

# Multi-Partner Project: Efficient Deep Learning Platforms for Next-Generation Embedded Edge-AI Systems

Rajendra Bishnoi<sup>1</sup> Mohammad Amin Yaldagard<sup>1</sup> Konstantinos Stavrakakis<sup>1</sup> Said Hamdioui<sup>1</sup>  
 Kanishkan Vadivel<sup>2</sup> Pankaj Upadhyay<sup>2</sup> Nicolas Daniel Rodriguez<sup>3</sup> Teresa van Dam<sup>4</sup>  
 Sander Steeghs-Turchina<sup>4</sup> Agathe Archet<sup>5</sup> Prathamesh Satish Deshpande<sup>6</sup>  
 Giovanni Grandi<sup>7</sup> Hana Krichene<sup>8</sup> William Fabre<sup>8</sup> Fabian Chersi<sup>8</sup>

<sup>1</sup> Delft University of Technology, Netherlands; <sup>2</sup> IMEC, Netherlands; <sup>3</sup> Silicon Austria Labs, Austria;

<sup>4</sup> Almende B.V, Netherlands; <sup>5</sup> cortAIx labs, Thales Research & Technology, France;

<sup>6</sup> Infineon Technologies AG, Germany; <sup>7</sup> Codasip GmbH, Germany; <sup>8</sup> Université Paris-Saclay, CEA, List, France.

**Abstract**—The objective of our collaborative multi-partner project is to create an open-source Deep Learning framework called AIDGE for edge and embedded Artificial Intelligence (AI), built around an established European value chain. The framework is designed to support diverse application domains that function independently while serving a broad international community. It offers an integrated, full-stack workflow from Neural Network design and optimization to AI application development and hardware-level implementation with automated code generation for specific hardware targets. The platform aims to provide researchers and developers with a flexible environment to explore novel AI concepts, rapidly prototype solutions, and ensure strong alignment between academic research and industrial requirements. This paper summarizes the progress, outcomes, and milestones achieved up to the second year of this three-year project.

## I. INTRODUCTION

Embedded systems are compact, constrained, and application-specific computing units that are integrated into larger systems to perform dedicated control, monitoring, and automation functions. When Artificial Intelligence (AI) capabilities are incorporated into these systems, they evolve into intelligent platforms capable of real-time data processing, predictive maintenance, and adaptive decision-making [1], [2]. Currently, AI-enabled embedded and edge systems have become the backbone of a wide range of applications, from autonomous driving, smart manufacturing, and robotics to healthcare diagnostics, wearable devices, precision agriculture, environmental monitoring, and intelligent transportation networks [3]. By processing data locally, these systems achieve rapid response times while significantly reducing dependency on remote cloud servers. This not only ensures low latency and high reliability but also enhances data privacy and security by keeping sensitive information within the device or local network [4], [5].

The NEUROKIT2E project [6] aims to bridge the gap between AI model development and the implementation of embedded hardware. It delivers an integrated suite of tools called AIDGE [7] that spans the complete workflow, from designing

and training AI models to generating optimized, standalone code tailored for resource-constrained hardware platforms. This comprehensive framework not only accelerates AI deployment at the edge but also empowers developers to efficiently translate advanced AI algorithms into practical, real-world embedded systems.

In the NEUROKIT2E project, we address the challenges associated with each abstraction layer of the project development cycle. For instance, a key challenge is to understand the application requirements and align them with the constraints and capabilities of the execution platform. Next, for Deep Neural Network (DNN) models, one of the main challenges is to find the right balance between high accuracy and maintaining computational efficiency. This is because performance improvements usually demand more resources that can consume higher energy, as well as longer training and inference times. Moreover, the challenge associated with neural network mapping on hardware is to perform an effective conversion of the complex structure of DNN models into hardware configurations that improve performance while using less energy and resources. The primary challenges in hardware modeling are to effectively simulate the behavior of hardware while ensuring the model accurately reflects its key design metrics, such as latency, energy consumption, area, and leakage. For such hardware implementations, a challenge lies in translating high-level designs into physical hardware while providing generic building blocks, methodologies, and tools that can be reused across different architecture classes, such as Neuromorphic, Tensor-based, and programmable systems. In this project, we address the aforementioned challenges and develop an open-source framework for embedded edge AI systems that can support various applications.

