RESEARCH-ARTICLE

Special Sessions - Hardware-Software Co-Design for Machine Learning Systems Made Open-Source

MEHDI BARADARAN TAHOORI, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

VINCENT MEYERS, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

MAHBOOBE SADEGHIPOUR ROODSARI, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

HUASHUANGYANG XU, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

JÜRGEN BECKER, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

TANJA HARBAUM, Karlsruhe Institute of Technology, Karlsruhe, Baden-Wurttemberg, Germany

View all

Published: 28 September 2025

Citation in BibTeX format

CODES+ISSS '25: International Conference on Hardware/Software Codesign and System Synthesis
September 28 - October 3, 2025
Taipei, Taiwan

Conference Sponsors:
SIGDA
SIGMICRO
SIGBED

Special Session – Hardware-Software Co-Design for Machine Learning Systems Made Open-Source

Mehdi Tahoori[1]*, Vincent Meyers[1], Mahboobe Sadeghipour Roodsari[1], Huashuangyang Xu[1], Juergen Becker[1], Tanja Harbaum[1], Felix Frombach[1], Julian Hoefer[1], Georgios Sotiropoulos[1], Jörg Henkel[1], Zeynep Demirdag[1], Heba Khdr[1], Hassan Nassar[1], Ulf Schlichtmann[2], Johannes Geier[2], Philipp van Kempen[2], Georg Sigl[2,5], Stefan Koegler[2], Matthias Probst[2], Jürgen Teich[3], Frank Hannig[3], Muhammad Sabih[3], Batuhan Sesli[3], Norbert Wehn[4], Lukas Steiner[4], Wolfgang Kunz[4], Mohamed Shelkamy Ali[4]
mehdi.tahoori@kit.edu
[1]Karlsruhe Institute of Technology – [2]Technical University of Munich –
[3]Friedrich-Alexander-Universität Erlangen-Nürnberg – [4]University of Kaiserslautern-Landau –
[5]Fraunhofer Institute for Applied and Integrated Security (AISEC)
Germany

Abstract

Chip technologies are crucial for the digital transformation of industry and society. Machine Learning (ML) and Artificial Intelligence (AI) are increasingly shaping both daily life and industrial applications, with AI hardware playing a vital role in enabling efficient and scalable ML deployment. However, significant challenges remain in bridging the gap between ML algorithm development and hardware implementation, particularly for edge ML applications where efficiency, power constraints, and adaptability are critical. In such resource-constrained environments, hardware-software co-design becomes essential to achieve the necessary trade-offs between performance, energy efficiency, and system responsiveness. One of the key bottlenecks in ML hardware development is the lack of seamless integration between ML toolchains and electronic design automation (EDA) tools for hardware synthesis and mapping. Current solutions often require extensive manual optimization and costly proprietary software, limiting accessibility and innovation. Open-source tools can play a transformative role in democratizing ML hardware design, fostering collaboration, and addressing the growing shortage of skilled professionals. This paper covers key aspects of hardware-software co-design for ML systems, such as ML algorithms, hardware design, compiler technologies and system security, with a focus on open-source solutions. We highlight the critical need for open-source toolchains that connect ML model development with hardware synthesis and optimization and present solutions for custom hardware, as well as FPGA accelerators.

ACM ISBN 979-8-4007-1992-9/2025/09
https://doi.org/10.1145/3742873.3756928

CCS Concepts

• **Hardware → Hardware-software codesign**; **Safety critical systems**; • **Security and privacy → Hardware attacks and countermeasures**.

Keywords

Hardware-software co-design, Machine learning

ACM Reference Format:
Mehdi Tahoori, Vincent Meyers, Mahboobe Sadeghipour Roodsari, Huashuangyang Xu, Juergen Becker, Tanja Harbaum, Felix Frombach, Julian Hoefer, Georgios Sotiropoulos, Jörg Henkel, Zeynep Demirdag, Heba Khdr, Hassan Nassar, Ulf Schlichtmann, Johannes Geier, Philipp van Kempen, Georg Sigl, Stefan Koegler, Matthias Probst, Jürgen Teich, Frank Hannig, Muhammad Sabih, Batuhan Sesli, Norbert Wehn, Lukas Steiner, Wolfgang Kunz, Mohamed Shelkamy Ali . 2025. Special Session – Hardware-Software Co-Design for Machine Learning Systems Made Open-Source. In *Proceedings of Special Session – Hardware-Software Co-Design for Machine Learning Systems Made Open-Source (CODES+ISSS '25).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3742873.3756928

1 Introduction

The ongoing digital transformation of society and industry is closely linked to the rapid introduction and use of ML. From smart sensors in industrial automation [1] to personal health monitoring devices [2], AI systems are becoming ubiquitous across all sectors. In 2023 alone, the U.S. Food and Drug Administration (FDA) approved 223 AI-enabled medical devices, up from just six in 2015, demonstrating the scale of AI use in safety-critical applications [3]. This explosion in adoption is driven by both algorithmic progress and unprecedented investment: in 2024, global private AI investment reached USD 252.3 billion, a 26% increase on the previous year [3]. In parallel, regulatory frameworks such as the European Union's AI Act are shaping standards for trustworthy and safe deployment of AI technologies, especially in high-risk applications.

As the complexity and size of these systems increase, so does their need for efficient hardware execution. Specialized AI hardware [4] has become essential when it comes to ensuring the energy

*Corresponding author

efficiency, real-time responsiveness, and cost-effectiveness required to run ML workloads in both data centers and resource-constrained environments [5], such as mobile and edge devices. While cloud infrastructure continues to play a major role in AI processing, the market share and deployment of edge AI systems is accelerating rapidly due to latency, privacy, and energy constraints. This trend complements the overall surge in demand for AI hardware components, with recent forecasts projecting that AI will account for over 70% of the global semiconductor market by 2030 [6].

However, the development of ML systems remains highly fragmented. On the one hand, ML algorithms are developed with high-level frameworks such as PyTorch or TensorFlow, which focus on the expressiveness of the model and training performance. On the other hand, the hardware design and optimization cycle, which is usually controlled by EDA (Electronic Design Automation) tools, is governed by constraints such as latency, power consumption and area and remains largely decoupled from the algorithm development process. This decoupling leads to inefficiencies in mapping ML workloads to hardware, resulting in oversized designs or unmet application-level requirements, especially for embedded or real-time use cases.

This paper outlines the main technical challenges and opportunities in enabling HW-SW co-design for AI systems using open-source frameworks. Our goal is to develop methods and tools that enable joint optimization of ML models and hardware architectures across multiple levels of abstraction. Specifically, we aim to:

- Support application-driven customization of accelerators, including error detection mechanisms and confidence-aware inference.
- Enable design space exploration that includes memory subsystem tuning and architectural specialization.
- Leverage open instruction set architectures and modular IP blocks for configurability and safety extensions.
- Integrate differentiable cost models, ML compiler transformations, and hardware synthesis into a unified co-design stack.
- Incorporate security mechanisms to protect against model theft, fault injection, and side-channel leakage through hard-ware-aware obfuscation, access control, and real-time monitoring.
- Develop toolchains for early-stage simulation and synthesis of secure and robust ML accelerators tailored to deployment contexts such as edge, cloud, or safety-critical environments.

This paper is organized as follows: Section 2 outlines the motivation and challenges of HW-SW co-design for modern AI, especially in edge and embedded systems. Section 3 presents methods for developing open-source AI accelerators with built-in security enhancements. Section 4 explores the hardware design space and DRAM subsystem optimization for ML workloads. Section 5 focuses on the co-design of compilers and accelerators for efficient AI execution. Section 6 addresses the security of ML systems, including fault detection and robustness. Finally, Section 7 concludes the paper with a summary and outlook on open-source HW-SW co-design for AI.

