Towards an Agile Design Methodology for Efficient, Reliable, and Secure ML Systems

Shail Dave[1*], Alberto Marchisio[2*], Muhammad Abdullah Hanif[3*], Amira Guesmi[4*],
Aviral Shrivastava[1], Ihsen Alouani[4], Muhammad Shafique[3]
[1]*Arizona State University (ASU), USA*
[2]*Technische Universität Wien (TU Wien), Austria*
[3]*New York University Abu Dhabi (NYUAD), UAE*
[4]*Université Polytechnique Hauts-De-France (UPHF), France*

shail.dave@asu.edu, alberto.marchisio@tuwien.ac.at, mh6117@nyu.edu, amira.guesmi@uphf.fr
aviral.shrivastava@asu.edu, ihsen.alouani@uphf.fr, muhammad.shafique@nyu.edu

***Abstract*—The real-world use cases of Machine Learning (ML) have exploded over the past few years. However, the current computing infrastructure is insufficient to support all real-world applications and scenarios. Apart from high efficiency requirements, modern ML systems are expected to be highly reliable against hardware failures as well as secure against adversarial and IP stealing attacks. Privacy concerns are also becoming a first-order issue. This article summarizes the main challenges in agile development of efficient, reliable and secure ML systems, and then presents an outline of an agile design methodology to generate efficient, reliable and secure ML systems based on user-defined constraints and objectives.**

***Index Terms*—ML, Neural Networks, Performance, Energy efficiency, DNN, Reliability, Security, Privacy, Agility, Robustness, Codesign.**

I. Introduction

The vast spectrum of real-world applications that can benefit from cutting-edge machine learning algorithms is driving the need for more efficient, dependable and secure ML systems. Moreover, the developments driven by this revolution are leading to new use cases and applications having more stringent constraints. For example, the recent developments in the domains of autonomous driving, robotics, smart healthcare, smart cities, and other always-on use cases are demanding to push the boundaries of energy efficiency, latency, reliability, security, accuracy, throughput, storage, and agility further to meet the ever-increasing real-world requirements. Conventional design methodologies and system stack tools are insufficient to support this revolution, and novel methodologies are required that can adapt to new workloads and efficiently generate the required system stack tools and techniques to build ultra-efficient and robust ML systems. Towards this, we highlight the following key challenges.

Efficiency: Hardware designers have conventionally defined Neural Processing Unit (NPU) accelerators via templates [1]–[3]. An architectural template for an NPU specifies what kinds of computational and memory units can be interconnected and how. Various system stack tools for the NPU, such as cost models, simulators, and compilers, are developed manually by experts, limiting support to only the template architecture [1], [3], [4]. As workloads evolve or application requirements become stringent, novel architectural features need to be integrated and explored [5]. But, tools from the prior system stack cannot be reused much without significant additional efforts for new architecture. Moreover, because design space is limited to exploring hyperparameters of one architecture, efficient architectures from the broad space remain unexplored, leading to inefficient designs. Further, existing accelerator design explorations [3], [6]–[9] use black-box or NPU-agnostic optimizations; without reasoning about the effectiveness of explored solutions, they require thousands of trials or days for the vast space.

Reliability: Conventionally hardware-induced reliability threats are mitigated through redundancy [10], [11], which, together with the compute-intensive nature of state-of-the-art Deep Learning (DL) models, translates to huge overheads. As the deep learning community is progressing towards deeper and more complex networks, it is imperative to consider cost-effective techniques designed by exploiting the intrinsic characteristic of DL models. A few cost-effective techniques have been proposed to address individual reliability issues [12]–[14]; however, the literature still lacks systematic methodologies that can combine such techniques to offer an effective solution against the complete spectrum of reliability threats.

Security and Privacy: Recent research works have highlighted several security and privacy issues [15], [16], which make modern ML networks leak sensitive information and generate malicious results. Due to the variety of Deep Neural Network (DNN) models and vulnerability threats [17]–[19], common trends in the research community are to employ several security-oriented optimizations at different stages of the system design and implementation flow. However, it remains unclear how such security-oriented techniques can be

*These authors contributed equally to this work.

978-1-6654-1060-1/22/$31.00 ©2022 IEEE

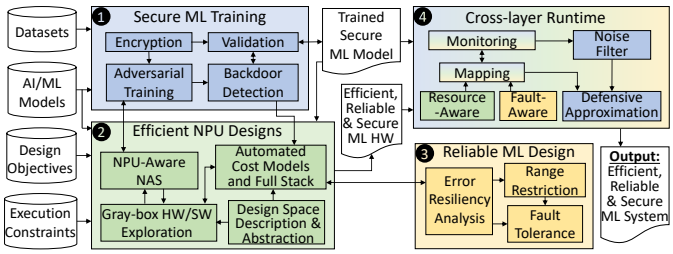

Fig. 1. A cross-layer, agile methodology for designing efficient, reliable, and secure ML systems. Note that, in our approach, reliability and security are not mere after-thoughts but rather first-hand metrics in the codesign flow.

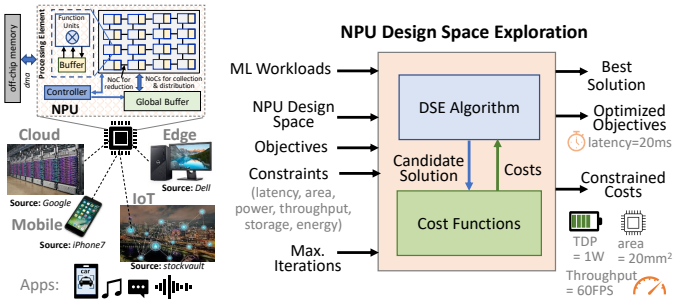

Fig. 2. Exploring efficient hardware/software designs (right) for NPUs under different deployment scenarios (left).

integrated into a framework that combines other optimization objectives for ML systems, such as resiliency, latency, or energy efficiency.

To address the aforementioned challenges, in this work, we present a methodology for designing and deploying efficient, reliable, and secure ML systems. The step-by-step flow of the proposed methodology is shown in Figure 1.

1) **Secure ML Training:** Given the ML models and datasets, security and privacy-preserving optimization techniques, such as adversarial training and backdoor detection are applied on encrypted data, to prevent privacy and adversarial vulnerabilities. Moreover, the resultant secure trained ML model is validated.
2) **Efficient NPU Designs:** Based on design objectives and execution constraints for ML models, the methodology jointly explores the hardware and software configurations of various NPU architectures, as well as NPU-aware neural architectures. Tools for the stack for an NPU, including cost models, are automatically generated. The design space description and the cost models integrates the security and reliability metrics and analysis as well, enabling the design of efficient, reliable, and secure ML hardware.
3) **Reliable ML Design:** The most advanced reliability optimizations are applied. Based on an error resiliency analysis, range restriction is employed to achieve high fault tolerant hardware.
4) **Cross-Layer Runtime Techniques:** During execution, runtime monitoring is conducted to enable resource-aware and fault-aware mapping, and advanced security optimizations such as noise filtering and defensive approximation are applied. Consequently, the output is complete ML system that is efficient, reliable, and secure.

Main contributions of this article are summarized as follows:

1) We advocate for a cross-layer, agile methodology for designing efficient, reliable, and secure and privacy-preserving ML systems. Our approach builds upon the key idea that reliability and security should not be mere after-thoughts but rather first-hand metrics in a co-design flow.
2) To mitigate efficiency-related challenges, we propose an agile design methodology. The key idea is to define design space description, which can enable comprehensive exploration of arbitrary architectures for target ML workloads. Our methodology uses bottleneck analysis for gray-box optimization, yielding explainability for obtained outputs and agile exploration.
3) To mitigate reliability-related challenges, we highlight the most prominent cost-effective fault-mitigation techniques and then present a systematic methodology for effectively combining them.
4) To mitigate the security vulnerabilities and challenges, we discuss the most advanced privacy-preserving and security-preserving techniques for hardware-level intrusions and adversarial vulnerabilities, related to both the DNN-based conventional architectures and SNN-based neuromorphic architectures.

II. Agile Methodology for Designing Efficient ML Systems

The training and inference of ML models are increasingly done on domain-specific accelerators aka *NPUs or Neural Processing Units*. Various applications impose stringent constraints on their execution on NPUs, which the design methodology for the NPU must meet. In this section, we review challenges faced by previous template-based design approaches. Then, we present an agile design methodology for obtaining efficient hardware/software codesigns for NPUs, along with automating full-stack development for a broad set of NPU architectures.