## II. OVERVIEW OF NEUROKIT2E PROJECT

NEUROKIT2E aims to develop, an open-source AI code generation framework optimized for both common (CPU, GPU) and embedded targets. The framework supports a diverse range of applications, including Electric Vehicles, Digital Society

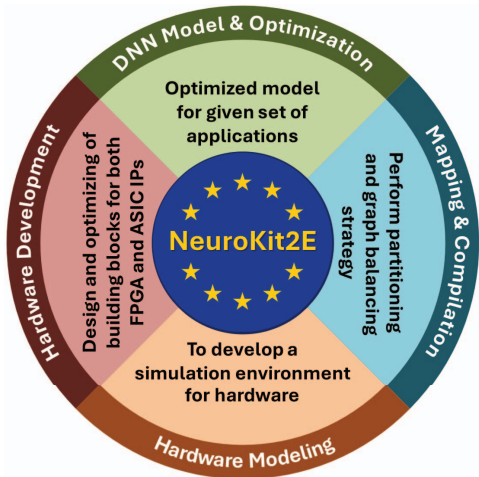


Fig. 1. Overview of the NEUROKIT2E project.

(Earth Observation and Biometrics), Mobility (Rail Transport), Health and Well-being, and Smart Buildings. As shown in Fig. 1, the project’s development workflow encompasses several key stages: DNN model design and optimization, mapping and compilation, hardware modeling, and hardware development and optimization.

The four main objectives of our project are listed below:

- Develop a Unified and Versatile Software Development Kit (SDK) and Hardware Modeling Framework.
- Develop Innovative Hardware-Aware Machine Learning Optimization Techniques.
- Address Industrial Challenges for Usability of AI.
- Enable Full-Scale HLS IP Generation.

In this paper, we present the full-stack solutions for AI embedded edge systems that we are developing as part of the NEUROKIT2E project. In Section III, we present the details of DNN modeling and optimizations, followed by descriptions of neural network mapping to hardware configurations in Section IV. We address hardware modeling in Section V, describing key aspects of the hardware simulation environment. In Section VI, we cover hardware generation and optimization, where we describe the hardware design implementations. Section VII concludes the paper.

### III. MODELING AND OPTIMIZATION

#### A. Spiking Neural Network (SNN) Optimization

Spiking Neural Networks (SNNs) are biologically plausible neural network (NN) models that use spiking neurons as their fundamental computational units [8]. SNNs are well-known for their energy-efficiency, making them suitable for edge and embedded AI applications. In the NEUROKIT2E project, we explored the optimization of SNNs for embedded systems, focusing on two main aspects: algorithmic optimizations and quantization techniques. We reproduced the results of [9] on the same hand gesture recognition (HGR) task explained in [9], using time-domain mm-wave radar data. The computational complexity of the spiking convolutional neural network (SCNN) model in [9] was improved by algorithmic optimizations, while retaining high accuracy. An SCNN model contains

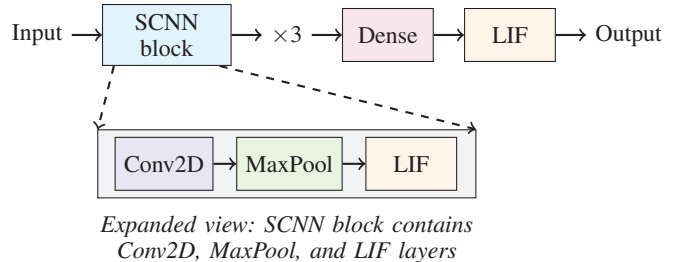


Fig. 2. Model architecture of the spiking neural network in [9]: three SCNN blocks (expanded view below), followed by a Dense layer and a final LIF layer.

both standard artificial neural network (ANN) layers (e.g., convolutional, pooling, dense) and spiking layers (e.g., leaky integrate-and-fire (LIF) neurons).

Fig. 2 shows the architecture of the SCNN model in [9] used for the HGR task. Each SCNN block consists of a convolutional layer, a max-pooling layer, and a LIF spiking neuron layer. Input is fed to three consecutive SCNN blocks, followed by a dense layer and a final LIF layer that produces the output membrane potentials for classification. This architecture was studied and reproduced with a test accuracy of 98.08%, which is close to the reported accuracy of 98.45% in [9]. To improve the computational efficiency of the model, a layer-by-layer assessment of the operations (OPs), such as multiply-accumulate (MAC) operations and other computations (COMPs), was performed to determine their impact on overall performance. Table I gives an overview of the estimated operations per layer in the baseline SCNN model in [9]. As seen from Table I, the convolutional layers dominate the computational load, accounting for the majority of MAC operations in the baseline model. Furthermore, the first convolutional layer (Conv1) has the highest number of MACs. Hence, to reduce the computational complexity of the model, we focused on optimizing the first convolutional layer. Motivated by MobileNets [10], Conv1 was replaced with a depthwise separable convolution, which decomposes a standard convolution into a depthwise convolution followed by a pointwise convolution. This decomposition reduced the number of MAC operations in Conv1 from  $\approx 1\text{M}$  to  $\approx 205\text{k}$  MACs, as shown in Table I. Quantization strategies were also applied to the optimized SCNN model to further reduce its memory footprint and computational complexity. For this purpose, Brevitas [11] was used to perform quantization-aware

