CIMNAS: A Joint Framework for Compute-In-Memory-Aware Neural Architecture Search

Olga Krestinskaya, Mohammed E. Fouda, Ahmed Eltawil, and Khaled N. Salama

***Abstract*—To maximize hardware efficiency and performance accuracy in Compute-In-Memory (CIM)-based neural network accelerators for Artificial Intelligence (AI) applications, co-optimizing both software and hardware design parameters is essential. Manual tuning is impractical due to the vast number of parameters and their complex interdependencies. To effectively automate the design and optimization of CIM-based neural network accelerators, hardware-aware neural architecture search (HW-NAS) techniques can be applied. This work introduces CIMNAS, a joint model-quantization-hardware optimization framework for CIM architectures. CIMNAS simultaneously searches across software parameters, quantization policies, and a broad range of hardware parameters, incorporating device-, circuit-, and architecture-level co-optimizations. CIMNAS experiments were conducted over a search space of 9.9×10^{85} potential parameter combinations with the MobileNet model as a baseline and RRAM-based CIM architecture. Evaluated on the ImageNet dataset, CIMNAS achieved a reduction in energy-delay-area product (EDAP) ranging from $90.1\times$ to $104.5\times$, an improvement in TOPS/W between $4.68\times$ and $4.82\times$, and an enhancement in TOPS/mm^2 from $11.3\times$ to $12.78\times$ relative to various baselines, all while maintaining an accuracy of 73.81%. The adaptability and robustness of CIMNAS are demonstrated by extending the framework to support the SRAM-based ResNet50 architecture, achieving up to an $819.5\times$ reduction in EDAP. Unlike other state-of-the-art methods, CIMNAS achieves EDAP-focused optimization without any accuracy loss, generating diverse software-hardware parameter combinations for high-performance CIM-based neural network designs. The source code of CIMNAS is available at https://github.com/OlgaKrestinskaya/CIMNAS.**

***Index Terms*—Hardware-aware Neural Architecture Search, In-memory Computing, Software-Hardware Co-design**

I. Introduction

The exponential growth of Artificial Intelligence (AI) applications and increasing AI model complexity are raising the energy demands for training and processing AI workloads [1]. This trend has created a demand for more sustainable and energy-efficient hardware solutions for AI applications. Compute-In-Memory (CIM) neural network accelerators have emerged as promising architectures for achieving energy-efficient AI processing [2]–[6]. To maximize the hardware efficiency of CIM accelerators and maintain high performance for neural network workloads, it is essential to co-optimize both neural network model parameters and CIM hardware parameters [7]. Furthermore, achieving optimal efficiency in CIM hardware requires a holistic approach that considers all levels of hardware design, including device-, circuit-, and architecture-level optimizations [8]. With a design space reaching 10^{85} possible parameter combinations, manually optimizing such an extensive design space is infeasible. An effective approach to address this co-optimization challenge is Hardware-Aware Neural Architecture Search (HW-NAS) [9]–[11].

Initially, neural architecture search (NAS) techniques were developed for purely software-based neural network models to automate the search for optimal parameter combinations [12]. Later, these techniques incorporated hardware feedback [10], [13] and were adapted for various hardware types, including CIM architectures [9], [14]. Current state-of-the-art HW-NAS frameworks for CIM primarily focus on optimizing neural network models for hardware implementation [15]–[17], performing hardware design space exploration separately [9], [18], [19], or jointly optimizing model parameters with a limited set of hardware parameters [20], [21]. However, to achieve both high model accuracy and efficient hardware performance in CIM design, it is essential to co-optimize a broad range of parameters, especially due to their complex interdependencies.

