlock shape. Squareloop first performs a cycle-accurate simulation for *one* representative instance of each case, and then statistically weights those latencies by the case–occurrence counts extracted from the mapping. This hierarchical approach, depicted in Fig. 7, provides exact memory latency with high speed.

All-dataspace Evaluation. A DNN layer's execution overlaps three pipelines—input read (❶), compute (❷), and output write (❸), as shown in Fig. 8. Because these phases can be overlapped, the overall latency cannot be computed by simply accumulating the per-dataspace post-slowdown latency. Instead, SquareLoop's evaluation considers the critical path of the pipeline. It first adds slowdown to each phase, and then calculates the critical latency, one worst case where the crypto latency makes all three phases to be memory bound is illustrated in Fig. 8.

4.2 Efficient Inter-Layer AuthBlock Searching

Co-optimizing mappings and security policies (AuthBlocks) across an entire multi-layer network presents a computationally intractable design space. A key challenge is that the optimal AuthBlock shape for one layer often conflicts with the optimal shape for subsequent layers that consume its output data. To address this, we introduce an efficient, multi-phase search algorithm (visualized in that systematically prunes this vast space to find a high-performance, globally-aware solution.

4.2.1 Design Space. Two choices dominate latency:

Mapping (M). has three performance-related design choices.

- **Order**. Any permutation of rank variables.
- **Tiling**. Each rank variable may be hierarchically split into arbitrary levels. When the factor does not divide the loop bound exactly, we will use imperfect factorization [10].
- **Parallelism**. After tiling, any loop can be processed in parallel by marking it as "parallel". The product of all parallel factors can not exceed the hardware's maximal parallelism.

AuthBlock Shape (A). The granularity for encrypting and authenticating data, defined as nested loops of ranks of a dataspace.

- **Tiling**. Each loop may be tiled to at most 2 levels. Zero would be padded for dimensions whose tiling factor does not divide the loop bound exactly.
- **Interline**. Specify how data is being stored across memory lines and the address of each data.
- **Intraline**. Specify the data layout within a single memory line.
- **AuthBlock Shape**. Specify the data of each AuthBlock.

The global search space is the Cartesian product

$$\mathcal{S} = \prod_{l \in \mathcal{L}} \big(\mathbb{M}(l) \times \mathbb{A}\big),$$

where $\mathbb{M}(l)$ is the set of all mappings for layer l and $\mathbb{A}$ is the set of feasible AuthBlock shapes.

4.2.2 Dependency Constraint. For any tensor, its **producer** and all **consumers** must share the *same* AuthBlock shape to pass integrity checks. We call this producer–consumer cluster a *dependency group* g; it is the smallest unit that must agree on the AuthBlock shape A. Given a mapping m_l for layer l, let $\mathbb{A}_l(m_l) \subseteq \mathbb{A}$ denote the AuthBlock shapes that do not introduce extra latency for l under m_l (the layer's "interface"). The group-feasible set is

$$\mathbb{A}_g = \bigcap_{l \in g} \mathbb{A}_l(m_l).$$

If $\mathbb{A}_g$ is empty, we search through whole design space of AuthBlock shapes for a choice ($\mathbb{A}_g^\star$) achieving best performance for all layers in the dependency group.

4.2.3 Breaking Constraints with Re-Hash. Alternatively, for each dependency group, we could also breaks the dependency by changing the AuthBlock shape of the same data generated by procedurer via re-calculating the hashes, termed as "re-hash", allowing distinct AuthBlocks before and after the "re-hash" barrier. A re-hash is inserted *only* when its cost is outweighed by the enabled latency reduction. In practice, we default to re-hashing residual connections,