TABLE I  
ESTIMATED OPERATIONS PER LAYER FOR BASELINE AND OPTIMIZED SCNN MODELS

Baseline model [9]			Optimized SCNN model		
Layer	MACs	COMPs	Layer	MACs	COMPs
Conv1	$\approx 1\text{M}$	-	Depthwise	$\approx 185\text{k}$	-
			Pointwise	$\approx 20\text{k}$	-
MaxPool1	-	$\approx 7\text{k}$	MaxPool1	-	$\approx 7\text{k}$
LIF1	$\approx 3\text{k}$	$\approx 2\text{k}$	LIF1	$\approx 3\text{k}$	$\approx 2\text{k}$
Conv2	$\approx 301\text{k}$	-	Conv2	$\approx 301\text{k}$	-
MaxPool2	-	384	MaxPool2	-	384
LIF2	192	$\approx 192$	LIF2	192	$\approx 192$
Conv3	$\approx 110\text{k}$	-	Conv3	$\approx 110\text{k}$	-
MaxPool3	-	256	MaxPool3	-	256
LIF3	128	$\approx 128$	LIF3	128	$\approx 128$
Dense	390	-	Dense	390	-
LIF4	6	$\approx 6$	LIF4	6	$\approx 6$
Total	$\approx 1.4\text{M}$	$\approx 10\text{k}$		$\approx 620\text{k}$	$\approx 10\text{k}$

TABLE II  
TEST ACCURACY OF DIFFERENT MODELS

Model	Test accuracy
Baseline model in [9]	98.08%
Optimized SCNN model	97.86%
QAT of optimized SCNN model	97.93%

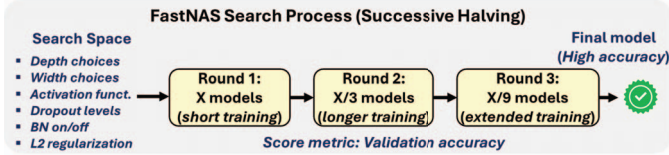


Fig. 3. Adapted FastNAS implementation for optimized MLP.

training (QAT) of the optimized model. The weights and activations of the optimized SCNN model were quantized to 8-bit integers. Table II shows the test accuracy of the baseline model in [9], the optimized SCNN model with depthwise separable convolutions, and the QAT of the optimized SCNN model. As seen from Table II, even after the algorithmic optimizations and quantization, the test accuracy of the optimized SCNN model remains high, with only a slight drop compared to the baseline model.

#### B. MLP Optimization for Embedded Device Identification

Based on the feature set and the methodology proposed by Reinhardt *et al.* [12], a compact Multi-layer Perceptron model (MLP) was developed to recognize electrical appliances based on their power usage data. A multistage optimization pipeline was implemented to ensure robustness and hardware efficiency. First, light feature engineering was performed. A Random Forest model was then trained to serve as a teacher for knowledge distillation, allowing the distilled MLP to reproduce complex nonlinear boundaries within the constraints of embedded inference.

To further refine the model architecture, FastNAS was applied to automatically explore network topologies under strict parameter and latency constraints. Fig. 3 shows our adapted implementation of FastNAS which resulted in an optimal two-layer MLP with improved accuracy–efficiency trade-offs. The original Random Forest model occupied 43 MB, whereas the distilled and optimized MLP required only 0.02 MB, representing a 99.95% size reduction. The resulting network was subsequently quantized to 8-bit integer precision using the quantization module of the AIDGE framework, enabling deployment as a standalone C++ model in an embedded device and achieving substantial memory reductions with slight performance degradation. This workflow demonstrates an effective path from data preprocessing to quantized deployment, enabling efficient device recognition performed on resource-constrained embedded platforms.