2 Motivation and Background

This section outlines the technological trends, challenges, and emerging opportunities that motivate a shift toward integrated hardware–software co-design for AI systems.

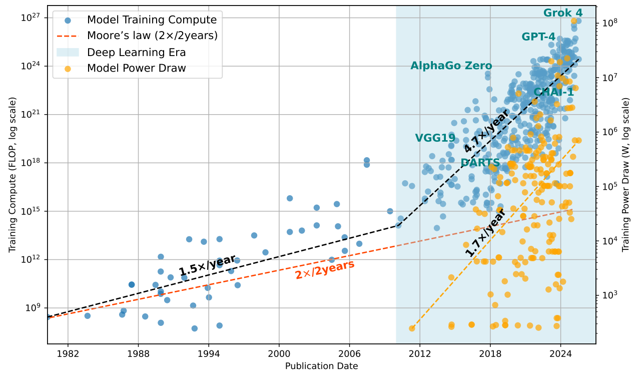

Figure 1: Evolution of training compute (FLOPs) for notable AI models over time [7]. Points represent published models, with a distinct inflection point around 2010 marking the onset of the "Deep Learning Era" (shaded in blue). Dashed lines illustrate compute growth trends before and after 2010, computed from the earliest and latest models in each period.

2.1 The Case for Hardware-Software Co-Design

Modern AI systems increasingly demand custom-tailored hardware solutions to meet their soaring computational requirements and energy constraints. Since the onset of the deep learning era around 2010, the training cost (in FLOPs) of frontier models has surged by several orders of magnitude (see Figure 1), far outpacing Moore's law and reflecting the exponential growth in model scale and complexity. At the same time, as displayed in Figure 2, the economic footprint of AI hardware is expanding rapidly: the AI share of the global semiconductor market is expected to rise from under 10% in 2020 to over 70% by 2030.

These trends underscore the importance of tightly integrating hardware and software design, ensuring that ML models and their target hardware are optimized together rather than separately. This need is even more critical in edge and embedded systems, where the available compute power, energy budget, and physical form factor impose hard constraints on AI deployment. Mapping large AI models to such constrained hardware targets remains a major challenge. To meet application-level requirements without exceeding hardware capabilities, co-design must begin early by embedding hardware constraints directly into model training, architectural exploration, and optimization workflows.

For edge and embedded AI systems, the co-design must explicitly target performance, power consumption, area and cost (PPAC) while meeting accuracy requirements (PPAAC).

Cross-layer modeling: Achieving effective PPAAC co-design requires models that bridge both software and hardware domains. Software-level optimizations must reflect hardware constraints such as available area, latency, and power consumption. Contrarily, hardware models must accurately capture how these constraints impact the accuracy and performance of AI models.

Accuracy under constraints: Deploying large AI models on constrained edge hardware often involves aggressive quantization, pruning, and compression techniques to meet tight latency and energy requirements. Maintaining accuracy under these limitations demands hardware-aware training and optimization methods.

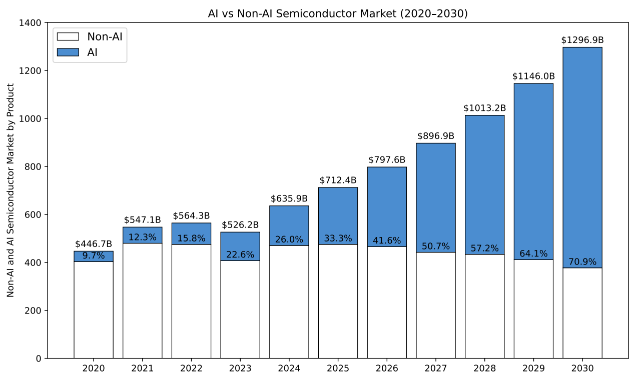

Figure 2: Projected growth of the AI vs. Non-AI semiconductor market from 2020 to 2030 [6]. The increasing share of AI-specific semiconductor products, which grows from 9.7% in 2020 to a projected 70.9% by 2030, based on forecasts.

Specifically, closed-loop approaches are needed where model accuracy is continuously evaluated against realistic hardware constraints during the training process.

Hardware-aware software abstractions: Support for fine-grained optimizations, such as mixed-precision arithmetic and bespoke compute structures, must be exposed through software abstractions that enable dynamic control and customization.

Reliability and security: Co-design must embed functional safety, reliability, and security into the design process from the outset. These aspects cannot be retrofitted but must be integrated within the early phases of hardware-software co-optimization.

Trustworthy deployment: Incorporating these dependability requirements early ensures that edge and embedded AI systems are not only efficient but also reliable and secure, crucial for deployment in safety-critical or mission-critical environments.

2.2 Challenges in Today's Ecosystem

Despite the growing interest in HW-SW co-design for AI, several systemic barriers remain. ML developers are separated from hardware designers by incompatible toolchains, proprietary interfaces and divergent optimization goals. Most current frameworks require manual intervention to map quantized, pruned, or sparsely structured models into hardware. Moreover, there is no unified abstraction that can translate ML-centric metrics (e.g., accuracy, latency under noise, robustness) into hardware constraints in a differentiable manner.

Unlike conventional applications, AI architectures can be refined or learned during deployment (e.g., via Neural Architecture Search), and system-level requirements can change as data sets or tasks change. This requires a paradigm shift: from late optimization to a holistic, early co-design that encompasses the algorithm, architecture, and compiler levels. Differentiable models that capture hardware costs (e.g., area, energy, memory) must become part of the training loop itself. Furthermore, the size and complexity of AI accelerators requires modern approaches for efficient hardware verification [8]. Likewise, security is often neglected in current design processes. Protection against threats such as model theft, fault injection or side-channel leakage is rarely considered during training or synthesis, but treated as an afterthought. This fragmented view hinders the development of trustworthy AI systems, especially for use in adversarial or mission-critical environments. Similarly, safety aspects such as fault detection, fault tolerance or safe evasion mechanisms are not systematically integrated into today's ML and EDA toolchains. Without any co-design support for security-relevant components, developers are faced with a high manual effort and do not have the tools to make principled trade-offs between performance, robustness, and trustworthiness.

2.3 Towards an Open-Source Co-Design Ecosystem

Open-source ecosystems can play a transformative role in overcoming these challenges. Open-source tools serve as a common reference platform, enabling reproducibility, crowd-sourced innovation, and continuous improvements from both academia and industry. These platforms can be easily customized to meet industrial use-cases, facilitating a wider adoption and specializeddeployments.

Traditional closed-source EDA tools are typically application-agnostic, limiting their effectiveness for AI-specific hardware optimization. In contrast, open-source toolchains enable transparent exchange of detailed hardware information and accurate abstraction models, facilitating hardware-aware software optimizations like quantization-aware training and mixed-precision inference.

A stable open-source ecosystem is particularly important for sovereign chip design initiatives, educational accessibility, and industrial innovation [9]. With the environmental impact and cost of AI decreasing by 40% and 30% annually respectively [3], the bottleneck is shifting from technical capabilities to accessibility and openness. Open-source hardware-software co-design toolchains are poised to become essential for driving this next wave of innovation.