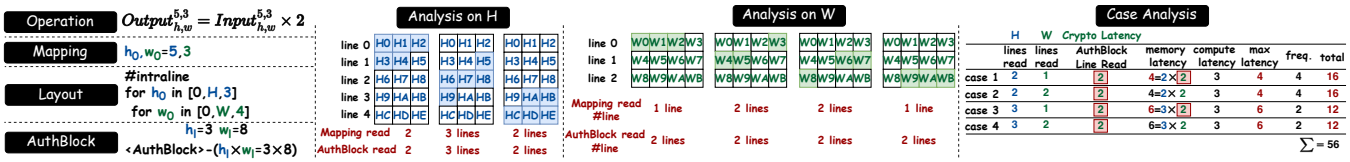


Figure 7: Illustration of the single dataspace cost analysis in SquareLoop. Given a Mapping for computation (e.g., a 5×3 tile) and an AuthBlock for security (e.g., a 3×4 tile), SquareLoop analyzes the access patterns for each dimension (H and W). This identifies all unique memory access scenarios, or "cases." For each case, it calculates the off-chip memory latency (determined by the maximal latency of (1) reading/writing requested data, and (2) reading/writing AuthBlocks. We assume that tags could be read in parallel with the actual data, whose latency could be hidden behind). Then the overall latency is derived from multiplying the maximal value between memory and compute latency by the number of times the case occurs (frequency). By summing the costs of all cases, SquareLoop computes the exact total latency for the given configuration with high speed.

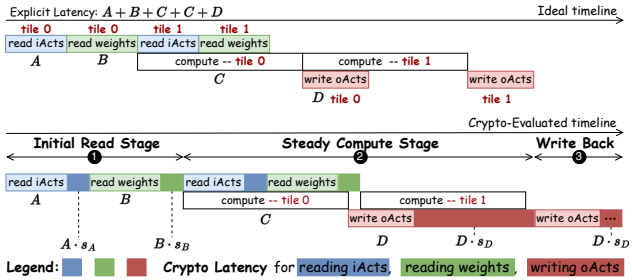


Figure 8: All-dataspace Evaluation. The introduction of authenticated encryption adds to the latency of off-chip memory read/write, which slows down all three phases individually. SquareLoop considers all cases and then weighted-sums all critical latencies with their corresponding occurrences.

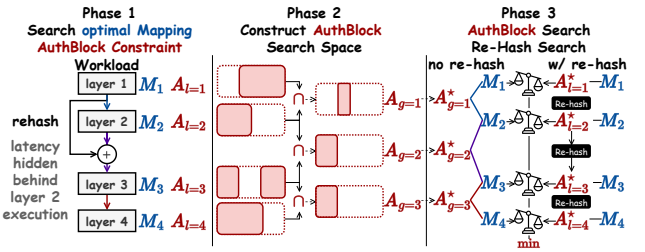


Figure 9: Three-phase mapping-AuthBlock co-search illustration. Step 1 searches optimal mapping ($M_{<layerID>}$) and AuthBlock constraint ($A_{<layerID>}$) for each layer. Step 2 intersects AuthBlock constraint for each dependency group to obtain optimal AuthBlock constraint per group ($A_{g=<ID>}$). Step 3 searches the optimal AuthBlock shape for each group ($A^{\star}_{g=<ID>}$) and then evaluate performance (left). It further inserts rehash to separate dependency group into disjointed layers to use their preferred layer-wise optimal AuthBlock shape ($A^{\star}_{<layerID>}$). The optimal latency among these two choice for each dependency group will be taken.

whose cost can be fully overlapped with the following convolution in our workloads.

4.2.4 Search Procedure. The algorithm takes three phases (Fig. 9):

Phase 1: Per-layer mapping. For each layer l, explore (mapping, AuthBlock constraint) pair to minimize its own latency and energy.

$$(m_l, A_{l=l}) = \arg\min_{A \in \mathbb{A}} \min_{m \in \mathbb{M}(l)} \text{latency}(l; m, A),$$

Phase 2: Per-group AuthBlock Intersection. For each dependency group g, the group-level feasible AuthBlock set is the intersection of its member layers' constraints: $\mathbb{A}_g = \bigcap_{l \in g} \mathbb{A}_l$.

- If the intersection is non-empty ($\mathbb{A}_g \neq \emptyset$), choose the AuthBlock that minimizes the sum of per-layer latencies: $A^{\star}_g = \arg\min_{A \in \mathbb{A}_g} \sum_{l \in g} \text{latency}(l; m^{\star}_l, A)$.
- If empty ($\mathbb{A}_g = \emptyset$), fall back to explore the full set $\mathbb{A}$ to find optimal AuthBlock, i.e. $A^{\star}_g = \arg\min_{A \in \mathbb{A}} \sum_{l \in g} \text{latency}(l; m^{\star}_l, A)$.

Phase 3: Greedy re-hash insertion. Iterate producer→consumer edges in descending benefit; if inserting a single re-hash splits a group and *reduces* total latency (including re-hash cost), perform the split, update group assignments, and recompute affected $\mathbb{A}_g$. Repeat until no profitable split remains.