### IV. MAPPING AND COMPILATION

#### A. Multi-Core Mapping Flow

SENeCA [13] (Scalable Energy-efficient Neuromorphic Computer Architecture) is a memory-dominated, event-driven processor built from clusters of cores that combine a RISC-V controller, SIMD Neuron Processing Elements (NPEs), loop

buffers, and a NoC for fine-grained dataflow. Its event-driven nature means computation is triggered by sparse, meaningful events rather than by dense frame copies, which both enlarges the mapping design space and changes the cost metrics: peak memory provisioning is traded for run-time communication, and optimal mappings depend strongly on event sparsity and temporal locality.

The mapping space is therefore multidimensional: partition granularity (single layers vs. clustered subgraphs), node placement onto cores, NoC routing and multicast grouping, intra-core scheduling and vectorization (RISC-V versus coprocessor microcode), and temporal strategies (static tiling vs. profile-guided remapping). Each axis interacts with hardware constraints (local SRAM budgets, vector width, NPE count) and KPIs (latency, energy, memory, NoC utilization), producing an exponentially large, data-dependent search space that requires hybrid analytic, heuristic, and constrained optimization techniques.

At the **system level** our flow performs three coordinated steps: (1) partitioning — cluster tightly coupled nodes to minimize inter-core traffic while balancing compute load (Boost Graph Library); (2) placement and routing — solve a constrained MILP (Google OR-tools) augmented by heuristics to map clusters onto cores and generate NoC routes and multicast groups under topology, bandwidth and latency constraints; and (3) scheduling/codegen — employ Timeloop to explore intra-core tilings, loop orders and vectorization choices and emit template-based C and micro-code that respect local memory and co-processor capabilities. Objective functions expose Pareto tradeoffs (latency vs. energy vs. area) and support profile-guided pruning to focus search on mappings that match observed event statistics.

At the **core level**, mapping refines operation ordering, loop nest transformations, and vectorization across neuron/state and weight dimensions, exploiting the dual RISC-V/loop-buffer control to minimize data movement and maximize reuse. Finally, this multilevel mapping flow is integrated into the NEUROKIT2E / AIDGE toolchain, using hardware models in Section V to drive automated, profile-guided mapping and code generation —bridging algorithm design, architecture modeling, and deployable edge-AI implementations.

#### B. Flexible DNN Mapping Model for Dataflow Architectures

The method of mapping a deep neural network (DNN) onto a hardware architecture is crucial, as it significantly influences the energy consumption and overall latency of the system. In this work, we propose a flexible mapping model for deploying DNNs on NoC-based dataflow architectures. This model aims to optimize the rapid implementation of DNNs while improving their adaptability to different dataflow schemes [14], such as Row Stationary, Output Stationary, or Input Stationary. It simplifies the mapping process to provide an accurate estimate of key performance indicators, including performance, power consumption, and area occupancy (PPA).

The mapping model operates at an abstract level (Fig. 4), requiring only architectural dimensions (memory capacity, parallelism degree, dataflow configuration). Similarly, for the

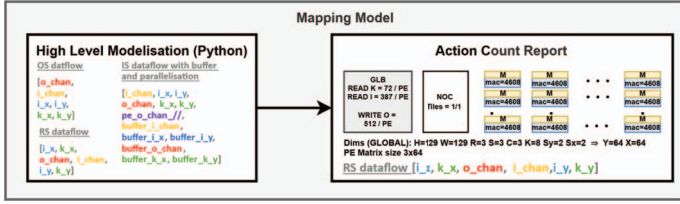


Fig. 4. Mapping model for different dataflow schemes.

DNN, only layer dimensions and types are needed, not the actual weight values. The model provides a simple interface for specifying mappings through dimension vectors and loop orderings, which naturally describe data movement patterns within the architecture. From these high-level specifications, the model generates comprehensive action count reports that quantify Multiply-Accumulate (MAC) and accumulation operations, memory access patterns, and per PE data requirements. These reports enable benchmarking and comparison of different mapping strategies to identify optimal configurations for specific layers and dataflow schemes.

The proposed generic mapping model offers great flexibility for analysis and optimization, supporting various dimensions and types of dataflows. It is independent of specific architectural implementations, DNN models, and dataflow types. This abstraction reduces the complexity associated with deep nesting and parallelism between multiple data types, thereby reducing the modeling time for large-scale applications.

This model will then be integrated into a hardware/software exploration and co-optimization tool, enabling the design of high-performance architectures that effectively balance latency, power, and area constraints for each layer of a DNN.