3 Accelerator IP Development and Safety Extensions in Open-Source AI Hardware

As AI systems scale in complexity and importance, it is becoming increasingly important to ensure reliable and efficient machine learning on resource-constrained hardware. While traditional Neural Networks (NNs) offer promising and exciting solutions for many relevant computational tasks, they still suffer from several shortcomings for actual applications. These shortcomings include struggles with memory and performance limitations of edge devices, as well as false predictions when confronted with unexpected situations[10]. Furthermore, the continuous scaling down of CMOS technology leads to other issues in terms of reliability, like Single-event upsets (SEUs). While SEUs do not cause permanent damage to the circuit, they can still lead to erroneous system behavior. Therefore, it is crucial to implement effective error detection and correction mechanisms, to enable safety-critical applications under deployment in harsh environments.

The gap between computationally demanding AI-models and limited edge computing hardware is bridged by Hyperdimensional Computing (HDC) [11], a brain-inspired paradigm that utilizes high-dimensional binary vectors and bit-wise operations to enable real-time, low-power classification. In contrast to deep learning, HDC offers inherent noise tolerance and lightweight computation, making it well suited for edge cases. However, its application is

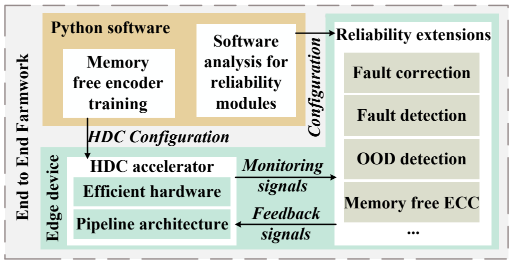

Figure 3: Overview of the end-to-end hardware-software co-design framework for reliable and efficient Hyperdimensional Computing. The system compiles PyTorch-based HDC models into FPGA-ready bitstreams and integrates memory-free encoders, pipeline architectures, and software modules for fault detection, error correction, and out-of-distribution (OOD) detection.

limited by memory-intensive encoding/classification phases and susceptibility to hardware errors during voltage scaling as an edge device. In the following, we present an approach that offers a modular solution to the aforementioned limitations which hinder the widerspread deployment of safety-critical HDC-systems.

An overview of this part of the work can be found in Figure 3. First, we addressed the memory by redesigning the HDC parameter generator so that the parameters no longer need to be stored, but are generated dynamically using cyclic logic. This approach reduces memory requirements without compromising accuracy. Our RTL-based framework supports pipeline parallelism, dynamic memory generation and sparse optimization, achieving up to 4× higher dimensionality on the same FPGA. It compiles PyTorch-trained models into ready-to-use bitstreams for full HDC implementations.

Our open-source ecosystem includes RCE-HDC [12, 13] for memory-free encoding, CED-HDC[14] for real-time error detection, and NUECC-HDC [15] for selective error correction and last but not least, the lightweight out-of-distrubution detector [16]. Together, they enable reliable, fault-tolerant, and energy-efficient edge intelligence. Our work lays the foundation for utilization of HDC in safety-critical edge applications while maintaining a tight power budget. Ongoing research is exploring memory-less ECC and adaptive protection strategies that bring HDC closer to practical, scalable and robust AI at the edge of the network.

Beyond specific techniques like HDC, we aim to provide further solutions to reduce the demand for hardware resources for safety-critical systems in a more generalizable approach, aiming to foster collaboration with an open-source solution. In the state of the art, Triple Modular Redundancy (TMR) remains a widely used technique to address the aforementioned issue of SEUs. In this approach, three identical instances of a circuit operate in parallel, and their outputs are compared. Although TMR offers robust fault tolerance, it comes with significant area and power overhead, which is problematic in scenarios where fault rates are low but the application still requires a certain safety level [17]. The drive for increased productivity by designers demands advanced tools and methodologies that enable the development of hardware systems

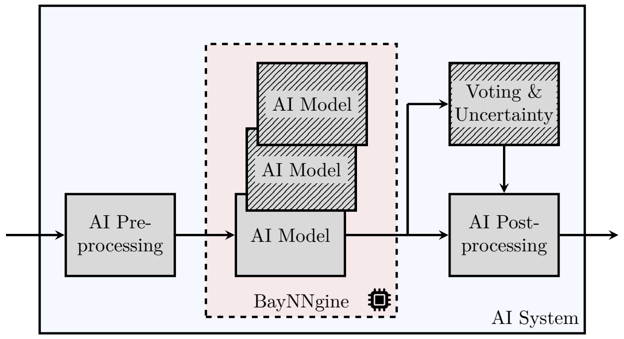

Figure 4: AI System with architectural redundancy patterns as recommended by ISO 8800 [10]. Integrating the BayNNgine allows to efficiently perform approximations of Bayesian Neural Networks (BayNNs) as described in [18].

that meet the fault tolerance requirements of their application environment while minimizing unnecessary redundancy. The safety level of a system can be measured by its diagnostic coverage, which describes the percentage of errors it can detect. This coverage is generally categorized into low (60 %), medium (90 %), and high (99 %) levels [10]. For a system to reach and provide a certain diagnostic coverage, different safety measures can be taken. Simple safety mechanisms like the integration of a parity bit into the data, detect only single-bit errors, but are easy to implement. More complex methods yield higher coverage but increase implementation costs and hardware demands. The choice of the appropriate and necessary safety mechanism is a trade-off between hardware efficiency, desired level of safety, and system robustness.

To simplify and enable the development of hardware for safety-critical systems, this work enhances RISC-V-based AI processor extensions and AI accelerators with appropriate safety mechanisms. By analyzing relevant fault classes and their impact on AI workloads, we can provide hardware generators for configurable and scalable safety extensions for open-source AI-IP-blocks, leveraging the capabilities of modern languages like Chisel. This enables designers to choose the safety level of a hardware module simply by adapting a generation parameter. The resulting interchangeable modules enable designers to easily adapt the safety of a design at a later stage of development without compatibility issues or additional development efforts. By targeting fundamental building blocks, the modular and scalable approach facilitates both the development of new safe and robust systems, as well as the improvement of existing designs to enable new applications in safety-critical domains. While this is a critical step, ensuring dependable AI model inference necessitates addressing additional challenges beyond hardware fault tolerance. Specifically, AI models struggle with unexpected data, leading to false classifications in situations that are new or less optimal in some way. For example, those affected by detrimental environmental factors such as fog or rain [18]. Classical Deep Neural Networks (DNNs) tend to be overconfident in their results in these situations. BayNNs's propose an improvement to this problem. Due to their stochastic properties, BayNNs's and their approximations can estimate the uncertainty of a model's output. Using this information, the system can default to a safe state in the case of high uncertainty. To enable Bayesian inference through

Monte Carlo sampling, we have successfully integrated custom Dropout hardware modules into the data path of FleXNNgine, an open-source and reconfigurable systolic array AI accelerator, which implements an efficient row-stationary dataflow to reduce costly memory accesses [19]. The data flow is depicted in Fig. 4. Further investigation concerns the integration of such safety mechanisms into further AI accelerators to improve their robustness.

To effectively select a diagnostic coverage level for each instance of a hardware module, it is important to understand the fault tolerance of the whole circuit. This also entails having a clear view of the the critical points more likely to lead to erroneous behavior. There exist two main methods for understanding the susceptibility of digital circuits to faults. Physics tools allow for continuous-time simulation of the layout's 3D model, which has significant accuracy but is very time-consuming. Random fault injection campaigns reduce the time and performance overhead since they work at higher abstraction layers but lack accuracy when it comes to providing information under specific environmental conditions.