A. NPU Design Requirements and Challenges

NPUs: Examples of NPUs include Google's Tensor Processing Unit (TPU) [20], tensor cores in NVIDIA A100 Ampere architecture, Samsung NPU [21], Sambanova's RDU [22], IBM's AI Accelerator [23], Microsoft Brainwave [24], Tesla's Self-Driving computer [25], Facebook's ML accelerator [26], etc. NPU architectures can be standalone, a co-processor, or a near-data processing engine [27]–[29]. Most NPUs are spatial architectures (e.g., Fig. 2), while others like SpiNNaker and Intel Loihi [30] are neuromorphic processors. We present an agile, end-to-end methodology for obtaining efficient hardware/software designs of NPUs, with spatial architectures as a target example.

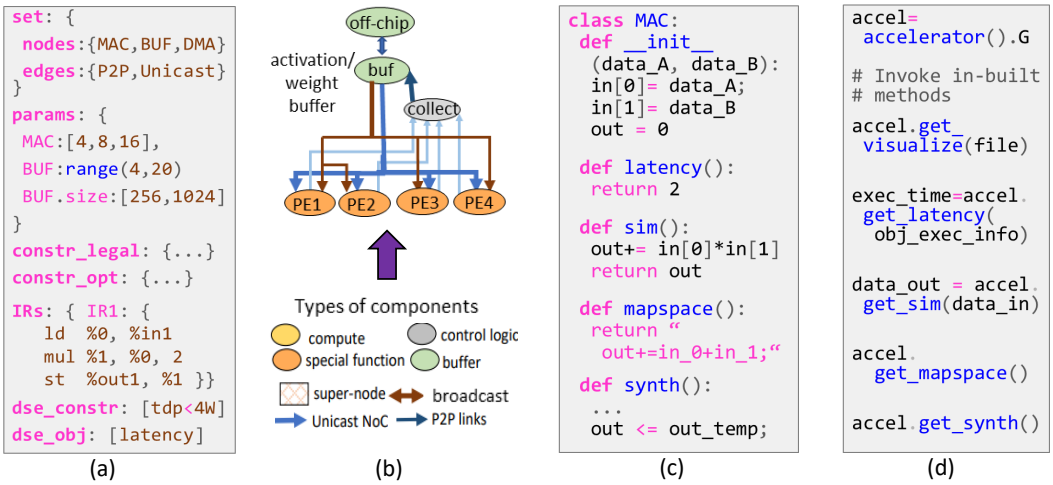

Fig. 3. Examples for DSDL based design explorations. a) Design space specification. b) Obtaining a flow graph of an NPU during exploration. c) Design library for components. d) Design automation for a flow graph.

NPU Design Flow Requirements: Design space exploration (DSE) of NPUs require efficient HW/SW co-design that meets strict constraints of ML applications (Fig. 2), i.e., within desired latency, area, power, throughput, accuracy, storage, energy budgets [31]. Further, we need an agile design methodology because sustaining acceleration becomes challenging as ML workloads evolve. Besides, automatic and efficient construction of system stack is needed, as NPU architectures must adapt to new workloads by supporting specializations like sparsity or novel implementations such as mixed-precision computations [5].

Prior ADL-based Design Methods: Prior NPU design approaches are mostly ADL-based, as in they define an *architecture template* [2], [32]–[35] in *an architecture description language (ADL)* [36], and build a system stack around it [3]. ❶ For a template, the architecture is fixed, i.e., what kind of computation and memory units are interconnected and how. The design space is limited to tuning only hyperparameters of this one architecture. It may lead to system *inefficiency*, as a broad set of architectures is left unexplored. ❷ Expert designers manually build tools like cost models, simulators, and compilers from scratch for the target architecture. This approach lacks flexibility. ❸ Lastly, prior DSE approaches have used *black-box* or *NPU-agnostic optimizations* like simulated annealing or Bayesian [7], [9], limiting the explainability and exploration agility. They cannot reason about high costs obtained or the effectiveness of potential design candidates. As a result, DSE can be costly in design time. The lack of dynamic DSE limits emerging applications such as deploying overlays for new DNN models on reconfigurable infrastructure or scheduling tasks of smart-city applications on distant, edge computing nodes, etc.

B. End-to-end Agile Design Workflow

Our methodology addresses the aforementioned challenges via *design space description language and framework (DSDL)*. The key idea is to allow comprehensive exploration of arbitrary architectures (Fig. 3a), so that much efficient designs for target ML workloads can be explored. With flow graph abstraction of architectures (Fig. 3b), DSDL can automatically build system stack tools like cost models, simulator, and compiler for an arbitrary flow graph (Fig. 3d). Lastly, it uses bottleneck analysis which yields explainability and agile exploration.

Overview of the Framework: Fig. 4 illustrates our agile design methodology for efficient ML accelerator designs. Application developers define ML models in frameworks like PyTorch, Tensorflow, or MXNet. ML compilers obtain graphs of these models and optimize them before lowering them into low-level IRs like LLVM. Developers also specify execution constraints and objectives for DSE. Our methodology enables comprehensive exploration by allowing designers and architects to specify the design space, i.e., components for computation, communication, memory, and control logic. Design libraries provide implementation of these components. Designers can also define rules for legality of the designs and pruning the search space, which are enforced during exploration of flow graphs (hardware architectures). Further, DSDL enables efficient codesign exploration through bottleneck analysis, which optimizes parameters that lower target costs. The functionality of various system stack tools (e.g., latency cost or programming the architecture) are auto-constructed by parsing the flow graph and aggregating the functionality corresponding to the individual nodes and edges. Since it allows determining cost models for arbitrary NPU architectures automatically, our methodology can yield efficient model/NPU codesigns through agile exploration.

Graph Optimizations: Graph-level optimizations help improve computational and memory efficiency by applying various transformations. They include operator fusion, algebric simplification, strength reduction, etc. [37]. Existing ML compilers such as XLA, Glow [38], TVM [8], and nGraph provide such optimizations and lower graphs into intermediate representations (IRs) like LLVM IR [39] or MLIR [40], which can be fed to design exploration mechanism.

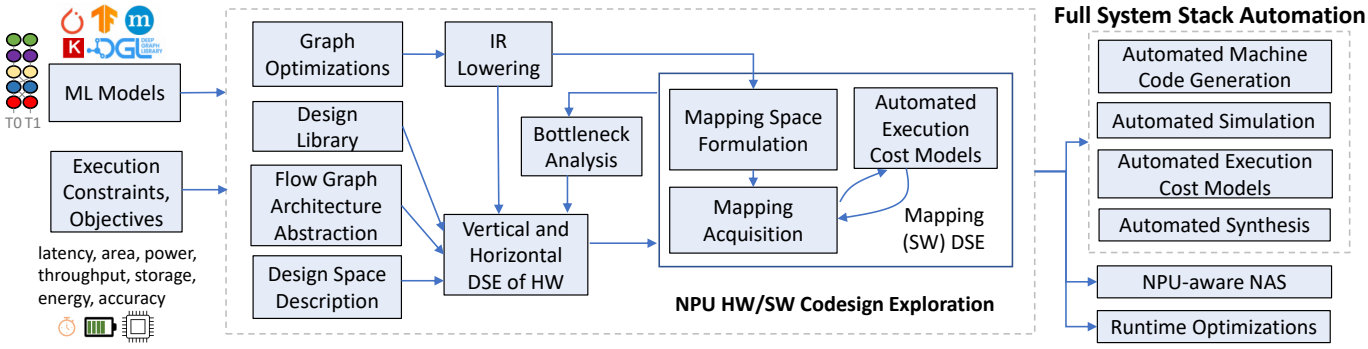

Fig. 4. Agile methodology for designing efficient NPUs with automated full-stack development.

NPU Design Abstraction and Specification: Design abstraction impacts the architectures that can be specified or explored by designers. Our methodology uses a flow graph representation [41]. In an NPU's flow graph, nodes are primary components of computation, memory, control logic [42], [43], or even a sub graph and edges are fixed/reconfigurable interconnects. It allows modular construction of designs and modeling arbitrary hierarchy and grouping of arbitrary components. Moreover, it facilitates the support extension for novel architectures. Further, since algorithms are compiled as data flow graphs [44], [45], they can be conveniently mapped on flow graphs of NPUs. Flow graphs can be constructed through a library, which provides APIs for defining nodes and edges as well as their grouping and replication.

Libraries and Frameworks for Modular Accelerator Construction: They define preliminary components for computation, memory, interconnections, and control [43], [46]. The definitions specify simulation functionality, execution costs like area, latency, or energy [46], or hardware synthesis of each component [43]. Frequently used high-level components can also be defined, such as vector units, systolic array, and computation/memory tiles [47], as specified in libraries like MAGNet [1] and AutoDNNChip [48]. Library also integrates reliability and security costs for components and specialized components such as razor flip flops for detecting timing violations or trusted memory.