### C. Hardware-aware Neural Architecture Search (HW-NAS)

Manually adapting the definition of the DNN for an embedded HW target may lead to suboptimal KPIs. This is particularly true for industrial image processing AI applications relying on sophisticated DNN architectures with a huge design space. To ensure optimized alignment between the DNN architecture and the HW constraints, Hardware-aware Neural Architecture Search (HW-NAS) methods have been developed to automatically optimize DNN architectures [15].

Unlike previous works [16], we developed an HW-NAS method that, for the first time, takes HW constraints into account within an Incremental Learning context through Multi-Objective Optimization as illustrated in Fig. 5 [17]. The resulting optimized DNN architectures show better compatibility with HW constraints, e.g., 75% fewer parameters as illustrated in Table III, and achieve a 65.8% higher accuracy than a traditional genetic NAS flow [18]. Furthermore, by design, our SEENAS flow provides accuracy, stability, compatibility with multi-objective optimization, and limited-expansion update strategies. Future work will further optimize the DNN architecture using the available AIDGE framework optimizations.

## V. HARDWARE MODELING

### A. Computation-in-Memory (CIM) Accelerator Modeling

A single CIM accelerator combines a Resistive RAM (RRAM) crossbar with peripheral circuits to perform vector-

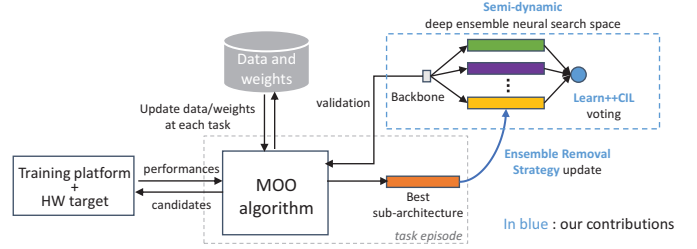


Fig. 5. Our SEENAS flow introduces a flexible updatable DNN search space for Multi-Objective Optimization (MOO) Incremental Learning HW-NAS.

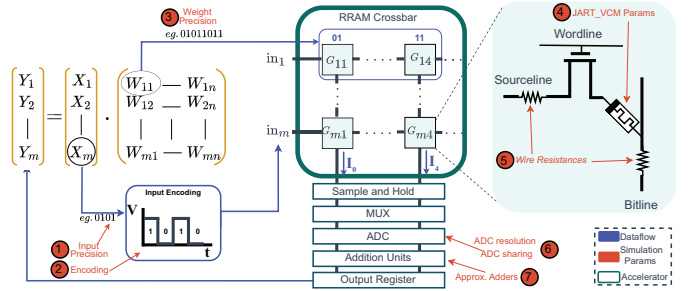


Fig. 6. Architecture of a CIM accelerator tile with an RRAM crossbar and peripheral circuits, with numbered labels indicating configurable design choices across device, circuit, and architectural levels.

matrix multiplication (VMM) within memory [19]–[25]. The architecture and dataflow are depicted in Fig. 6, where weights are stored as RRAM conductances, while digital inputs are converted to analog voltages applied across crossbar rows. Each RRAM cell conducts a current proportional to its stored weight, and summed column currents represent partial MAC results. These currents are captured by Sample-and-Hold (S&H) circuits, digitized by ADCs, and accumulated over bit-serial cycles to produce the final output [26]–[33]. Designing a CIM accelerator requires navigating a large and complex design space with many configurable parameters spanning device, circuit, and architectural levels, as illustrated by the numbered design choices in Fig. 6. Accurate modeling of analog effects such as IR drop and device variation is essential to produce valid results. Therefore, fast, modular, and accurate simulation tools are needed to explore this design space effectively.

To address these requirements, we implemented a behavioral RRAM crossbar model with full device fidelity by directly porting the complete JART physics into C++. This model is integrated with peripheral circuits using SystemC to enable accurate dataflow and cycle-accurate timing simulation. The simulator executes a custom instruction set, parsed by SystemC,

TABLE III  
OBTAINED PERFORMANCE ON CIFAR-100 AND ON A NVIDIA TITAN V GPU DEPENDING OF THE HW CRITERION TO OPTIMIZE, ALONGSIDE THE ACCURACY CRITERION.