Consequently, we propose a hybrid error estimation methodology for digital logic circuits. This methodology provides designers with accurate estimations regarding the fault tolerance of their circuit under the specific harsh conditions of their use case. Thus, they can make informed decisions about the trade-off between susceptibility to temporal upsets and efficiency, instead of using techniques like TMR with low susceptibility but significant efficiency overhead. At the gate level of the corresponding circuit, we utilize fault propagation algorithms for fast calculations, eliminating the need for time-consuming simulations. On the layout level, we perform fault characterization of each gate cell through extensive simulations to ensure accuracy. These two approaches in combination deliver fast and accurate insights about the corresponding circuit, enabling informed decisions in an early design phase. Combined with the hardware module safety extensions, this approach allows for the automation of the fault tolerant design process while also ensuring correct functionality of the corresponding system for the target application.

4 Design Space Exploration of Hardware Architectures and DRAM Subsystems for Optimized AI Systems

Deep neural networks consist of different layers with varying computational and memory requirements. A single hardware configuration for entire inference can not efficiently support all layers due to this variability. FPGAs provide runtime reconfigurability, enabling the architecture to adapt to the specific requirements for each layer or group of layers during execution. Changing the hardware configuration for each layer or group of layers leads to significant performance improvement and better resource utilization. However, runtime reconfiguration introduces overhead with latency, typically ranging from several milliseconds to tens of milliseconds depending on the FPGA, configuration interface, and bitstream size.

In order to achieve performance improvements from runtime reconfiguration, we propose grouping layers into *clusters* that share a common hardware architecture. Each cluster consists of a group of consecutive neural network layers, and a *clustering scheme* defines how a neural network is divided into clusters. Finding an optimal clustering scheme and a corresponding hardware configuration for each cluster is challenging due to a vast design space. Hence, there is a need for our Design Space Exploration (DSE) framework that systematically explores design space and identifies the optimal solution identifying optimal clusters. Results in Figure 5 show that our DSE framework is able to find the optimal solution with dividing AlexNet into two clusters. Moreover, the clusters are efficient in utilizing the resources using only 53% of the accelerator slot sized to fit AlexNet accelerator by a state-of-the-art solution [20].

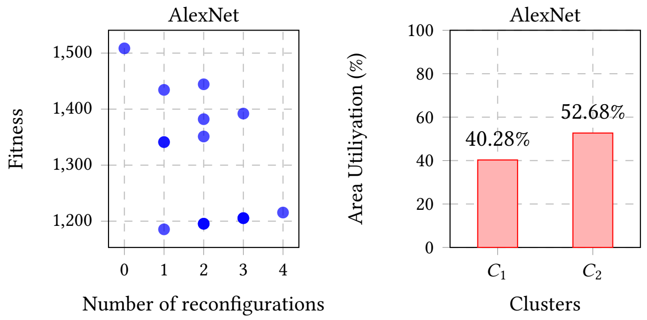

Figure 5: Different reconfiguration points and the final speed up of AlexNet and utilization of the hardware.

As the reconfiguration is a crucial part for our DSE, we need an efficient reconfiguration manager to perform it as quickly as possible. Therefore, we use CoRQ, an open-source reconfiguration manager [21, 22]. CoRQ removes the burden of the reconfiguration from any processing system and independently manages the reconfiguration. If the bitstream is small enough, it can be pre-fetched to a BRAM on chip to reduce the reconfiguration time and make it possible to give real-time guarantees.

After optimizing the accelerator using our DSE framework and using CoRQ to optimize the reconfiguration time, the FPGA can be used to accelerate Multi-DNNs [23]. In such a case, we would get different tasks to be run on the accelerators from different DNN workloads. The tasks can be issued in a certain order but because of different dependencies, they can be stalled till previous tasks end. To optimize this, we use task re-ordering to execute the first available task that has no dependencies. Figure 6 shows the results of using task re-ordering on a Dip-forty board. Compared to a baseline where tasks keep stalling, we achieve an average speed up of 2.1× for an object detection DNN. Integrating smart mapping and scheduling could enhance our DSE framework by further optimizing energy efficiency, as thoroughly studied in non-AI workloads [24].

Our current DSE framework assumes a fixed DRAM interface, while available FPGA platforms offer a variety of different DRAM interfaces. They range from single DDR and LPDDR devices over larger DIMMs up to multiple stacks of High Bandwidth Memory (HBM). These memories differ greatly in their available bandwidth. In addition, depending on the platform, different DRAM controller implementations and interconnect topologies to the remaining system are used, which can largely influence the memory access latency. Overlooking all these differences hides bottlenecks after optimizing compute and reconfiguration. Moreover, we consider that CoRQ prefetches small bitstreams into BRAM, but larger bitstreams

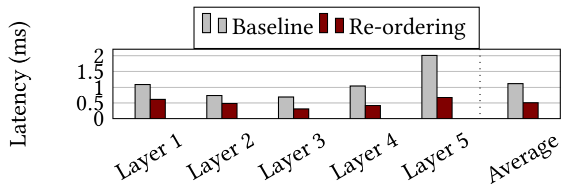

Figure 6: Performance evaluation of object detection DNN on a Dip-Forty board. The figure shows speedup achieved by task reordering compared to the baseline implementation.

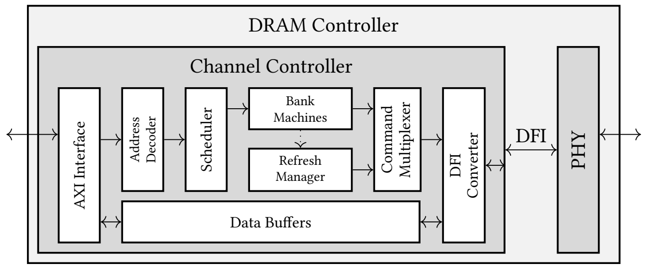

Figure 7: Architecture of parameterizable DRAM controller.

or BRAM-limited platforms rely on off-chip DRAM. In this case, an intelligent DRAM controller is crucial to achieve real-time performance. With our multi-DNN scheduler, off-chip memory faces contention, and without a latency-prioritizing DRAM controller, speedups diminish under heavy workloads because too many memory accesses are issued at once. To harness runtime reconfiguration and multi-DNN acceleration fully, our DSE needs accurate performance models for the DRAM subsystem to co-optimize cluster partitions, bitstream placement, and memory patterns, ensuring performance and timing across FPGA/DRAM combinations. Therefore, as a next step, the DSE framework will be combined with DRAMSys [25]. DRAMSys is an open-source DRAM simulator. Its unique feature is a cycle-accurate DRAM protocol modelling at the speed of an approximately-timed simulation. This allows for both fast and accurate DSE. DRAMSys offers support for all current DRAM standards specified by JEDEC and for various DRAM controller architectures.

In order to deploy the optimized DRAM controller on different FPGA platforms, RTL building blocks for a parameterizable controller architecture as shown in Figure 7 are developed in parallel. Internally, the controller consists of two parts, the channel controller and the Physical Layer (PHY). The channel controller translates incoming read and write commands into a standard-compliant sequence of DRAM commands. It maps system physical addresses to DRAM addresses (address decoder), reorders requests for improved performance (scheduler), tracks and changes the internal state of the DRAM (bank machines) and performs the required refresh for data retention (refresh manager). The PHY first initializes and calibrates the DRAM and afterwards translates the DRAM commands into the required signal levels. While the PHY is always implemented as hard IP on an FPGA, the channel controller can be constructed from FPGA resources. Therefore, we develop different version of its internal building blocks (i.e., with different policies and for different standards) so that an optimized DRAM controller can be assembled.