Comprehensive NPU Design Space Formulation: It determines NPU architectures and their designs that can be explored, and thus, overall efficiency. DSDL enables specifying vast space (Fig. 3a), as hierarchy and organization of components are explicitly explored [41]. Exploration of such broad space can lead to unseen architectures that are significantly more efficient (e.g., NPUs with no shared buffers for memory-bounded, no-reuse workloads [28]), unlike a specific architecture of the template. In DSDL, designers can define rules in terms of the legality of the generated designs, like what components can be interconnected or not and any relationships among their hyperparameters. In addition, there are also optimization rules for meaningful exploration in a pruned space. For instance, designers may want to explore homogeneous architectural components or a certain hierarchy of buffers. Designers also specify constraints, not just for latency, accuracy, etc., but also for reliability and security, based on such models supported by the NPU design library.

Based on the description, DSDL's iterative exploration takes place in three steps. The first is vertical exploration, which formulates a flow graph (Fig. 3b) and ensures its compatibility with workloads. Then, horizontal exploration figures out optimized hyperparameters for the flow graph (e.g., buffer sizes). For each design, DSDL optimizes the software, e.g., mapping configurations, for it. In fact, all three steps can be jointly explored, especially through an explainable DSE.

Automating Comprehensive Mapping Space Formulation: Mapping space for an NPU encapsulates all schedules (aka iteration spaces in a polyhedral compiler [49], [50]) that are possible corresponding to various loop optimizations like tiling, ordering, and unrolling, when executing a nested loop on an NPU [4], [5], [37]. To develop a compiler for a customized NPU architecture, experts have previously formulated the mapping space manually [1], [4], [34] or relied on NPU-agnostic loop optimizations [39]. Then, compiler mapped operations for a schedule by software pipelining [43], [51]. In contrast, our methodology can automate formulation of the mapping space by deriving program representation for each component and for the overall flow graph [41]. It helps deriving set of possible transformations for given functionality.

Automating Accurate Execution Cost Modeling: For a flow graph, we can obtain accurate execution costs automatically (Fig. 3d) by parsing the graph and aggregating costs of children (Fig. 3c) for each parent node [41]. Thus, it can not only model area or energy accurately (e.g., in Accelergy [46]), but also total latency. Besides, for optimizing execution on commercial hardware, cost models may be obtained through machine/deep learning [52] or profiling [53]. Alongside, our method also allows incorporating security and reliability costs for components, including for data leaking and NBTI for aging, which helps formulating overall security and reliability models for the NPU design.

Automated Machine Code Generation and NPU Simulation: Design library provides simulation functionality for each component (Fig. 3c), and optionally machine instructions to program the component (e.g., a PE). The collective simulation of an NPU architecture occurs as a dataflow through its flow graph. The control triggers that

are necessary for a looped execution and live-in/out data are communicated via a synthetic controller, which also simulates non-accelerator functionality [41]. The machine code can be generated by issuing instructions corresponding to a schedule.

Explainable HW/SW Codesign Exploration: Instead of black-box optimizations, our methodology uses bottleneck analysis for an explainable DSE [41], [54]. Bottleneck analysis for flow graphs is obtained through constructing the cost graphs from cost models, indicating the costs consumed by various datapaths of the NPU architecture. It informs about factors incurring high execution costs for each candidate design explored (e.g., for inference latency, computation time is $n\times$ higher than the time for NoC communication, DMA transfers, or decoding compressed tensors [5], [55]) and values of the parameters that can mitigate them (e.g., number of PEs or function units). With explainability about obtained outputs, iterative DSE can yield quick convergence, enabling DSE of an NPU architecture at even run time!

Runtime Optimizations: They ensure sustaining desired efficiency while meeting all necessary constraints, especially in distributed, multi-tenant [56], [57] environments. Such optimizations include dynamic model selection from a model zoo [58] (trade off accuracy with computations or memory footprint), data layout optimizations [37], and DVFS (dynamic voltage-frequency scaling) [59]. Additionally, runtime can optimize execution by offloading computations of ML models partially to the cloud during processing on resource-critical embedded systems [60], [61]. It may also coordinate contributions of participating devices for efficient convergence of federated learning models [62], [63]. All such runtime optimizations can be incorporated with our framework, as they are either orthogonal or they could leverage obtained cost models for various NPUs and DSE for executing a model's layers onto individual devices.

NPU-aware NAS: Recent techniques including [64]–[68] enable hardware-aware neural architecture search (NAS) [69] of ML models. Our methodology empowers them to quickly explore efficient NPU/model codesigns, since they can leverage accurate cost models of arbitrary NPU architectures and efficient hardware/software explorations, while iterating through various ML model architectures for applications.

III. Designing Reliable ML Systems

The success of DNNs has led to their adoption in safety-critical applications as well [70] [71]. These applications, in general, include all such applications in which even a small error can lead to severe consequences. A few examples of these are autonomous driving, robotics, and healthcare analytics. As DNNs are highly computationally intensive, dedicated hardware accelerators are employed to offer energy-efficient data inference. These accelerators are usually built using modern nano-scale CMOS technology. However, the devices fabricated using such technologies are highly susceptible to different reliability threats such as soft errors, aging, and process variations, which may lead to errors and thereby catastrophic consequences. Moreover, these reliability threats

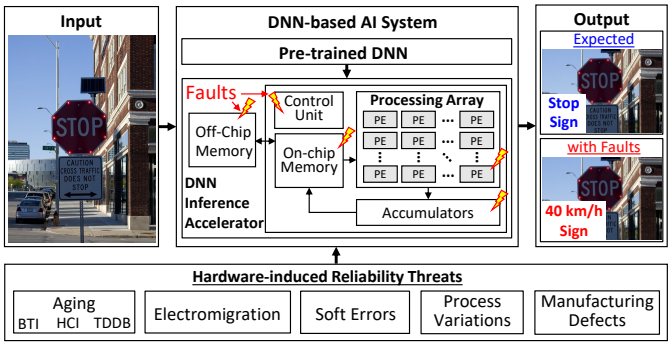

Fig. 5. Reliability issues and their impact on the output of a DNN. (The stop sign image is from the COCO dataset [73])

are becoming an increasingly greater concern with technology scaling.

Different mitigation techniques have been proposed to address reliability threats in modern systems. However, most of these techniques are based on redundancy. For example, Dual Modular Redundancy (DMR) [10] and Triple Modular Redundancy (TMR) [11] are two of the highly effective techniques for addressing different types of reliability threats, specifically soft errors. However, due to the compute-intensive nature of DNNs, these techniques result in ultra-high overheads and cannot be deployed in resource-constrained scenarios. Other techniques such as Error-Correcting Codes (ECC), critical variable re-computation, program duplication and instruction duplication [72] also have similar issues. *Therefore, alternate techniques are required for DNNs to mitigate the effects of reliability threats without incurring huge overheads, ideally at ultra-low cost.* These techniques can be developed by exploiting intrinsic characteristics of DNNs, employing redundancy only for critical neurons/computation, or by transforming critical errors into non-critical ones through system modifications.

A. Reliability Threats

Modern nano-scale devices face various reliability issues. Fig. 5 highlights the main types of reliability threats that can significantly degrade the performance of a DNN system. The figure also highlights a scenario in which these threats can lead to severe consequences, e.g., a fatal accident. The following text provides a brief introduction to different reliability threats.

- **Soft Errors** are bit-flips induced in computing systems due to radiation events. The main sources of these errors are alpha particles emitted by traces of impurities present in packaging materials and neutrons from cosmic radiation [74]. These errors are transient in nature and vanish once new data is written to the faulty locations/cells. However, when present, they can significantly degrade the accuracy of an application.
- **Aging** is gradual degradation of a fabricated device due to various physical phenomena like Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), and Electromigration (EM). It mainly affects the hardware

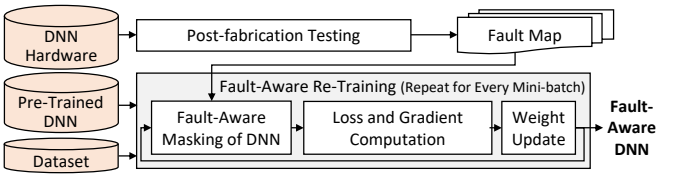

Fig. 6. fault-aware re-training methodology

characteristics of circuits, for example, by increasing the threshold voltage (V_{TH}) of transistors [75] or by eroding the wires. Aging, in general, results in timing errors, which can also translate to permanent faults over time.