Optimized HW criterion	$\bar{A}$	$A_f$	Params (M)	Efficiency (GFLOP/s)	Latency (ms)	Power (W)
None	64.23	51.03	14.5	997	15	67.1
# params	<b>66.27</b>	51.71	<b>3.5</b>	164	18.9	<b>54.7</b>
GFLOP/s	64.75	50.37	15.06	<b>1266</b>	17.9	66.4
Latency	65.60	49.7	13.5	687	<b>12.6</b>	61.9
Power	65.20	<b>51.94</b>	3.7	343	17.3	55.3
Best vs None improvement	+3%	+1.7%	-75%	+27%	-16%	-18.4%

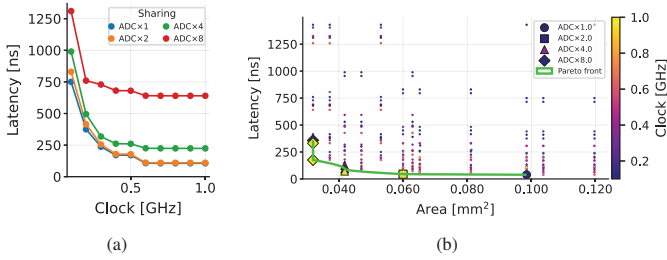


Fig. 7. Design space exploration results (a) latency vs clock frequency for different ADC sharing ratios and (b) Pareto-optimal area-latency front.

while a code-generation tool creates flexible instruction sequences for customizable dataflows. For precise latency, energy, and area estimation, NeuroSIM is integrated into the workflow. This combined approach enables efficient evaluation of diverse design parameters across device, circuit, and architecture levels. As a demonstration of the tool’s capabilities, we conducted a small-scale design space exploration across differently-sized VMM kernels on a  $128 \times 128$  crossbar. Fig. 7 shows frequency-latency trade-offs for varying ADC sharing configurations and area-latency Pareto fronts across ADC sharing and clock frequency settings.

### B. Multi-Core Dataflow Simulator

Event-driven architectures for neuromorphic computing are highly sensitive to data dynamism, where input characteristics directly drive computational patterns and resource utilization. Unlike conventional neural network accelerators with predictable dataflows, event-driven systems exhibit workload-dependent behavior that makes static mapping strategies inherently inefficient, leading to suboptimal resource utilization and unpredictable performance. This requires accurate simulation-based performance estimation to guide architectural design and optimize mapping strategies. Furthermore, integrating hardware simulation within the training pipeline enables co-optimization of both algorithmic and architectural parameters, facilitating the development of hardware-friendly models that incorporate design constraints such as event-dropout mechanisms and memory-aware scheduling policies. To address these requirements, we developed SeNSim (Scalable Neuromorphic Simulator), a fast, scalable, and highly configurable cycle-accurate simulator for multi-core event-driven processors. The simulator models a Spark architecture, where each core comprises a controller, an address generation unit, and a compute engine with configurable processing elements. These elements operate with dedicated local weight and neuron state memories while sharing access to global memory through a parameterizable Network-on-Chip interconnect.

The simulator employs a multi-stage pipeline architecture that explicitly models temporal dependencies inherent in event-driven neural network processing. As input events traverse the processing hierarchy, the system maintains dependency ordering to ensure correct execution semantics while capturing the impact of memory access patterns, computational latencies, and inter-core communication on overall performance. The framework provides comprehensive instrumentation for analyzing architectural bottlenecks, including memory subsystem uti-

lization, processing element efficiency, and network contention effects. This dependency-aware execution model enables quantitative analysis of how different scheduling strategies, particularly depth-first execution policies, affect critical metrics such as memory footprint, processing latency, and energy consumption. The simulation infrastructure generates detailed performance statistics that facilitate the architectural design space exploration and optimization of dataflow scheduling algorithms for event-driven neuromorphic computing systems, enabling rapid iteration during both algorithm development and hardware design phases.

## VI. HARDWARE GENERATION AND OPTIMIZATION

### A. Higher-Level Synthesis (HLS+)

High-Level Synthesis (HLS) is a design methodology that enables the automatic conversion of high-level algorithmic descriptions, typically written in languages such as C, C++, or SystemC, into hardware descriptions suitable for implementation on digital circuits like FPGAs or ASICs.

HLS allows engineers to describe *what* a system should do, rather than *how* it should do it (as is the case in HDLs languages), significantly improving productivity and abstraction. In modern digital system design, HLS has become an essential technology, particularly in domains such as signal processing, machine learning acceleration, and embedded systems. As hardware complexity continues to grow, HLS offers a promising path toward bridging the gap between software-level algorithm design and efficient, high-performance hardware implementation.