5 Co-Design of AI Applications: ML Compiler and Accelerator Units

AI workloads include matrix multiplications in vision transformers, convolutions in neural networks, recurrent operations in sequence models, and attention in large language models. Despite differences, they share concentrated computational hotspots. Optimizing these yields major performance and efficiency gains, motivating dedicated accelerators as custom hardware or processor extensions. The RISC-V Instruction Set Architecture (ISA) supports this via Custom Functional Units (CFUs), user-defined modules integrated with the core and invoked through reserved custom opcodes for efficient AI operations.

Several previous works have proposed custom extensions to RISC-V, aiming to deploy DNNs efficiently. Additionally, frameworks such as CFU-Playground have been developed to streamline the design and integration of these RISC-V CFUs. In [26], we proposed extending RISC-V with CFUs to accelerate weight clustering, while in [27], RISC-V CFUs were designed to exploit both unstructured and semi-structured sparsity for acceleration. The broader objective is to develop a suite of CFUs and custom instructions tailored to a wide range of ML workloads.

However, a major shortcoming is that existing compilers do not automatically recognize these customized instructions. To address this gap, Figure 8 illustrates a co-design flow integrating model training, compiler stages, and hardware specialization. The DNN is retrained to compensate for accuracy loss from weight clustering or sparsity, with layers mapped to different CFU configurations at the simulation level. Compiler steps handle code generation (e.g., TVM kernels) and optimizations targeting these custom instructions, which are then deployed on RISC-V cores with integrated CFUs. Open-source tools are utilized at all levels.

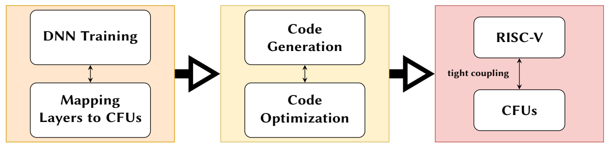

Figure 8: Proposed compiler-based ML deployment flow for co-designed accelerators.

Next, we demonstrate the utility of such an approach using the example of a Weight Clustering Accelerator (WCA). The WCA extension, as proposed in [26], defines custom RISC-V instructions for configuring the codebook (mapping between cluster index and actual `int8` weights), pushing packed weights, as well as updating and reading the internal accumulator register. The accelerator supports three different cluster sizes (2, 4, and 16 elements) and can perform 8 MAC operations per cycle.

The default approach to utilizing these instructions is to write full ML kernels manually. This leads to huge efforts when dealing with a large number of possible layer configurations. An automated integration of the WCA instructions is desirable. However, a SW compiler-level integration based on Directed Acyclic Graph (DAG) pattern matching, such as proposed in [28], is not feasible either, as the WCA instructions rely heavily on the processors' architectural state elements that cannot be modeled in the compiler. Further, the existence of side effects causes the SW compiler to skip optimizing the code near the invocation of the custom instructions. An alternative approach is to integrate the custom instructions on a higher abstraction level, such as the TensorIR (TIR) intermediate representation used by the TVM ML compiler suite [29]. Using a compiler-based ML deployment flow has further advantages compared to TensorFlow Lite Micro (TFLM) [30], which relies on hard-coded "reference" kernels:

- Less runtime and memory overhead due to interpreter-less execution.
- Generates target-optimized (and tuned) kernel code instead of using generic unoptimized implementations.
- Enables performing transformations of the data layouts, computing, and scheduling (at deployment time).

The first step to use the WCA instructions is to transpose the layout of the weights in the model, enabling vectorization over the common input channel axis. Since the weights are constant, this can be precomputed and does not lead to any runtime overhead.

The TVM MetaScheduler (TVM MS) [31] provides automated tensorization, which allows replacing some loop nests in a workload with a call to a "micro-kernel", i.e., a hand-optimized implementation of a matrix-multiplication or vector dot-product. This can be achieved by writing so-called tensor intrinsics, which are composed of two parts: (i) a description of the computation used for loop pattern matching and (ii) an implementation, which is inserted in case of a successful match. TVM further automatically rearranges the loops (tiling and reordering) to increase the likelihood of a successful match.

Using a micro-kernel has several advantages compared to writing a full custom kernel (including scheduling) for a given ML layer. First, it is possible to reuse the same micro-kernel in other layers (such as FullyConnected operations). On the other hand, we have full control over the optimization of the innermost loop (loop unrolling). The micro-kernel also takes care of resetting and reading the accumulator, as well as pushing the correct packed weights.

Since the WCA relies on clusters of values in the weight tensors, it is also important to make the ML compiler flow aware of those clusters.

Layers that do not conform to the constraints of the CFU ISA extension (input channels need to be a multiple of 8) can be invalidated using a tuning post-process. This post-process also takes care of detecting the clusters, compressing the weights, and configuring the codebook, which only has to be done once per layer and therefore should not be done in the micro-kernel itself. TVM's ability to infer the clusters and codebook automatically at code-generation time eliminates the need to manually re-pack the weights using custom modifications of the TensorFlow Lite (TFLite) flatbuffer used to store the model data.

Figure 9 shows the detailed runtime improvements achieved on the dominant layers in the ResNet model. The very first convolution cannot be accelerated because of its input channel count of 3, which is not a multiple of 8 and therefore would under-utilize the WCA, which is designed to perform 8 Multiply-Accumulates (MACs) per cycle. Depending on the layer configuration (kernel size, stride, channel count), the achieved speedups with TFLM + WCA range from 1.9× (Layer 6) to 8.5× (Layer 9). TVM's autotuned

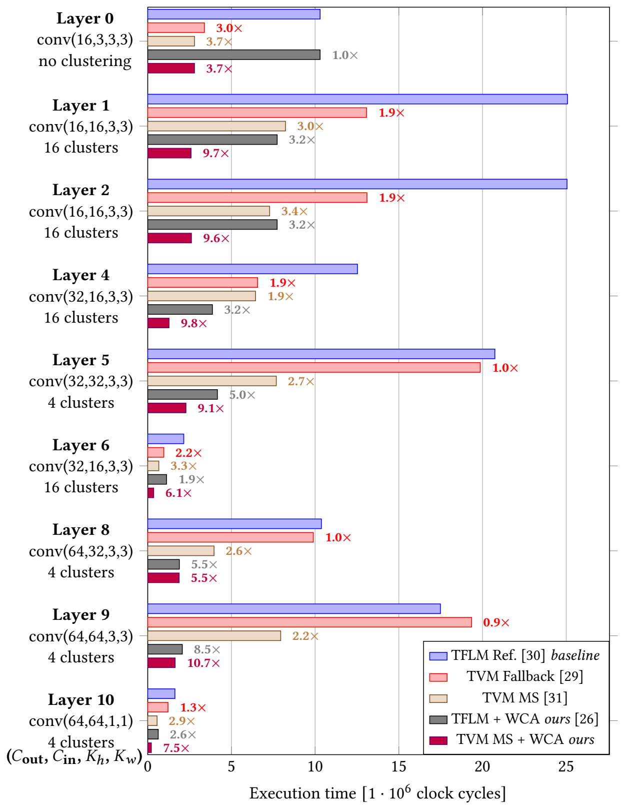

Figure 9: Layer-wise runtime of different ML deployment frameworks and WCA integration. Speedups annotated relative to each layer's baseline.

Conv2D kernels are consistently outperforming the reference TFLM implementations by 1.9× (Layer 4) to 3.7× (Layer 0). The automatic TVM-based integration of the WCA in supported layers yields additional speedups between 1.8× (Layer 6) and 5.1× (Layer 4) compared to TVM MS.