This procedure co-optimizes mappings and AuthBlocks across layers, enforcing integrity constraints by default while selectively inserting re-hash barriers to unlock additional end-to-end gains.

5 Evaluation

We evaluate SquareLoop (S^2Loop) on a set of AI accelerators (Eyeriss [1], Edge TPU [14], and SIGMA-like [22]) with parallel AES-GCM used for the cryptographic engine under dense machine learning workloads (ResNet18 [9], MobileNetV3 [11], and Bert [3]). The *S^2Loop is validated against RTL simulation of a taped-out in-house accelerator.*

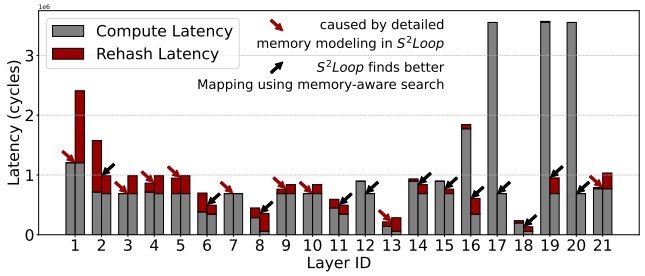


Figure 10: Layer-wise latency comparison for authenticated ResNet-18 on Eyeriss [1] (SecureLoop-left vs. S^2Loop Right)

5.1 Individual Layerwise Evaluation

Fig. 10 distills the performance gap between S^2Loop and the SotA SecureLoop on an authenticated ResNet-18 mapped to an Eyeriss accelerator. Because S^2Loop's rehash bars are 1.5~2× and 6× taller

in layers 3-5, 7, 9 and layer 1. This apparent increase is not a slowdown—it is a correction that exposes detailed on-chip memory costs that bandwidth-based memory modeling in SecureLoop overlooks.

More important is what happens to the compute latency. By (1) augmenting AuthBlock to cover full design space and (2) enabling layout-mapping co-search instead of standalone mapping search followed by a layout search, S^2Loop trims compute time in 18 of 21 layers: 1.1–1.4 × in the early layers with big height and width; 1.8–2.6 × in the middle; and up to 4.5~4.8 × in the deepest layers where feature maps are small and single-rank tiling wastes bandwidth. The richer design space aligns AuthBlock shape with compute tiles, eliminates redundant reads, and fully exploits on-chip reuse that SecureLoop's single tiling based AuthBlock strategy cannot reach.

Aggregated across the network, these per-layer gains yield a 1.32 × geometric-mean speed-up, cutting end-to-end authenticated inference from 9.8 M cycles. Thus, a more accurate cost model plus two-dimensional AuthBlock exploration improves the fidelity of S^2Loop facilitating the discovery of faster mappings. From the roofline view, S^2Loop improves the effective memory bandwidth than SecureLoop.

5.2 Interlayer Evaluation

Setup: SquareLoop's inter-layer search is evaluated on six (architecture, model) pairs: {Eyeriss, Edge TPU like, SIGMA like} × {ResNet18, MobileNetV3} (details in open-sourced repository). For each model, we first extract *dependency chains* by marking "dependency breakers" (e.g., MaxPooling) that force a full-tensor re-hash and split the model into independently optimizable segments, annotated as "forced rehash" in Fig. 11.

Baseline (left bar): We compare against a *baseline* that breaks all inter-layer dependencies (rehash between every adjacent pair), as shown in the left bar in Fig. 11.

Inter-layer search (right bar): Our approach performs two passes to search the best chain of dependent mapping and AuthBlock, one forward and one backward, and picks the one yielding lower total latency[1]. For every adjacent producer→consumer pair, the pass chooses the best of three actions: (1) break the dependency with an explicit re-hash, (2) constrain the consumer's DRAM-level *layout* and *AuthBlock* to match the producer's output, or (3) constrain the producer's output to the consumer's required input. Constraints are applied only at the off-chip memory level (e.g. DRAM) as hardware could freely reorder data in on-chip memory.

Performance gain across all six (arch, model) pairs. Interlayer search consistently beats always-rehash:

- **ResNet18:** −7% (Eyeriss), −3% (Edge TPU-like), −6% (SIGMA-like) ⇒ **1.08**×, **1.03**×, **1.06**× speedups.
- **MobileNetV3:** −45% (Eyeriss), −43% (Edge TPU like), −41% (SIGMA-like) ⇒ **1.82**×, **1.75**×, **1.69**× speedups.