In this project, we explored an even higher abstraction level, i.e. HLS+, starting from a description of the DNN architecture in ONNX or MLIR (Machine Learning Intermediate Representation) format, bypassing the description in C/C++. This allowed us to easily synthesize RTL code without the extensive manual interventions required to provide information at the structural and parametric level. As a result, we are now able to synthesize MobileNet1 for CEA’s NeuroCorgi [34] - a fixed-weight, dataflow architecture - starting from a very high-level description, typically in ONNX or JSON using the CorgiBuilder of the AIDGE framework.

### B. CIM Optimization on Read-disturb

RRAM enables CIM architectures that mitigate data-transfer and energy bottlenecks in von Neumann systems by executing multiply-and-accumulate (MAC) operations within memory arrays, offering substantial throughput and power efficiency for edge AI applications [35]–[40]. However, repeated reads induce read-disturb effects that gradually shift cell conductance, accumulating drift that degrades computational precision and long-term reliability in analog MAC operations [41], [42]. Traditional mitigation scheme employs fixed, periodic reprogramming intervals to restore device conductance levels; however, this approach demands extensive read-cycle tracking using large hardware counters and often triggers unnecessary refresh operations, leading to considerable energy and area overhead. To overcome these issues, a real-time, circuit-level read-disturb detection methodology was introduced in [43], which dynamically