Table 1 provides the end-to-end execution time of the whole model (inference time), consisting of a total of 15 layers of which eight are accelerated using the CFU.

Table 1: Aggregated non-accelerated layers, accelerated layers, and total model inference time $[1 \cdot 10^6$ clock cycles].

Inference time	TFLM	TFLM+WCA	TVM+WCA
Layers w/o accel.	128.5	13.5	3.9
Layers w/ accel.	–	29.2	11.8
Total $\sum$	128.5	42.6	15.7
Speedup vs. TFLM	–	3×	8.2×

6 Security of Machine Learning Systems

The development and deployment of AI accelerators raise significant security concerns, among which model theft is particularly critical [32, 33]. Trained AI models often require substantial investments in data collection, computational resources, and expert knowledge, making them highly valuable intellectual property.

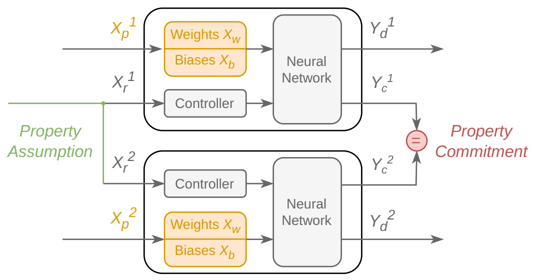

Figure 10: Computational model for model theft in AI accelerators as mentioned in [35].

Unauthorized extraction of these models can result in financial losses, erosion of competitive advantage, and violation of confidentiality. It is therefore essential to ensure that an accelerator is not only functionally correct but also resilient against adversarial attacks that could compromise the model and its associated intellectual property. Although considerable effort is invested in designing increasingly powerful AI workloads, corresponding investment in verifying and securing these systems remains limited [34].

Our work [35] addresses this gap by proposing a formal verification methodology to assess at the microarchitectural level whether an AI accelerator leaks model parameters through timing side-channel attacks. We consider a threat model where an attacker aims to steal model parameters without direct access, instead supplying inputs and observing execution times. If the accelerator's timing behavior depends on sensitive parameters, the attacker can reverse-engineer the model, compromising intellectual property.

We systematically detect parameter-dependent timing variations by employing a computational model (Fig. 10) that instantiates the device under verification twice ("two-safety model"). Each instance receives the same inputs, x_r, but different parameters, x_p, in order to determine whether variations in the parameters result in measurable deviations in execution time. A control output signal y_c indicates the completion of computation. We verify whether there exists a set of inputs where the two instances activate the control output y_c at different time points. If this is possible then the execution time depends on the model parameters, thereby exposing a potential risk of model theft.

We provide different versions of the proof to address scalability challenges and to ensure applicability to larger accelerators while keeping soundness of the proof. The first approach *fully unrolls* the computation in an end-to-end proof, but as expected, struggles with scalability due to the exponential growth of the state space for the solver. The second, more optimized approach, exploits the modularity of the layer computations by applying the proof in a *layer-wise* manner, which reduces unrolling to only the largest layer within the accelerator. The third version is a *single-cycle* proof that can achieve the same results as the fully unrolled and layer-wise proofs but allows us to restrict the considered verification time window to a single clock cycle to determine whether the execution time depends on the model parameters.

We successfully applied our methodology to two types of accelerators: systolic arrays [36] and dataflow-based designs, available in public domain [37, 38]. The proof method is currently being finalized and will be published open-source. Table 2 reports on the proof times and the corresponding results [35]. As shown, the fully-unrolled proof version fails to converge due to the large state space

Table 2: Experimental results for the different methodological approaches and their runtimes.

	Fully Unrolled		**Layer-Wise**		**Single-Cycle**	
Design	Leak?	Runtime (hh:mm)	Leak?	Runtime (hh:mm)	Leak?	Runtime (hh:mm)
Syst. Array	–	time-out	yes	01:30	yes	00:03
FINN-6	no	01:30	no	00:11	no	00:01
FINN-9	–	mem-out	no	00:16	no	00:01
FINN-12	–	mem-out	no	00:24	no	00:01
FINN-15	–	mem-out	no	00:31	no	00:02
FINN-18	–	mem-out	no	00:40	no	00:02

encountered by the solver. In contrast, the layer-wise approach successfully converges within reasonable time, with runtime increasing proportionally to the size of the accelerator design. The single-cycle proof demonstrates the fastest verification times in all cases.

In the case of the systolic array, a dependence of execution time on model parameters was detected. This can be attributed to an optimization feature known as exit branches [39], where the execution time varies depending on branch decisions that involve the model parameters, thereby violating the threat model. This behavior was flagged and reported to the developers. For the case of dataflow architectures, the FINN accelerators proved to be secure with respect to our threat model.

Other attack vectors to consider are Power and Electro-Magnetic (EM) Side-Channel Analysis (SCAs). Several successful attacks utilizing Power or EM SCA against NN implementations both in software [40] and hardware [41, 42] have been published. This demonstrates the threat posed to the intellectual property and private data associated with the deployment of NNs to edge devices and highlights the need for effective countermeasures.

While such countermeasures have been extensively investigated in the context of cryptographic circuits, they generally fall into two categories: hiding and masking. Hiding techniques aim to obfuscate side-channel leakage by introducing additional noise or misaligning trace recordings in the time domain. One prominent hiding method is shuffling, which leverages the large number of operations in NNs with interchangeable execution order to drastically increase the number of side-channel traces an attacker must collect to succeed [43]. Although shuffling's effectiveness can only be confirmed experimentally, it is lightweight to implement and offers strong protection when many parameters are shuffled, as is typical in NNs. However, this means shuffling cannot be integrated early in the design process. In contrast, masking is a provably information-theoretically secure countermeasure designed to ensure that power consumption or EM traces are statistically independent of secret data being processed. Several works demonstrate the effective implementation of several masking schemes in NN hardware accelerators [44–47]. Due to its theoretical security guarantees, masking allows early-stage verification of its effectiveness within the design process by verification of a circuit's side-channel resistance based on its hardware description language definition.

We investigate the suitability of open-source formal verification tools for the application to hardware accelerators of NNs:

CocoAlma [48] involves a multi step process starting with grouping linear combinations that a gate correlates to into correlation

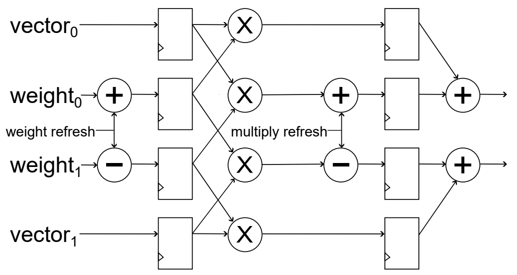

Figure 11: Schematic of an arithmetically masked Multiply Gadget utilizing a domain-oriented masking scheme. As the weight value itself does not change, the 'weight refresh' mask is applied before every multiplication. A 'multiply refresh' mask enables the secure addition of two shares in the upper and lower domains, respectively. Registers are utilized to prevent the propagation of glitches.

sets. These correlation sets are implicitly encoded and formulated as a boolean satisfiability problem. This problem is then solved using a SAT solver. However, the correlation set of the output of a logical operation cannot be directly computed from the correlation sets of its inputs. Therefore, an approximate approach is used, which is prone to false positives.

Silver [49] is based on evaluating the probability distribution of probed wires to determine the statistical independence of the joint distribution of probes from the joint distribution of sensitive inputs. The gate-level netlist is represented as Reduced Ordered Binary Decision Diagrams (ROBDDs) to model the physical circuit. This approach allows verification of formal security definitions such as t-Non-Interference (NI), t-Strong Non-Interference (SNI), or t-Probe-Isolating Non-Inference (PINI), as well as uniformity. While the method offers greater accuracy, it comes at the cost of reduced scalability. As a result, larger designs drastically increase verification times.