- **Process Variations and Permanent Faults** are variations in the characteristics of transistors that occur during fabrication of an integrated circuit. These variations occur due to imperfections in the manufacturing process and affect the performance of the fabricated hardware. Process variations are, in general, addressed by adding guardbands, e.g., by increasing the supply voltage or decreasing the operating frequency of the chip/subsystem to ensure correct functionality. Extreme variations result in permanent faults and significantly affect the yield of the fabrication process.

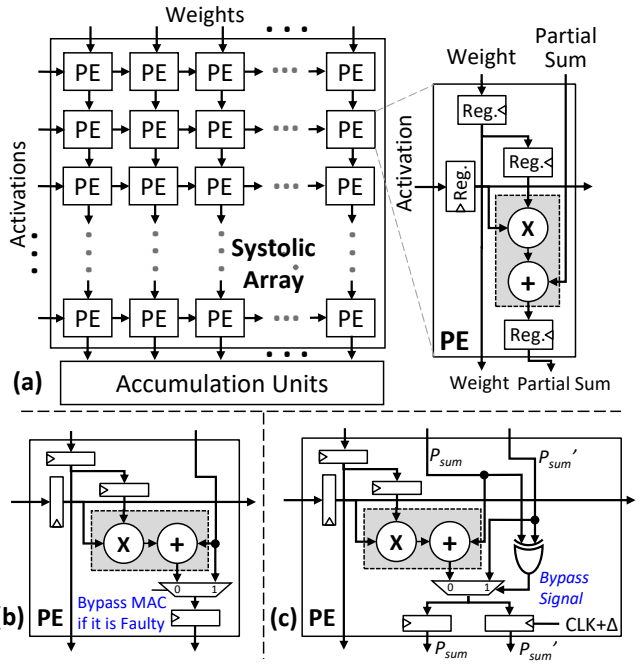

Fig. 7. (a) A systolic array designed for accelerating DNN inference. (b) Modified PE design for for mitigating permanent faults [12]. (c) Modified PE design for mitigating timing errors [13].

B. Cost-Effective Fault Mitigation

Conventional redundancy-based solutions result in huge run-time and/or design-time overheads. To avoid such undesirable overhead costs, different low-cost techniques have been proposed in the literature to mitigated reliability threats in DNN systems. These techniques mainly exploit the intrinsic characteristics of DNNs or work on the principle of transforming critical faults to non-critical through low-cost modifications at the hardware as well as software level. The following text highlight the key concepts proposed for addressing different types of reliability threats in DNN systems without incurring huge overheads.

1) Soft Error Mitigation: Robustness against soft errors is highly important, specifically for safety-critical applications and systems operating under harsh conditions. Studies have shown that these faults can result in significant degradation of the application-level accuracy when they occur at critical locations [76] [14]. As conventional techniques result in huge overheads in DNN-based systems, specialized low-cost techniques are designed to improve the resilience of DNNs against soft errors. To address soft errors in on-chip memory, Azizimazreah et al. proposed a zero-biased SRAM cell design that has a higher probability of switching to '0' state in case an error occurs [77]. The design is based on the observation that 0-to-1 bit-flips in DNNs result in a higher accuracy drop compared to 1-to-0 bit-flips. To address soft errors in the computational array of DNN accelerators, Chen et al. proposed Ranger [14], a range-restriction-based technique for improving the resilience of DNN systems against soft errors. It classifies all the activation values that fall out of a pre-defined range as faulty, and replaces them with a specific value from within the range. A similar technique is proposed in [78] for mitigating soft errors in on-chip memory. Apart from the above-mentioned techniques, algorithm-based fault tolerance, such as checksum-based error detection and correction [79], and selective redundancy-based techniques, in which only the critical neurons are implemented redundantly, have also been proposed to mitigate soft errors at low cost.

2) Permanent Fault Mitigation: Manufacturing defects or process variations induced permanent faults affect the yield of the chip fabrication process. The key goal of permanent fault mitigation techniques is to be able to improve the yield without incurring huge overheads and without affecting the performance of the systems in which the fabricated chips are deployed. As permanent faults are static in nature, the most effective technique for addressing them in DNN systems is Fault-Aware Training (FAT). To address permanent faults in the computational array of DNN accelerators, Zhang et al. proposed the concept of Fault-Aware Pruning (FAP) [12]. The technique exploits the fact that DNNs are, in general, resilient to small amount of pruning. To realize FAP, they added bypass paths in the processing elements of the array, which enable bypassing faulty MAC units. To further highlight the effectiveness of FAP, they coupled it with retraining using the methodology presented in Fig. 6 to achieve close to the baseline performance even at very high fault rates. Fig. 7(b) shows the modifications required in the PEs of the array shown in Fig. 7(a) to realize FAP approach. To address faults in the on-chip weight memory of DNN accelerators, Kim et al. proposed MATIC [80]. It employs a methodology similar to Fig. 6, however using memory fault maps, to generate a

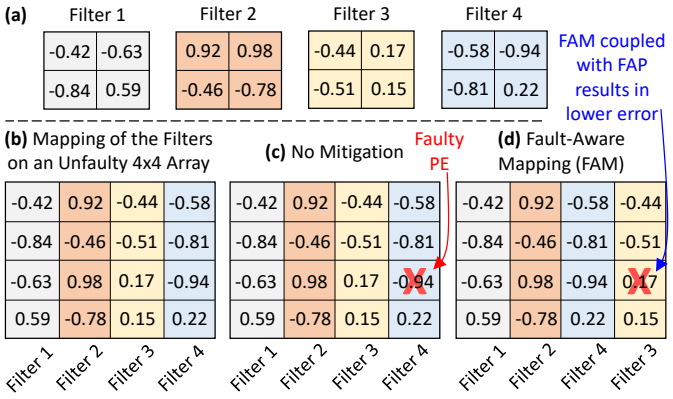

Fig. 8. An example of fault-aware mapping technique using four filter and a 4x4 systolic array.

fault-aware DNN. Note that although the technique has been proposed mainly for improving the energy-efficiency of DNN systems through voltage scaling of on-chip weight memories, it can directly be employed for mitigating permanent faults in the on-chip memories.

Retraining DNNs requires huge computational resources. Therefore, FAT can result in huge design-time overheads, specifically in cases where a DNN has to be tuned for multiple faulty chip and each chip can have a distinct fault pattern. Under such circumstances, FAT cannot be employed. For such cases, Hanif et al. proposed SalvageDNN [81], a saliency-driven fault-aware mapping technique. It works on the principle of mapping less significant weights on the faulty units. Fig. 8 presents an example of the working of SalvageDNN.

3) Aging Mitigation: Aging in CMOS circuits affects the characteristics of transistors and manifests as timing errors. To detect and mitigate these errors in DNN accelerators, Zhang et al. proposed TE-Drop [13], a timing error recovery technique for systolic-arrays in DNN accelerators. The technique employs razor flip-flops for detecting timing errors in MAC units. Then, by exploiting the resilience of DNNs to pruning, mitigate each error by stealing a clock cycle from the corresponding downstream MAC unit and bypassing its update. Fig. 7(c) presents the modifications required in the PEs of the array shown in Fig. 7(a) to realize the concept. Pandey et al. proposed a similar concept, GreenTPU [82], to mitigate timing errors by boosting the operating voltage of erroneous MAC units. Note that ThunderVolt and GreenTPU both have been proposed for boosting energy efficiency of a DNN system through voltage scaling. However, due to their effectiveness against timing errors, they are highly useful against aging in computational arrays of DNN accelerators as well.

To alleviate aging of on-chip SRAM-based memory cells in DNN accelerators, Hanif et al. proposed DNN-Life [83], a framework that employs read and write transducers to achieve ideal aging mitigation. The transducers mix data with random data, while keeping track of the encoding information, to balance the duty-cycle in each cell and minimize NBTI aging.

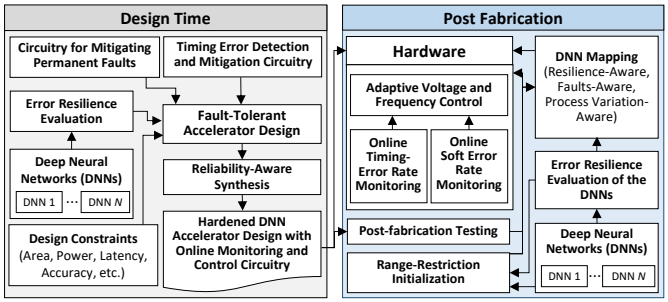

Fig. 9. Overall Methodology for Building Reliable ML Systems [76]

4) Overall Methodology for Building Reliable ML Systems: Fig. 9 shows our overall methodology for building reliable ML systems. In the design phase, a DNN accelerator is designed and then equipped with additional circuitry to support permanent fault mitigation and timing error detection and correction. The design is then passed through a reliability-aware synthesis [84] step, in which vulnerable nodes are selectively hardened to improve the system's resilience against soft errors. The hardware is then fabricated and passed for post-fabrication testing to locate permanent faults. The fault maps are then employed for fault-aware mapping and (if required) fault-aware re-training of DNNs. Once the DNNs have been tuned/adjusted, at the end, range-restriction methodologies are employed to define the bounds of activation values.