Two representative cases are shown in Fig. 11: *(a)* ResNet18 on an Edge TPU-like and *(b)* MobileNetV3 on SIGMA-like.

ResNet18→Edge TPU-like (Fig. 11a). The inter-layer search trims total latency by −3% relative to the baseline that re-hashes between all layers. Converted to end-to-end speedup, this is $1/(1-0.03) = \mathbf{1.03}\times$. Such performance gains primarily come from eliminating unnecessary re-hashes where producer/consumer layouts can be matched without inflating layer latency. The modest gain stems from ResNet18's high arithmetic intensity: its convolution layers are predominantly *compute-bound*, such that the increased memory latency caused by rehash could be partially hidden.

MobileNetV3→SIGMA-like (Fig. 11b). Here, inter-layer search reduces total latency by −41%, corresponding to $1/(1-0.41) = \mathbf{1.69}\times$ end-to-end speedup. This substantial improvement is attributable to two main factors. First, MobileNetV3's lower arithmetic intensity and reliance on smaller depthwise and pointwise convolutions make its layers memory-bound, meaning the performance penalty of re-hashing cannot be hidden by computation and directly impacts the critical path. Second, MobileNetV3 features longer dependency chains and fewer dependency breakers (like MaxPooling) compared to ResNet18, such that the baseline contains more rehash, presenting a larger optimization surface for the interlayer search algorithm to exploit.

Discussion: A key enabler of these results is S^{2}Loop's ability to decouple a tensor's physical layout from its logical representation by introducing *independent ranks*. This decoupling is critical for navigating the complex trade-offs in long dependency chains. It allows the search algorithm to constrain a tensor's input and output AuthBlocks and layouts *independently*, removing the indirect, and often conflicting, layout dependencies between non-consecutive layers in a chain. Without this separation, avoiding a re-hash would force the entire dependency chain to adopt a single, unified layout, severely restricting the optimization space.

Consequently, the interlayer search consistently identifies a constrained layout that is more optimal than breaking the dependency with a re-hash. While it is possible for a constrained layout to increase the latency of a single layer, we observe that this is rare. In the vast majority of cases, the algorithm finds at least one layout that satisfies the producer-consumer constraint without any performance penalty, successfully eliminating re-hash overhead while preserving per-layer optimality.

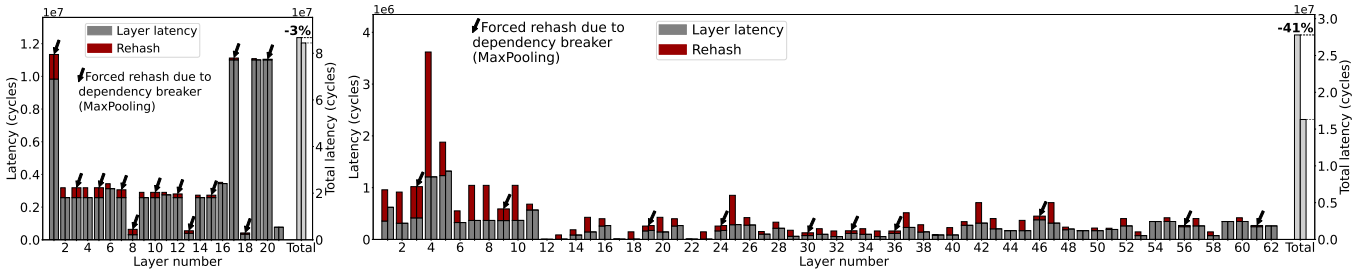
5.3 Wall clock time

Tab. 3 lists wall clock times of running the first 2 phases of the full interlayer search: the single layer search, which finds the locally most optimal layout for each layer, and the layout constraint search described in Section 5.2, which considers the possible layout constraints to eliminate rehash. The single layer search takes significantly longer time than the constrained search. The single layer search explores a huge design space of both the mapping and layouts described in §4.2. On the other hand, the constrained search uses the results of the single layer search to constrain the layouts across dependent layers. Since this only requires the mapping search given the layout constraints, it can be performed faster.

Table 3: Latency of mapping-layout-AuthBlock cosearch

Model	Architecture	Single layer search	Constraint search
ResNet18	Eyeriss	22min	2min
ResNet18	Edge TPU like	17min	5min
ResNet18	SIGMA like	15min	1min
MobileNetV3	Eyeriss	82min	15min
MobileNetV3	Edge TPU like	62min	13min
MobileNetV3	SIGMA like	50min	5min

[1]We note that different order negligibly affects overall latency.