Prover [50] is a recently developed method and builds upon Silver by reducing the size of observation and secret sets. It also reorders ROBDDs to increase performance while maintaining accuracy. This method can verify standard probing security, glitch-extended probing security, and uniformity.

As CocoAlma is error-prone and prone to false positives, we generally find it unsuitable for verification of NN accelerators. For example, it falsely classifies the arithmetically masked domain-oriented multiply gadget in Figure 11 as insecure. In contrast, Silver and Prover correctly verify this gadget, indicating that CocoAlma is unsuitable for this task.

Both Silver and Prover face challenges due to a lack of awareness of state and control signals. Additionally, their graph-based models cannot process combinatorial loops. These limitations are particularly problematic when verifying Multiply and Accumulate (MAC) Processing Elements (PEs), which are commonly found in accelerators for fully connected and convolution layers. However, we manage to overcome these limitations by unrolling MAC-PE accelerators into side-channel equivalent feed-forward circuits for verification. Furthermore, Silver scales exponentially with the number of probing positions [49] and is only usable for small gadgets.

In the verification of a complete NN accelerator, it may be necessary to replace the MAC-PEs with its multiply gadget in order to avoid combinatorial loops, which cannot be processed with the verification tools. This replacement is equivalent to adding a single MAC operation and ensures the feed-forward characteristic of the accelerator. Additionally, adding multiple random uniform numbers in the residue-class ring $\mathbb{Z}_{32}$ is equivalent, with regard to side-channel leakage, to adding a single random uniform number.

Furthermore, while individual gadgets or layers might be proven to be side-channel secure, it is important to ensure that no leakage occurs when using layers with different masking schemes, such as arithmetic and boolean masking. We choose Prover for verification of multiple layers, due to its improved performance over SILVER.

7 Conclusion

The development of efficient, safe, and secure machine learning systems is increasingly reliant on tight hardware-software co-design. As AI continues to shape critical applications across industry and society, the need for customizable, trustworthy, and resource-aware ML accelerators becomes more urgent—especially in edge and embedded environments.

This paper has presented a comprehensive perspective on open-source HW-SW co-design for ML systems. We highlighted the inefficiencies of current design flows due to fragmented toolchains and the lack of integration between ML training frameworks and EDA tools. In response, we introduced a modular co-design approach that spans multiple levels of abstraction, enabling application-driven customization of AI hardware.

We explored four key areas: (1) the development of AI accelerators with integrated safety extensions using open hardware; (2) design-space exploration of hardware and memory architectures with DRAM subsystem awareness; (3) compiler-accelerator co-design for optimizing ML execution pipelines; and (4) robust and secure ML hardware systems that incorporate fault detection, OOD detection, and protection against model theft.

Our results demonstrate that safety and security are no longer optional add-ons but must be integral to the co-design process from the earliest stages. By embedding uncertainty-aware inference, error correction mechanisms, and security features like obfuscation and logic locking into the co-design flow, we provide a foundation for building dependable AI systems.

We argue that open-source ecosystems are essential to realize this vision. They democratize access to design tools, facilitate reproducibility, and accelerate innovation across research, education, and industry. The methods and frameworks described in this work are stepping stones toward such an ecosystem. In the long term, our goal is to make hardware-software co-design of efficient, safe, and secure ML systems accessible to a broad community of designers, from academic researchers to industrial practitioners, in order to foster the next generation of trustworthy and adaptive AI platforms.

Acknowledgments

This work is funded by the German Federal Ministry of Research, Technology and Space (BMFTR) in the framework of design tools for sovereign chip development with open source (DE:Sign DI-EDAI, grant numbers 16ME0990K, 16ME0991, 16ME0992, and 16ME0993).

References

[1] S. A. Singh and K. A. Desai. 2023. Automated surface defect detection framework using machine vision and convolutional neural networks. *Journal of Intelligent Manufacturing*, 34, 4, 1995–2011. doi:10.1007/s10845-021-01878-w.

[2] R. Begg, J. Kamruzzaman, and R. Sarker. 2006. *Neural networks in healthcare: Potential and challenges*. Igi Global. doi:10.4018/978-1-59140-848-2.

[3] N. Maslej et al. 2025. Artificial intelligence index report 2025. *The Computing Research Repository (CoRR)*. arXiv: 2504.07139 [cs.AI].

[4] Y. Chen et al. 2020. A survey of accelerator architectures for deep neural networks. *Engineering*, 6, 3, 264–274. doi:10.1016/j.eng.2020.01.007.

[5] X. Xu et al. 2018. Scaling for edge inference of deep neural networks. *Nature Electronics*, 1, 4, 216–222. doi:10.1038/s41928-018-0059-3.

[6] International Business Strategies. 2024. Why AI will propel Semiconductor Market to $1 Trillion and Achieve 1.0nm by 2030. Tech. rep. SEMI, (Nov. 2024). https://www.semi.org/sites/semi.org/files/2024-11/MS-SEMIWEB11.2024.pdf.

[7] Epoch AI. 2024. Data on notable AI models. (June 2024). Retrieved 07/20/2025 from https://epoch.ai/data/notable-ai-models.

[8] K. Vittal. 2024. Advanced SoC verification enables a new era of AI chips. https://www.synopsys.com/blogs/chip-design/advanced-soc-verification-enables-new-era-ai-chips.html.

[9] F. Hannig and J. Teich. 2021. Open source hardware. *IEEE Computer*, 54, 10, 111–115. doi:10.1109/MC.2021.3099046.

[10] 2024. ISO/PAS 8800:2024 Road vehicles — Safety and artificial intelligence. Standard. International Organization for Standardization, CH.

[11] P. Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1. doi:10.1007/s12559-009-9009-8.

[12] M. S. Roodsari, J. Krautter, V. Meyers, and M. Tahoori. 2024. E^{3}HDC: Energy efficient encoding for hyper-dimensional computing on edge devices. In *Proc. of the 34th Int'l Conference on on Field-Programmable Logic and Applications (FPL)*. IEEE, 274–280. doi:10.1109/FPL64840.2024.00045.

[13] M. S. Roodsari, J. Krautter, and M. Tahoori. 2024. OTFGEncoder-HDC: Hardware-efficient encoding techniques for hyperdimensional computing. In *Proc. of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE, 1–2.

[14] M. S. Roodsari, V. Meyers, and M. Tahoori. 2025. CED-HDC: Lightweight concurrent error detection for reliable hyperdimensional computing. In *Proc. of the 43th IEEE VLSI Test Symposium (VTS)*. IEEE, 1–7. doi:10.1109/VTS65138.2025.11022900.

[15] M. S. Roodsari, S. Hemaram, and M. Tahoori. 2025. Non-uniform error correction for hyperdimensional computing edge accelerators. In *Proc. of the 30th IEEE European Test Symposium (ETS)*. IEEE, 1–6. doi:10.1109/ETS63895.2025.11049622.

[16] M. S. Roodsari, V. Meyers, and M. Tahoori. 2025. Lightweight concurrent out-of-distribution detection in hyperdimensional computing hardware. In *Proc. of the 31th IEEE Int'l Symposium on On-Line Testing and Robust System Design*. IEEE.