IV. Securing ML Systems

Since the DNN-based algorithms have achieved high accuracy for a large variety of tasks [85], [86], it is legitimate to consider their employment in safety-critical applications [19], [87]. However, for these applications, it is crucial to guarantee robustness against security threats [16], [88]–[90] to avoid catastrophic consequences. As shown in Figure 10, such threats include vulnerabilities at the hardware-level, which undermine the correct functionality of computation and memory components, adversarial ML threats that aim at forcing a DNN to output wrong labels in the presence of corrupted inputs, and privacy-related issues. Due to the heterogeneity of the computational engines and their vulnerabilities, we discuss the security threats and their countermeasures for both conventional DNN architectures and SNN-based neuromorphic architectures.

A. Hardware-Level Security Threats

At hardware level, different types of vulnerabilities have been studied. When the DNN weights are stored in DRAM and SRAM memory cells, the bits can be flipped through Row-Hammer attacks [91] or laser injection [92]. While ML algorithms are relatively resilient to random bit-flips, the analysis conducted in NeuroAttack [93] demonstrated that only 4 bit-flips in the most vulnerable weight locations are sufficient to fool DNNs and SNNs on the CIFAR10 dataset. The Bit-Flip Attack methodology [94] uses a progressive

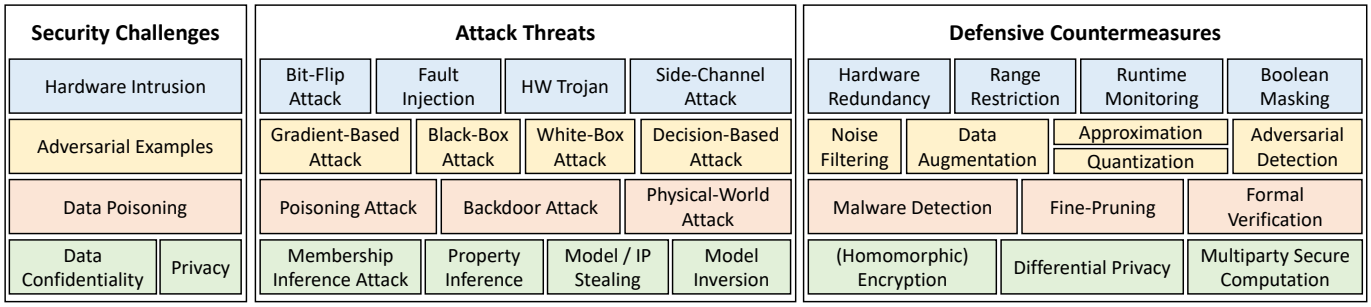

Fig. 10. Overview of the Security threats for ML Systems.

bit search method to find the most vulnerable bits, while the work of [95] proposed a method to generate targeted misclassification through bit-flips. More generically, fault-injection attacks [96] can target not only the weights, but also activation functions [97]. Moreover, the work of [98] studied fault-injection threats for SNNs such as input spike corruption and SNN threshold manipulation.

The defensive techniques aiming at achieving **fault tolerance** are based on deploying hardware redundancy [99], or algorithm-based fault tolerance methods [79] for detecting and correcting the errors in the convolutional layers. Concurrently, Ranger [14] directly rectifies the faulty output by applying a transformation that selectively restricts the value ranges in DNNs, and FT-ClipAct [78] replaces the unbounded activation functions with their clipped versions, thus alleviating the impact of high-intensity faulty activation values.

Since hardware accelerator architectures are likely manufactured in off-shore fabrication facilities, they are vulnerable to **hardware trojans**, which are maliciously-introduced hardware modifications. The work of [100] injected trojans into the computational engine to alter the behavior of DNNs' activation functions. NeuroAttack [93] injects trojans into the weight memory which are triggered when a carefully-crafted adversarial pattern is recognized at the input of DNNs or SNNs. Since they can be detected through runtime monitoring [101], a key feature for the hardware trojans to achieve stealthiness is having extremely low area and power consumption overhead, compared to the victim hardware design.

Moreover, power **side-channel attacks** can be applied to DNN hardware accelerators to recover the input image from the collected power traces, thus threatening their privacy integrity [102]. The work of [103] studied power and time-based side-channel attacks for TinyML targeting embedded microcontrollers to recover the DNN model parameters and inputs. The state-of-the-art defensive methods against side-channel attacks are based on the Boolean masking [104], in which all the hardware blocks for computing the linear and non-linear DNN operations are masked.

B. Adversarial ML

Security for DNN-based Conventional Architectures: While ML applications are already in use in mainstream products and systems, they suffer from vulnerabilities to adversarial attacks that threaten their integrity and trustworthiness. In particular, adversarial examples modify an input to a ML classifier with carefully crafted perturbations chosen by a malicious actor to force the classifier to output a wrong label. If adversaries are able to manipulate the decisions of a ML classifier to their advantage, they can jeopardize the security and integrity of the system, and even threaten the safety of people it interacts with. For example, adding adversarial noise to a stop sign that leads an autonomous vehicle to wrongly classify it as a speed limit sign potentially leads to crashes and loss of life. In fact, adversarial examples have been shown effective in real-world context [105], [106]: that when printed out, an adversarially crafted image can fool the classifiers even under different lighting conditions and orientations. Therefore, understanding and mitigating these attacks is essential to developing safe and trustworthy intelligent systems.

When attacking a DNN-based model, we can distinguish two main attack scenarios based on *attacker knowledge*. A *white-box setting* in which the adversary has complete knowledge of the training data of the victim model in addition to the target model's architecture and parameters. In contrast, in a *Black-box setting*, the adversary has partial or no access to the victim model's architecture and parameters. The adversary uses the results of querying the victim to reverse engineer the classifier and create a substitute model used to generate the adversarial examples. The attacker intention is to slightly modify the source image so that it is classified incorrectly by the target model, without special preference towards any particular output which is known as *untargeted attack*. However, in a *targeted attack*, the attacker aims at a specified wrong target class. Attacks could be performed on different phases of the ML flow, and accordingly can be classified into two categories: *Poisoning or training attacks*, when the attacker attempts to alter the training process by poisoning the training data in order to create specific classification errors [107]. *Backdoor attacks* [108] are also based on providing poisoned data to the victim to train the model with. In fact, the attacker aims to create a backdoor

that allows the input instances that are created using the backdoor key to be classified as a target label. On the other hand, *Inference attacks* or specifically *evasion attacks* [18], [109] are attacks that attempt to perturb an input in a way that it seems normal for a human but is wrongly classified by ML models.

Researchers have devised several defenses in response to these adversarial ML attacks, some of which focus on detection [110], [111], while others focus on prevention and preparation of the ML model to defend against adversarial examples such as adversarial training [18], input preprocessing [112]–[114], Gradient masking based defenses [115], [116]. These defense strategies either alter the DNN structure, tweak the training procedure, or train the model only against known adversarial threats, which limits the defense scope to known vulnerabilities. Another set of defense techniques are inspired by hardware-efficiency techniques such quantization [117], [118]. Authors in [119] proposed Defensive approximation (DA), which leverages approximate computing (AC) to build robust models. DA tackles the problem of robustness to adversarial attacks from a new perspective, i.e., approximation in the underlying hardware and targets both robustness and energy/resource challenges. In fact, DA exploits the inherent fault tolerance of deep learning systems [88] to provide resilience while also obtaining by-product gains of AC in terms of energy and resources. The AC-induced perturbations tend to help the classifier generalize and enhances its confidence and consequently enhance the classifier's robustness.

Security for SNN-based Neuromorphic Architectures: On neuromorphic architectures, the adversarial attacks and defenses can take advantage on different properties. For SNNs based on discrete data, white-box attacks [120] and black-box attacks [121], [122] are generated and deployed. After generating the adversarial examples in the DNN domain, the work of [123] demonstrated that the SNN generated through CNN-to-SNN conversion can be fooled by the same adversarial examples generated in the DNN domain. Moreover, the work of [124] proposed an attack algorithm based on the SNN gradient estimation both in the spatial and temporal domain.

Besides the conventional defense methodologies, recent work have demonstrated that it is possible to fine-tune the SNN structural parameters to improve its robustness. The work of [125] studied the impact of discrete input encoding and non-linear activations, i.e., the leak factor in Leaky-Integrate-and-Fire (LIF) neurons, on the SNN's adversarial robustness. The work of [126] analyzed the SNNs robustness to adversarial attacks with different values of the LIF neuron's firing voltage thresholds and time window boundaries. Using these methods, the SNN robustness results up to 85% higher than its equivalent DNN.