## REFERENCES

- [1] V. Mazzia *et al.*, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," *Ieee Access*, vol. 8, pp. 9102–9114, 2020.
- [2] V. Yazdanpanah *et al.*, "Reasoning about responsibility in autonomous systems: challenges and opportunities," *AI & SOCIETY*, vol. 38, no. 4, pp. 1453–1464, 2023.
- [3] C. Badue *et al.*, "Self-driving cars: A survey," *Expert systems with applications*, vol. 165, p. 113816, 2021.
- [4] H. Hua *et al.*, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.
- [5] L. Huijbregts *et al.*, "ESAM: Energy-efficient SNN Architecture using 3nm FinFET Multiport SRAM-based CIM with Online Learning," in *Design Automation Conference (DAC)*, 2024, pp. 1–6.
- [6] R. Bishnoi *et al.*, "Multi-Partner Project: A Deep Learning Platform Targeting Embedded Hardware for Edge-AI Applications (NEUROKIT2E)," in *DATE*, 2025, pp. 1–7.
- [7] F. Chersi *et al.*, "AIDGE: A Framework for Deep Neural Network Development, Training and Deployment on the Edge," in *European Conference on Edge AI 2023*, 2025, pp. 41–53.
- [8] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.
- [9] A. Shaaban *et al.*, "RT-SCNNs: real-time spiking convolutional neural networks for a novel hand gesture recognition using time-domain mm-wave radar data," *International Journal of Microwave and Wireless Technologies*, vol. 16, no. 5, p. 783–795, 2024.
- [10] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: <http://arxiv.org/abs/1704.04861>
- [11] A. Pappalardo, "Xilinx/brevitas," 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.3333552>
- [12] A. Reinhardt *et al.*, "On the Accuracy of Appliance Identification Based on Distributed Load Metering Data," *Procedia Computer Science*, vol. 19, pp. 668–677, 2013.
- [13] G. Tang *et al.*, "SENECA: building a fully digital neuromorphic processor, design trade-offs and challenges," *Frontiers in Neuroscience*, vol. Volume 17 - 2023, 2023.
- [14] R. Li, "Dataflow & Tiling Strategies in Edge-AI FPGA Accelerators: A Comprehensive Literature Review," 2025. [Online]. Available: <https://arxiv.org/abs/2505.08992>
- [15] H. Benmeziane *et al.*, "A comprehensive survey on hardware-aware neural architecture search," *arXiv preprint arXiv:2101.09336*, 2021.
- [16] M. Shahawy *et al.*, "Exploring the intersection between neural architecture search and continual learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.
- [17] D. V. Ferreira *et al.*, "Semi-Dynamic Ensemble for Evolutionary NAS in Continual Learning," *IEEE International Conference on Pattern Recognition Systems*, 2025.
- [18] X. Du *et al.*, "Evolutionary NAS in light of model stability for accurate continual learning," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.
- [19] R. Bishnoi *et al.*, "Energy-efficient computation-in-memory architecture using emerging technologies," in *IEEE ICM*, 2023, pp. 325–334.
- [20] S. Diware *et al.*, "Severity-based hierarchical ecg classification using neural networks," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 17, no. 1, pp. 77–91, 2023.
- [21] S. Diware *et al.*, "Mapping-aware biased training for accurate memristor-based neural networks," in *IEEE AICAS*, 2023, pp. 1–5.
- [22] S. Diware *et al.*, "Accurate and energy-efficient bit-slicing for rram-based neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 7, no. 1, pp. 164–177, 2022.
- [23] S. Diware *et al.*, "Reliable and energy-efficient diabetic retinopathy screening using memristor-based neural networks," *IEEE Access*, vol. 12, pp. 47 469–47 482, 2024.
- [24] A. Singh *et al.*, "Low-power memristor-based computing for edge-ai applications," in *ISCAS*, 2021, pp. 1–5.
- [25] S. Diware *et al.*, "Adaptive multi-threshold encoding for energy-efficient ecg classification architecture using spiking neural network," in *DATE*, 2025, pp. 1–7.
- [26] A. Singh *et al.*, "A 115.1 TOPS/W, 12.1 TOPS/Mm<sup>2</sup> Computation-in-Memory Using Ring-Oscillator Based ADC for Edge AI," in *AICAS*, 2023, pp. 1–5.
- [27] A. Singh *et al.*, "SRIF: Scalable and Reliable Integrate and Fire Circuit ADC for Memristor-Based CIM Architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1917–1930, 2021.
- [28] A. Singh *et al.*, "Accelerating RRAM Testing with a Low-cost Computation-in-Memory based DFT," in *International Test Conference (ITC)*, 2022, pp. 400–409.
- [29] A. Singh *et al.*, "Energy-efficient Multi-Operand XOR Logic-based CIM Accelerator using RRAM technology," in *ICCAD*, 2025.
- [30] M. Fieback *et al.*, "Defects, fault modeling, and test development framework for RRAMs," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–26, 2022.
- [31] A. Singh *et al.*, "Referencing-in-Array Scheme for RRAM-based CIM Architecture," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1413–1418.
- [32] Y. Biyani *et al.*, "SABCIM: Self-Adaptive Biasing Scheme for Accurate and Efficient Analog Compute-in-Memory," in *IEEE ASP-DAC*, 2026, pp. 1–7.
- [33] Y. Biyani *et al.*, "C3CIM: Constant column current memristor-based computation-in-memory micro-architecture," in *DATE*, 2025, pp. 1–7.
- [34] I. Miro-Panades *et al.*, "A 772 $\mu$ J/frame ImageNet Feature Extractor Accelerator on HD Images at 30FPS," in *2024 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2024.
- [35] S. Yu *et al.*, "Compute-in-Memory Chips for Deep Learning: Recent Trends and Prospects," *IEEE Circuits and Systems Magazine*, vol. 21, no. 3, pp. 31–56, 2021.
- [36] S. Diware *et al.*, "Unbalanced Bit-slicing Scheme for Accurate Memristor-based Neural Network Architecture," in *International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.
- [37] A. Sehgal *et al.*, "Continuous On-Chip Learning in Neural Networks using SOT-MRAM based CIM Architectures," in *ICCAD*, 2025, pp. 1–8.
- [38] A. Sehgal *et al.*, "Enhancing Parallelism and Energy-Efficiency in SOT-MRAM based CIM Architecture for On-Chip Learning," in *DAC*, 2025, pp. 1–7.
- [39] E. Hua *et al.*, "Neural Network Architectures for Scalable Quantum State Tomography: Benchmarking and Memristor-Based Acceleration," *arXiv preprint arXiv:2507.23007*, 2025.
- [40] M. Mayahinia *et al.*, "A voltage-controlled, oscillation-based adc design for computation-in-memory architectures using emerging rram," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 2, pp. 1–25, 2022.
- [41] S. Diware *et al.*, "Dynamic Detection and Mitigation of Read-disturb for Accurate Memristor-based Neural Networks," in *IEEE International Conference on AI Circuits and Systems (AICAS)*, 2024, pp. 393–397.
- [42] S. Diware *et al.*, "Hardware-Aware Quantization for Accurate Memristor-Based Neural Networks," in *International Conference on Computer Aided Design (ICCAD)*, 2024.
- [43] M. A. Yaldagard *et al.*, "Read-disturb Detection Methodology for RRAM-based Computation-in-Memory Architecture," in *International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2023, pp. 1–5.
- [44] N. D. Rodriguez, "Circuitos integrados de bajo consumo para arquitecturas de redes neuronales profundas," PhD thesis, Universidad Nacional del Sur, Bahia Blanca, Argentina, 2024, available at <https://repositoriodigital.uns.edu.ar/handle/123456789/7328>.