[17] F. Mireshghallah, M. Bakhshalipour, M. Sadrosadati, and H. Sarbazi-Azad. 2019. Energy-efficient permanent fault tolerance in hard real-time systems. *IEEE Trans. on Computers*, 68, 10, 1539–1545. doi:10.1109/TC.2019.2912164.

[18] J. Höfer et al. BayNNgine: Hardware-enabled bayesian neural network support for dependable AI inference. To appear in Proc. of the IEEE 38th Int'l System-on-Chip Conference (SOCC), (2025).

[19] F. Lesniak, A. Gutermann, T. Harbaum, and J. Becker. 2024. Enhanced accelerator design for efficient CNN processing with improved row-stationary dataflow. In *Proc. of the Great Lakes Symposium on VLSI*. ACM. doi:10.1145/3649476.3658737.

[20] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52, 1, 127–138. doi:10.1109/JSSC.2016.2616357.

[21] M. Damschen, L. Bauer, and J. Henkel. 2017. CoRQ: Enabling runtime reconfiguration under WCET guarantees for real-time systems. *IEEE Embedded Systems Letters*, 9, 3, 77–80.

[22] L. Kadel, H. Nassar, L. Bauer, and J. Henkel. 2025. Secure runtime reconfiguration of FPGAs via lightweight authenticated encryption for IoT systems. In *Proc. of the 10th Int'l Conference on Smart and Sustainable Technologies (SpliTech)*.

[23] K. Balaskas et al. 2024. Heterogeneous accelerator design for multi-DNN workloads via heuristic optimization. *IEEE Embedded Systems Letters*, 16, 4, 317–320.

[24] H. Khdr, M. Bakr Sikal, B. Dietrich, and J. Henkel. 2025. Towards the optimization of hardware efficiency through machine learning. In *Proc. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*.

[25] L. Steiner et al. 2020. DRAMSys4.0: A fast and cycle-accurate SystemC/TLM-based DRAM simulator. In *Proc. of the Int'l Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. Springer, 110–126.

[26] M. Sabih, B. Sesli, F. Hannig, and J. Teich. 2024. Accelerating DNNs using weight clustering on RISC-V custom functional units. In *Proc. of the Conference on Design, Automation and Test in Europe*. IEEE. doi:10.23919/DATE58400.2024.10546844.

[27] M. Sabih et al. 2024. Hardware/software co-design of RISC-V extensions for accelerating sparse DNNs on FPGAs. In *Proc. of the Int'l Conference on Field Programmable Technology (FPT)*. IEEE.

[28] P. van Kempen, M. Salmen, D. Mueller-Gritschneder, and U. Schlichtmann. 2024. Seal5: Semi-automated LLVM support for RISC-V ISA extensions including autovectorization. In *Proc. of the 27th Euromicro Conference on Digital System Design (DSD)*. IEEE, 335–342. doi:10.1109/DSD64264.2024.00052.

[29] T. Chen et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *Proc. of 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen.

[30] R. David et al. 2021. TensorFlow Lite Micro: Embedded machine learning for TinyML systems. In *Proc. of the Conference on Machine Learning and Systems (MLSys)*. mlsys.org, 800–811.

[31] J. Shao et al. 2022. Tensor program optimization with probabilistic programs. In *Proc. of the Int'l Conference on Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 35783–35796.

[32] G. Dong et al. 2019. Floating-point multiplication timing attack on deep neural network. In *Proc. of the IEEE Int'l Conference on Smart Internet of Things (SmartIoT)*. IEEE, 155–161. doi:10.1109/SmartIoT.2019.00032.

[33] C. Gongye, Y. Fei, and T. Wahl. 2020. Reverse-engineering deep neural networks using floating-point timing side-channels. In *Proc. of the 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. doi:10.1109/18072.2020.9218707.

[34] Q. Xu, M. Tanvir Arafin, and G. Qu. 2021. Security of neural networks from hardware perspective: A survey and beyond. In *Proc. of the 26th Asia and South Pacific Design Automation Conference*. ACM. doi:10.1145/3394885.3431639.

[35] M. S. Ali et al. 2025. Security risks in AI accelerators: Detecting RTL vulnerabilities to model theft with formal verification. In *Proc. of the IEEE European Test Symposium (ETS)*, 1–6. doi:10.1109/ETS63895.2025.11049644.

[36] P. P. Bernardo et al. 2020. Ultratrail: A configurable ultralow-power TC-ResNet AI accelerator for efficient keyword spotting. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 39, 11. doi:10.1109/TCAD.2020.3012320.

[37] M. Blott et al. 2018. FINN-*R*: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. on Reconfigurable Technology and Systems*, 11, 3, 16:1–16:23. doi:10.1145/3242897.

[38] FINN Team at AMD Research. 2016. FINN: dataflow compiler for QNN inference on FPGAs. Retrieved 07/20/2025 from https://xilinx.github.io/finn/.

[39] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proc. of the 23rd Int'l Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469. doi:10.1109/ICPR.2016.7900006.

[40] L. Batina, S. Bhasin, D. Jap, and S. Picek. 2019. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proc. of the 28th USENIX Security Symposium (USENIX Security)*. USENIX Association, 515–532. https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

[41] A. Dubey, R. Cammarota, and A. Aysu. 2020. MaskedNet: The first hardware inference engine aiming power side-channel protection. In *Proc. of the IEEE Int'l Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 197–208. doi:10.1109/HOST45689.2020.9300276.

[42] K. Yoshida et al. 2020. Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In *Proc. of the IEEE Int'l Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5. doi:10.1109/ISCAS45731.2020.9180580.

[43] M. Brosch, M. Probst, and G. Sigl. 2022. Counteract side-channel analysis of neural networks by shuffling. In *Proc. of the Conference on Design, Automation and Test in Europe (DATE)*, 1305–1310. doi:10.23919/DATE54114.2022.9774710.

[44] M. Brosch, M. Probst, M. Glaser, and G. Sigl. 2024. A masked hardware accelerator for feed-forward neural networks with fixed-point arithmetic. *IEEE Trans. on Very Large Scale Integration Systems*, 32, 2, 231–244. doi:10.1109/TVLSI.2023.3340553.

[45] A. Dubey, R. Cammarota, and A. Aysu. 2020. BoMaNet: Boolean masking of an entire neural network. In *Proc. of the 39th Int'l Conference on Computer-Aided Design (ICCAD)*. ACM, 51:1–51:9. doi:10.1145/3400302.3415649.

[46] S. Maji, U. Banerjee, S. H. Fuller, and A. P. Chandrakasan. 2023. A threshold implementation-based neural network accelerator with power and electromagnetic side-channel countermeasures. *IEEE Journal of Solid-State Circuits*, 58, 1, 141–154. doi:10.1109/JSSC.2022.3215670.

[47] A. Dubey et al. 2023. Hardware-software co-design for side-channel protected neural network inference. In *Proc. of the IEEE Int'l Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 155–166. doi:10.1109/HOST55118.2023.10133716.

[48] V. Hadžić and R. Bloem. 2021. COCOALMA: A versatile masking verifier. In *Proc. of the Conference on Formal Methods in Computer Aided Design (FMCAD)*, 14–23. doi:10.34727/2021/isbn.978-3-85448-046-4_9.

[49] D. Knichel, P. Sasdrich, and A. Moradi. 2020. SILVER – Statistical independence and leakage verification. In *Proc. of the 26th Int'l Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer, 787–816. doi:10.1007/978-3-030-64837-4_26.

[50] F. Zhou, H. Chen, and L. Fan. 2024. Prover – Toward more efficient formal verification of masking in probing model. Cryptology ePrint Archive, Paper 2024/1202. (2024). https://eprint.iacr.org/2024/1202.