Since event-based sensing with dynamic vision sensors (DVS) is suitable for being deployed with high efficiency on low-power neuromorphic hardware, recent works

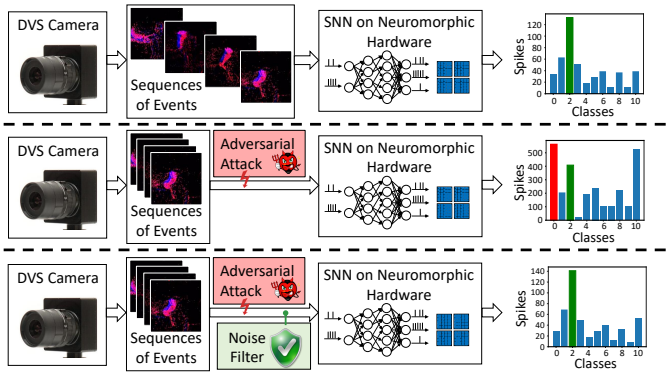

Fig. 11. Example showing event-based adversarial attacks applied to gesture recognition systems implemented on neuromorphic hardware, in which the DVS-noise filter is applied to improve the SNN robustness [134].

demonstrated their applicability in safety-critical applications, such as autonomous driving, recognition and tracking [127]–[129]. Therefore, it is key to analyze the security aspects for event-based data. Towards this, the work of [130] modified the adversarial example generation algorithms of PGD, SparseFool, and Adversarial patches to be applied on event series. The work of [131] generated event-based adversarial examples on 3D point clouds. Moreover, the DVS-Attacks [132] collect a set of stealthy yet efficient adversarial attack methodologies. The Sparse DVS-Attack injects sparse events in time, while the Frame, Corner, and Dash DVS-Attacks inject events in the whole time duration of the event sequences, injecting events in a frame around the sample, in a corner of the image, or in forms of dashed lines, respectively. Moreover, the Mask Filter-Aware Dash DVS-Attack is a modification of the Dash DVS-Attack, which limits the time duration of the perturbations.

While the noise filters for neuromorphic sensors [133] have been originally designed for protecting against thermal noise and junction leakage fluctuation, their application to the input of neuromorphic computing engines serves as a defense mechanism against adversarial attacks [134] (see Figure 11). Among these **DVS-noise filters**, the background-activity filter (BAF) maintains only the events which have spatio-temporal correlation, thus filtering out the spurious events. The mask filter (MF) is designed to filter out the noise activity in pixels which have low temporal contrast. While the BAF makes the neuromorphic system robust against Sparse Attacks, the MF is resistant to Frame, Corner, and Dash Attacks. However, if the attacker has knowledge of the noise filter that is employed to protect the neuromorphic system, the MF-Aware Dash DVS-Attack [132] can potentially break such defense.

C. Privacy

ML algorithms are vulnerable to privacy threats, which are critical when data confidentiality is an issue, e.g., when revealing the identity of the patients in clinical records. Membership Inference Attacks [135] aim at determining whether a data sample belongs to the training dataset. More

generically, Property Inference Attacks [136] infer certain properties that hold only for a fraction of the training data, and are independent from the features that the DNN model aims to learn. On the other hand, Model Stealing methods [137] aim at duplicating the functionality of the ML model and extract its parameters, and Model Inversion Attacks [138] aim to infer sensitive features of the training data.

Towards avoiding these leakages of confidential information, several privacy-preserving techniques can be employed. **Homomorphic Encryption (HE)** ensures that the data remains confidential, since the attacker does not have access to the decryption keys. CryptoNets [139] apply HE to perform DNN inference on encrypted data, and the work of [140] extends the encryption to the complete training process. Another state-of-the-art technique is **Differential Privacy**, which can be guaranteed through the injection of noise to the stochastic gradient descent process (Noisy SGD) [141], or through Private Aggregation of Teacher Ensembles (PATE) [142], in which the knowledge learned by an ensemble of "teacher" models is transferred to a "student" model. Several frameworks based on **Multiparty Secure Computation** have been designed, including SecureML [143], Gazelle [144], and SecureNN [145]. Concurrently, Privacy-Preserving Domain Adaptation techniques [146] preserve the privacy by transferring the knowledge from a labeled source domain to an unlabeled target domain. Moreover, in the PrivateSNN methodology [147] the DNN-to-SNN conversion is followed by weight encryption with spike-based training on synthetic data for privacy-preserving SNNs.

V. Conclusions

Cutting-edge ML algorithms and applications are driving the need for more efficient, reliable, and secure ML systems, requiring a cross-layer codesign methodology. Key takeaways from the proposed methodology are:

- Efficient designs for evolving ML workloads can be obtained by a design space description, which allows exploring various NPU architectures and design metrics, not just efficient designs of an NPU.
- Agile design flow can be enabled by full system stack automation for a wide range of NPU architectures and execution metrics.
- Agile design space exploration of ML systems can be enabled by reasoning about obtained costs and design decisions, e.g., with gray-box optimizations and bottleneck analysis.
- Reliability threats such as soft errors, process variations, permanent faults and aging can be mitigated in a cost-effective manner by exploiting intrinsic characteristics of DNNs. Moreover, a mitigation technique designed to address one threat can (to some extent) mitigate other reliability threats as well. Therefore, thorough exploration should be performed to select the best combination to collectively address all the threats. Moreover, as some of the techniques incur slight energy/power and area overheads, this exploration should be performed collectively with the search for an efficient NPU design.
- Security threats like adversarial attacks, fault injections and privacy attacks can be detected by runtime fault monitoring or adversarial detection techniques; the robustness can be increased by applying adversarial training at design time, or noise filters and defensive approximation at runtime.
- Cross-layer codesign for efficiency, reliability, and security can be enabled by integrating corresponding cost models for design exploration, training of the models, and run-time monitoring and mapping.

Acknowledgment

At ASU, this work was supported in part by NSF under Grant CCF 1723476—NSF/Intel Joint Research Center for Computer Assisted Programming for Heterogeneous Architectures (CAPA). At TU Wien, this work has been supported in part by Intel Corporation through Gift funding for the project "Cost-Effective Dependability for Deep Neural Networks and Spiking Neural Networks", and by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. At NYUAD, different parts of these works were also supported in parts by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG001, Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104, and Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010. At UPHF, this work has been supported in part by RESIST project funded by Région Hauts-de-France through STIMULE scheme (AR 21006614).

References

[1] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[2] Y.-H. Lai, H. Rong, S. Zheng, W. Zhang, X. Cui, Y. Jia, J. Wang, B. Sullivan, Z. Zhang, Y. Liang *et al.*, "Susy: A programming model for productive construction of high-performance systolic arrays on fpgas," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[3] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, p. 27–42.

[4] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[5] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[6] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[7] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 622–636.

[8] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[9] L. Nardi, D. Koeplinger, and K. Olukotun, "Practical design space exploration," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2019, pp. 347–358.

[10] R. Vadlamani et al., "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *IEEE DATE*, 2010, pp. 27–32.

[11] R. E. Lyons et al., "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, vol. 6, no. 2, pp. 200–209, 1962.

[12] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *IEEE VTS*. IEEE, 2018, pp. 1–6.

[13] J. Zhang et al., "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *ACM/IEEE DAC*, 2018, pp. 1–6.

[14] Z. Chen et al., "Ranger: Boosting error resilience of deep neural networks through range restriction," *arXiv preprint arXiv:2003.13874*, 2020.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[16] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Des. Test*, vol. 37, no. 2, pp. 30–57, 2020.

[17] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *2019 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2019, Miami, FL, USA, July 15-17, 2019*. IEEE, 2019, pp. 553–559.

[18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[19] M. Shafique, A. Marchisio, R. V. W. Putra, and M. A. Hanif, "Towards energy-efficient and secure edge AI: A cross-layer framework ICCAD special session paper," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*. IEEE, 2021, pp. 1–9.

[20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[21] J.-S. Park, H. Lee, D. Lee, J. Moon, S. Kwon, S. Ha, M. Kim, J. Park, J. Bang, and S. L. I. Kang, "Samsung neural processing unit: An ai accelerator and sdk for flagship mobile ap," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE Computer Society, 2021, pp. 1–21.

[22] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, and A. Sujeeth, "Accelerating scientific applications with sambanova reconfigurable dataflow architecture," *Computing in Science & Engineering*, vol. 23, no. 02, pp. 114–119, 2021.