(a) ResNet18 on Edge-TPU-like (b) MobileNetV3 on SIGMA-like, representing flexible reconfigurable accelerators [22, 24].

Figure 11: Interlayer latency comparison (Baseline with all dependencies broken on the left vs. Interlayer search result)[Kyungmi: Can we make the font in the graph much larger?]

5.4 Ablation Study

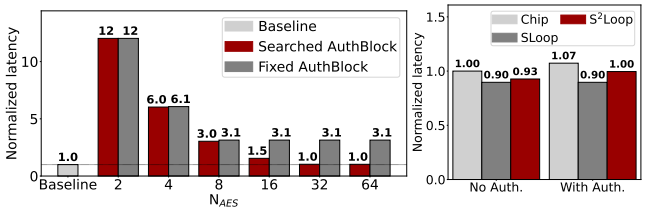
Besides the AuthBlock shape, the number of dedicated AES engines also affects overall performance, as shown in Eq. (2). To evaluate the effect of the number of dedicated cryptographic engines on the latency of authenticated encryption, we sweep the number of engines per data space in Eyeriss for the fourth layer of MobileNetV3 under two experimental setups. In the first setup, we perform a new AuthBlock search for each engine count. In the second setup, we reuse a fixed AuthBlock across all configurations—specifically, the AuthBlock optimized for $N_{\text{AES}} = 1$. These configurations are compared against the baseline of S^2Loop without encrypted authentication. The results are presented in Fig. 12a.

We assume that cryptographic engines are constrained to encrypt and authenticate entire AuthBlocks, meaning that each AuthBlock must be processed by a single engine and cannot be split among multiple engines. Under this constraint, the fixed AuthBlock optimized for $N_{\text{AES}} = 1$ becomes significantly less optimal than searched AuthBlocks for $N_{\text{AES}} = 16$ and more. This is due to its use of large AuthBlocks, which are well-suited for a single engine. Large AuthBlocks reduce the additional per-block overhead for cryptographic operations and tags, but they limit efficient work distributions when multiple engines are available. Indeed, we observe that increasing the number of cryptographic engines beyond 4 provides no further latency improvement when using this fixed AuthBlock. At that point, each engine handles only one AuthBlock per tile, and the authentication and encryption cannot be further parallelized.

Conversely, when re-optimizing the AuthBlock for each engine count, the encryption and authentication latency decreases more significantly. For $N_{\text{AES}} = 32$ and more, the cryptographic overhead over baseline can be almost entirely eliminated.

5.5 RTL Validation

We validate **SquareLoop (S^2Loop)** against an in-house accelerator RTL, using a Conv2d workload with the shape $(M, C, P, Q, R, S) = (64, 64, 32, 32, 3, 3)$, which resembles common 3x3 convolutions in ResNet18. The workload further uses AuthBlock shapes of $(C, P, Q) = (16, 1, 4)$ for the inputs. Two cases - with and without authenticated encryption - are considered. In these scenarios, S^2Loop reports latencies that are 7.2% and 7.3% lower than those of the RTL model, respectively. These discrepancies are attributable to control-related overheads present in the RTL implementation—such as instruction



(a) Effect – Number of Crypto Engine (b) Chip vs. Simulation

Figure 12: Ablation study and RTL validation Evaluation [Kyungmi: Can we make the font size much larger too?]

transfer and chip configuration—which cannot be modeled with an analytical simulator with a simplified architecture description.

We further compare these results to **SecureLoop (*SLoop*)** on the same single layer workload and observe that S^2Loop yields more accurate latency estimates in both cases, by 3.2% without encrypted authentication and 9.4% with encrypted authentication. This improved accuracy stems largely from S^2Loop's more realistic memory simulation as discussed in Section 3.2, which includes the additional latency incurred when fetching the first DRAM tile—a cost that cannot be overlapped with computation. The corresponding results are shown in Fig. 12b.

6 Related Work

Full-memory authenticated encryption (AE) with *AES-GCM* is the standard defence for TEEs, coupling counter-mode encryption and a GMAC tag per authentication block (AuthBlock) [5]. AE thwarts cold-boot and Rowhammer attacks but adds per-access latency, tag storage, and AES/GF-engine area; naïve GPU-class deployments can halve effective bandwidth [8].