[23] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky *et al.*, "A scalable multi-teraops deep learning processor core for ai trainina and inference," in *2018 IEEE Symposium on VLSI Circuits*. IEEE, 2018, pp. 35–36.

[24] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[25] E. Talpes, D. D. Sarma, G. Venkataramanan, P. Bannon, B. McGee, B. Floering, A. Jalote, C. Hsiong, S. Arora, A. Gorti *et al.*, "Compute solution for tesla's full self-driving computer," *IEEE Micro*, vol. 40, no. 2, pp. 25–35, 2020.

[26] M. Anderson, B. Chen, S. Chen, S. Deng, J. Fix, M. Gschwind, A. Kalaiah, C. Kim, J. Lee, J. Liang *et al.*, "First-generation inference accelerator deployment at facebook," *arXiv preprint arXiv:2107.04140*, 2021.

[27] J. H. Kim, S.-h. Kang, S. Lee, H. Kim, W. Song, Y. Ro, S. Lee, D. Wang, H. Shin, B. Phuah *et al.*, "Aquabolt-xl: Samsung hbm2-pim with in-memory processing for ml accelerators and beyond," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–26.

[28] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[29] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Hypar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 56–68.

[30] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[31] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[32] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 159–172.

[33] M. Minutoli, V. G. Castellana, C. Tan, J. Manzano, V. Amatya, A. Tumeo, D. Brooks, and G.-Y. Wei, "Soda: a new synthesis infrastructure for agile hardware design of machine learning accelerators," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–7.

[34] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1160–1174, 2021.

[35] A. Podobas, K. Sano, and S. Matsuoka, "A template-based framework for exploring coarse-grained reconfigurable architectures," in *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2020, pp. 1–8.

[36] S. A. Chin, K. P. Niu, M. Walker, S. Yin, A. Mertens, J. Lee, and J. H. Anderson, "Architecture exploration of standard-cell and fpga-overlay cgras using the open-source cgra-me framework," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 48–55.

[37] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian, "The deep learning compiler: A comprehensive survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 708–727, 2020.

[38] N. Rotem, J. Fix, S. Abdulrasool, G. Catron, S. Deng, R. Dzhabarov, N. Gibson, J. Hegeman, M. Lele, R. Levenstein *et al.*, "Glow: Graph lowering compiler techniques for neural networks," *arXiv preprint arXiv:1805.00907*, 2018.

[39] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, 2004, pp. 75–86.

[40] C. Lattner, J. Pienaar, M. Amini, U. Bondhugula, R. Riddle, A. Cohen, T. Shpeisman, A. Davis, N. Vasilache, and O. Zinenko, "Mlir: A compiler infrastructure for the end of moore's law," 2020.

[41] S. Dave and A. Shrivastava, "Design space description language for automated and comprehensive exploration of next-gen hardware accelerators," *in Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE'22)*, 2022, co-located with the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2022).

[42] S. Pal, S. Feng, D.-h. Park, S. Kim, A. Amarnath, C.-S. Yang, X. He, J. Beaumont, K. May, Y. Xiong *et al.*, "Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 175–190.

[43] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 268–281.

[44] A. Das and A. Kumar, "Dataflow-based mapping of spiking neural networks on neuromorphic hardware," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 419–422.

[45] S. Dave and A. Shrivastava, "Ccf: A cgra compilation framework," 2018.

[46] 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)*. IEEE, 2019, pp. 1–8.

[47] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 13–26.

[48] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 40–50.

[49] M. Wolfe, "More iteration space tiling," in *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, 1989, pp. 655–664.

[50] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," in *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2008, pp. 101–113.

[51] S. Dave, M. Balasubramanian, and A. Shrivastava, "Ramp: Resource-aware mapping for cgras," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[52] A. Adams, K. Ma, L. Anderson, R. Baghdadi, T.-M. Li, M. Gharbi, B. Steiner, S. Johnson, K. Fatahalian, F. Durand *et al.*, "Learning to optimize halide with tree search and random programs," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[53] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[54] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal, "Automated bottleneck-driven design-space exploration of media processing systems," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, 2010, pp. 1041–1046.

[55] S. Dave, A. Shrivastava, Y. Kim, S. Avancha, and K. Lee, "dmazerunner: Optimizing convolutions on dataflow accelerators," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 1544–1548.

[56] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. 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)*. IEEE, 2020, pp. 681–697.

[57] M. H. Quraishi, E. B. Tavakoli, and F. Ren, "A survey of system architectures and techniques for fpga virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2216–2230, 2021.

[58] W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett, "Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3110–3118.

[59] F. Tu, W. Wu, Y. Wang, H. Chen, F. Xiong, M. Shi, N. Li, J. Deng, T. Chen, L. Liu *et al.*, "Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 658–673, 2020.

[60] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," *ACM Transactions on Embedded Computing Systems (TECS)*, 2021.

[61] M. Ghasemi, S. Heidari, Y. G. Kim, A. Lamb, C.-J. Wu, and S. Vrudhula, "Energy-efficient mapping for a network of dnn models at the edge," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2021, pp. 25–30.

[62] Y. Ruan, X. Zhang, S.-C. Liang, and C. Joe-Wong, "Towards flexible device participation in federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021, pp. 3403–3411.

[63] Y. G. Kim and C.-J. Wu, "Autofl: Enabling heterogeneity-aware energy efficient federated learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 183–198.

[64] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4154–4165, 2020.

[65] Y. Zhou, X. Dong, B. Akin, M. Tan, D. Peng, T. Meng, A. Yazdanbakhsh, D. Huang, R. Narayanaswami, and J. Laudon, "Rethinking co-design of neural architectures and hardware accelerators," *arXiv preprint arXiv:2102.08619*, 2021.

[66] Y. Lin, M. Yang, and S. Han, "Naas: Neural accelerator architecture search," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1051–1056.

[67] K. Choi, D. Hong, H. Yoon, J. Yu, Y. Kim, and J. Lee, "Dance: Differentiable accelerator/network co-exploration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 337–342.

[68] G. Li, S. K. Mandal, U. Y. Ogras, and R. Marculescu, "Flash: Fast neural architecture search with hardware optimization," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–26, 2021.

[69] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A comprehensive survey on hardware-aware neural architecture search," *arXiv preprint arXiv:2101.09336*, 2021.

[70] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[71] V. Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[72] M. Shafique et al., "Exploiting program-level masking and error propagation for constrained reliability optimization," in *ACM/IEEE DAC*, 2013, pp. 1–9.

[73] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[74] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE T-DMR*, vol. 5, no. 3, pp. 305–316, 2005.

[75] K. Kang et al., "Nbti induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution?" in *ACM/IEEE ASP-DAC*, 2008, pp. 726–731.

[76] M. A. Hanif et al., "Robust machine learning systems: Reliability and security for deep neural networks," in *IEEE IOLTS*, 2018, pp. 257–260.

[77] A. Azizimazreah, Y. Gu, X. Gu, and L. Chen, "Tolerating soft errors in deep learning accelerators with reliable on-chip memory designs," in *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2018, pp. 1–10.

[78] L. H. Hoang, M. A. Hanif, and M. Shafique, "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*. IEEE, 2020, pp. 1241–1246.

[79] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "FT-CNN: algorithm-based fault tolerance for convolutional neural networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 7, pp. 1677–1689, 2021.

[80] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze, and V. Sathe, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1–6.

[81] M. Hanif at al., "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware

mapping," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, 2020.

[82] P. Pandey, P. Basu, K. Chakraborty, and S. Roy, "Greentpu: Improving timing error resilience of a near-threshold tensor processing unit," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[83] M. Abdullah Hanif and M. Shafique, "Dnn-life: An energy-efficient aging mitigation framework for improving the lifetime of on-chip weight memories in deep neural network hardware architectures," *arXiv e-prints*, pp. arXiv–2101, 2021.

[84] D. B. Limbrick, N. N. Mahatme, W. H. Robinson, and B. L. Bhuva, "Reliability-aware synthesis of combinational logic with minimal performance penalty," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2776–2781, 2013.

[85] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[86] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, p. 113, 2020.

[87] M. A. Neggaz, I. Alouani, S. Niar, and F. J. Kurdahi, "Are cnns reliable enough for critical applications? an exploratory study," *IEEE Des. Test*, vol. 37, no. 2, pp. 76–83, 2020.

[88] M. A. Neggaz, I. Alouani, P. R. Lorenzo, and S. Niar, "A reliability study on cnns for critical embedded systems," in *36th IEEE International Conference on Computer Design, ICCD 2018, Orlando, FL, USA, October 7-10, 2018*. IEEE Computer Society, 2018, pp. 476–479.

[89] M. Naseer, M. F. Minhas, F. Khalid, M. A. Hanif, O. Hasan, and M. Shafique, "Fannet: Formal analysis of noise tolerance, training bias and input sensitivity in neural networks," in *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*. IEEE, 2020, pp. 666–669.

[90] A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique, "Capsattacks: Robust and imperceptible adversarial attacks on capsule networks," *CoRR*, vol. abs/1901.09878, 2019.

[91] Y. Kim, R. Daly, J. S. Kim, C. Fallin, J. 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 *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 361–372.

[92] M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria, "How to flip a bit?" in *16th IEEE International On-Line Testing Symposium (IOLTS 2010), 5-7 July, 2010, Corfu, Greece*. IEEE Computer Society, 2010, pp. 235–239.

[93] V. Venceslai, A. Marchisio, I. Alouani, M. Martina, and M. Shafique, "Neuroattack: Undermining spiking neural networks security through externally triggered bit-flips," in *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8.

[94] 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, Seoul, Korea (South), October 27 - November 2, 2019*. IEEE, 2019, pp. 1211–1220.

[95] J. Bai, B. Wu, Y. Zhang, Y. Li, Z. Li, and S. Xia, "Targeted attack against deep neural networks via flipping limited weight bits," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[96] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, S. Parameswaran, Ed. IEEE, 2017, pp. 131–138.

[97] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Practical fault attack on deep neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 2204–2206.

[98] K. Nagarajan, J. Li, S. Sayyah, S. Kannan, and S. Ghosh, "Fault injection attacks in spiking neural networks and countermeasures," *Frontiers in Nanotechnology*, vol. 3, 01 2022.

[99] E. Ozen and A. Orailoglu, "Boosting bit-error resilience of DNN accelerators through median feature selection," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3250–3262, 2020.

[100] J. Clements and Y. Lao, "Hardware trojan design on neural networks," in *IEEE International Symposium on Circuits and Systems, ISCAS 2019, Sapporo, Japan, May 26-29, 2019*. IEEE, 2019, pp. 1–5.

[101] F. Khalid, S. R. Hasan, O. Hasan, and F. R. Awwad, "Runtime hardware trojan monitors through modeling burst mode communication using formal verification," *Integr.*, vol. 61, pp. 62–76, 2018.

[102] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018*. ACM, 2018, pp. 393–406.

[103] 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 Things J.*, vol. 8, no. 15, pp. 12 079–12 092, 2021.

[104] A. Dubey, R. Cammarota, V. Suresh, and A. Aysu, "Guarding machine learning hardware against physical side-channel attacks," *CoRR*, vol. abs/2109.00187, 2021.

[105] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 1625–1634.

[106] Y. Man, M. Li, and R. M. Gerdes, "Ghostimage: Remote perception attacks against camera-based image classification systems," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14-15, 2020*, M. Egele and L. Bilge, Eds. USENIX Association, 2020, pp. 317–332.

[107] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 6106–6116.

[108] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[109] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2574–2582.

[110] K. Roth, Y. Kilcher, and T. Hofmann, "The odds are odd: A statistical test for detecting adversarial examples," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 5498–5507.

[111] G. Cohen, G. Sapiro, and R. Giryes, "Detecting adversarial samples using influence functions and nearest neighbors," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 14 441–14 450.

[112] F. Khalid, M. A. Hanif, S. Rehman, J. Qadir, and M. Shafique, "Fademl: Understanding the impact of pre-processing noise filtering on adversarial machine learning," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, J. Teich and F. Fummi, Eds. IEEE, 2019, pp. 902–907.

[113] A. Guesmi, I. Alouani, M. Baklouti, T. Frikha, and M. Abid, "Sit: Stochastic input transformation to defend against adversarial attacks on deep neural networks," *IEEE Design Test*, pp. 1–1, 2021.

[114] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau, "Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression," 2017.

[115] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 582–597.

[116] A. Nayebi and S. Ganguli, "Biologically inspired protection of deep networks from adversarial attacks," 2017.

[117] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[118] F. Khalid, H. Ali, H. Tariq, M. A. Hanif, S. Rehman, R. Ahmed, and M. Shafique, "Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks," in *25th IEEE International Symposium on On-Line Testing and Robust System Design, IOLTS 2019, Rhodes, Greece, July 1-3, 2019*, D. Gizopoulos, D. Alexandrescu, P. Papavramidou, and M. Maniatakos, Eds. IEEE, 2019, pp. 182–187.

[119] A. Guesmi, I. Alouani, K. N. Khasawneh, M. Baklouti, T. Frikha, M. Abid, and N. B. Abu-Ghazaleh, "Defensive approximation: securing cnns using approximate computing," in *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*. ACM, 2021, pp. 990–1003.

[120] A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial training for probabilistic spiking neural networks," in *19th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2018, Kalamata, Greece, June 25-28, 2018*. IEEE, 2018, pp. 1–5.

[121] A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique, "Is spiking secure? A comparative study on the security vulnerabilities of spiking and deep neural networks," in *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8.

[122] B. Paudel, A. Itani, and S. Tragoudas, "Resiliency of SNN on black-box adversarial attacks," in *20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Pasadena, CA, USA, December 13-16, 2021*, M. A. Wani, I. K. Sethi, W. Shi, G. Qu, D. S. Raicu, and R. Jin, Eds. IEEE, 2021, pp. 799–806.

[123] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," in *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*. IEEE, 2019, pp. 1–8.

[124] L. Liang, X. Hu, L. Deng, Y. Wu, G. Li, Y. Ding, P. Li, and Y. Xie, "Exploring adversarial attack in spiking neural networks with spike-compatible gradient," *CoRR*, vol. abs/2001.01587, 2020.

[125] S. Sharmin, N. Rathi, P. Panda, and K. Roy, "Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations," in *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXIX*, ser. Lecture Notes in Computer Science, vol. 12374. Springer, 2020, pp. 399–414.

[126] R. El-Allami, A. Marchisio, M. Shafique, and I. Alouani, "Securing deep spiking neural networks against adversarial attacks through inherent structural parameters," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 774–779.

[127] A. Viale, A. Marchisio, M. Martina, G. Masera, and M. Shafique, "Carsnn: An efficient spiking neural network for event-based autonomous cars on the loihi neuromorphic research processor," in *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*. IEEE, 2021, pp. 1–10.

[128] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a DVS camera on the loihi neuromorphic processor," in *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–9.

[129] J. Jiao, H. Huang, L. Li, Z. He, Y. Zhu, and M. Liu, "Comparing representations in tracking for event camera-based SLAM," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 1369–1376.

[130] J. Büchel, G. Lenz, Y. Hu, S. Sheik, and M. Sorbaro, "Adversarial attacks on spiking convolutional networks for event-based vision," *CoRR*, vol. abs/2110.02929, 2021.

[131] W. Lee and H. Myung, "Adversarial attack for asynchronous event-based data," *CoRR*, vol. abs/2112.13534, 2021.

[132] A. Marchisio, G. Pira, M. Martina, G. Masera, and M. Shafique, "Dvs-attacks: Adversarial attacks on dynamic vision sensors for spiking neural networks," in *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*. IEEE, 2021, pp. 1–9.

[133] A. Linares-Barranco, F. Perez-Peña, D. P. Moeys, F. Gomez-Rodriguez, G. Jiménez-Moreno, S. Liu, and T. Delbrück, "Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time FPGA applications," *IEEE Access*, vol. 7, pp. 134 926–134 942, 2019.

[134] A. Marchisio, G. Pira, M. Martina, G. Masera, and M. Shafique, "R-SNN: an analysis and design methodology for robustifying spiking neural networks against adversarial attacks through noise filters for dynamic vision sensors," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021*. IEEE, 2021, pp. 6315–6321.

[135] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 3–18.

[136] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM, 2018, pp. 619–633.

[137] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. USENIX Association, 2016, pp. 601–618.

[138] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 1322–1333.

[139] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016, pp. 201–210.

[140] K. Nandakumar, N. K. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 40–48.

[141] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 308–318.

[142] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, "Scalable private learning with PATE," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[143] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 19–38.

[144] C. Juvekar, V. Vaikuntanathan, and A. P. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds. USENIX Association, 2018, pp. 1651–1669.

[145] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[146] Y. Kim, D. Cho, and S. Hong, "Towards privacy-preserving domain adaptation," *IEEE Signal Process. Lett.*, vol. 27, pp. 1675–1679, 2020.

[147] Y. Kim, Y. Venkatesha, and P. Panda, "Privatesnn: Fully privacy-preserving spiking neural networks," *CoRR*, vol. abs/2104.03414, 2021.