To trim this cost, accelerator-aware schemes align AuthBlocks with tensor tiles. *MGX* regenerates counters on-chip and maps AuthBlocks to tiles, cutting security slowdown to <5 % [12]. *TNPU* replaces Merkle trees with a streaming integrity protocol over tile-sized blocks for NPUs [18]. *SecureLoop* formalizes AuthBlock selection: it enumerates 1-D block sizes per layer, then uses simulated annealing to co-optimize layers [17]. S^2Loop highlights two gaps: (i) 1-D blocks cause redundancy in multi-dimensional tensors; (ii) naive memory modeling is inaccurate.

Light-weight, *non-cryptographic* defences trade security for speed. Weight-encoding flags malicious bit flips at runtime [20]; *SEAL* encrypts only "critical" data chosen by sensitivity analysis, gaining 1.4× speed-up while forfeiting full confidentiality [26]. Such methods complement but cannot replace full AE.

Separate literature shows that *layout and dataflow co-design* is vital for performance. *FEATHER* adds hardware reordering, and *LayoutLoop* proves that per-layer (layout,mapping) optimization yields 1.3 $\sim$ 2.9× speed-ups [24]. Yet these works ignore AE overhead, and AE blocks constrain layout.

SquareLoop unifies these threads: it generalizes AuthBlocks to multi-dimensional hyper-rectangles and embeds their cryptographic cost in a fast analytical search. This retains LayoutLoop's layout freedom while avoiding SecureLoop's combinatorial explosion, delivering globally optimal, crypto-aware scheduling.

7 Conclusion

This paper introduced S^2Loop, a framework that overcomes the performance overhead of prior secure ML accelerators by replacing rigid 1D authentication blocks with flexible, multi-dimensional structures. Our novel search algorithm finds mappings considering detailed data layouts and AuthBlock, bringing better choices with 1.32× end-to-end speedup to ResNet18 compared to SotA SecureLoop. S^2Loop's latency predictions are validated to within 7.3% of an RTL implementation—a 9% improvement over existing tools. We open-source S^2Loop to provide a powerful and validated tool for future research in efficient and secure accelerator design.

References

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[2] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 55–71.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[4] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, "Advanced encryption standard (aes)," 2001-11-26 2001.

[5] M. J. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac," 2007.

[6] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, ser. HPCA '03. USA: IEEE Computer Society, 2003, p. 295.

[7] M. Gilbert, Y. N. Wu, A. Parashar, V. Sze, and J. S. Emer, "Looptree: Enabling exploration of fused-layer dataflow accelerators," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 316–318.

[8] 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," *Commun. ACM*, vol. 52, no. 5, p. 91–98, May 2009. [Online]. Available: https://doi.org/10.1145/1506409.1506429

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[10] M. Horeni, P. Taheri, P.-A. Tsai, A. Parashar, J. Emer, and S. Joshi, "Ruby: Improving hardware efficiency for tensor algebra accelerators through imperfect factorization," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 254–266.

[11] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[12] 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, NY, USA: Association for Computing Machinery, 2022, p. 726–741. [Online]. Available: https://doi.org/10.1145/3470496.3527418

[13] *DDR4 SDRAM Standard (JESD79-4B)*, JEDEC Solid State Technology Association, 2021, defines DDR4 DRAM architecture, including line (row) size and organization. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[14] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten lessons from three generations shaped google's tpuv4i : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1–14.

[15] 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.

[16] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1–19.

[17] 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. New York, NY, USA: Association for Computing Machinery, 2023, p. 194–208. [Online]. Available: https://doi.org/10.1145/3613424.3614273

[18] 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.

[19] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[20] Q. Liu, W. Wen, and Y. Wang, "Concurrent Weight Encoding-Based Detection for Bit-Flip Attack on Neural Network Accelerators," in *Proceedings of the 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–8.

[21] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, ser. HASP '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2948618.2954331

[22] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[23] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge tpu accelerators for convolutional neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2102.10423

[24] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching," in *Proceedings of the 51th Annual International Symposium on Computer Architecture*, ser. ISCA '24. Argentina: Association for Computing Machinery, 2024.

[25] F. Yao, A. S. Rakin, and D. Fan, "Deephammer: depleting the intelligence of deep neural networks through targeted chain of bit flips," in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20. USA: USENIX Association, 2020.

[26] P. Zuo, Y. Hua, L. Liang, X. Xie, X. Hu, and Y. Xie, "Sealing neural network models in encrypted deep learning accelerators," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1255–1260.