In this work, we introduce CIMNAS, a Compute-In-Memory-Aware Neural Architecture Search framework that performs joint co-optimization of model parameters, quantization policies, and CIM hardware parameters, with a focus on optimizing a wide range of hardware parameters to enhance efficiency without sacrificing performance accuracy. Unlike traditional approaches, CIMNAS targets a comprehensive range of hardware-level parameters, including device, circuit, and architectural levels, to maximize hardware efficiency without compromising model accuracy. CIMNAS explores an extensive search space of size 9.9×10^{85} of possible combinations of parameters, jointly optimizing neural network model parameters (e.g. number of layers, kernel sizes, and per-layer expansion factors), quantization policies (e.g. layer-wise precision for weights and activations for depth-wise and point-wise convolutions), and CIM hardware configurations (e.g. bits per cell in CIM device, operation voltage, cycle time, CIM crossbar sizes, number of CIM macros per tile, number of tiles sharing a single router, number of tile groups per chip and global buffer size). By integrating all these dimensions into a joint search process, CIMNAS avoids sub-optimal solutions caused by independent or sequential tuning and mitigates the risk of local minima. In contrast to state-of-the-art frameworks, CIMNAS is uniquely tailored for the full CIM design hierarchy, enabling the discovery of EDAP-optimized (Energy–Delay–Area–Product) hardware configu-

Olga Krestinskaya, Ahmed Eltawil and Khaled N. Salama are with King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia. Emails:{ok@ieee.org, ahmed.eltawil@kaust.edu.sa, khaled.salama@kaust.edu.sa}. Mohammed Fouda is with Compumacy for Artificial Intelligence Solutions, Cairo, Egypt. Email: foudam@uci.edu.

This work was supported by the King Abdullah University of Science and Technology through the Competitive Research Grant program under grant URF/1/4704-01-01.

TABLE I
STATE-OF-THE-ART HW-NAS APPROACHES AND COMPARISON WITH CIMNAS.

Framework	Optimization (search space)					Algorithm	Search space size	SW-HW co-opt.	Approach	Backbone network	Baseline EDAP^{*5} / Optimized EDAP	Accuracy drop (EDAP - optimized)^{*6}
	SW/ Model	Q^{*1}	Hardware									
			D^{*2}	C^{*3}	A^{*4}							
AnalogNAS [15]	✓	-	-	-	-	EA	7.3×10^{10}	-	HW feedback	ResNet32	-	-
NAS4RRAM [16]	✓	-	-	-	-	EA	4.7×10^{5}	-	HW feedback	ResNet20, ResNet32	-	-
Flash [22]	✓	-	-	-	-	SHGO	6.4×10^{10}	-	HW feedback	DenseNets	-	-
NACIM [17]	✓	✓	-	-	-	RL	2.6×10^{25}	-	HW feedback	VGG11	3.9	↓ 11.0%
CoMN^{*7} [18]	-	-	✓	✓	✓	BO	2.2×10^{4}	-	HW feedback, DSE, two-stage	CNNs^{*8}	-	-
Joint HWC^{*9} [23]	-	-	✓	✓	✓	EA	1.9×10^{7}	-	DSE	CNNs^{*8}	-	-
NAX [20]	✓	-	-	✓	-	DS	2.4×10^{11}	✓	Joint	ResNet20	1.1-5.9^{*10}	↓ 1.1-11.1%^{*10}
CIMNet [24]	✓	✓	✓	✓	-	EA	1.0×10^{12}	✓	Joint	EfficientNet	-	↑ 0.1-0.2%^{*12}
Gibbon [25]	✓	✓	✓	✓	-	EA	4.3×10^{84}	✓	Joint	ResNet18	51.1-59.1^{*10}	↓ 6.3-6.7%^{*10}
XPert [21]	✓	✓^{*11}	-	✓	-	DS	7.1×10^{34}	✓	Two-stage	VGG16	10.4	↓ 1.2-1.3%
CIMNAS (this work)	✓	✓	✓	✓	✓	EA	**9.9×10^{85}**	✓	Joint	MobileNet	**73.0-92.7**^{*13} **82.4-107.5**^{*14}	↑ **0.6%**^{*15}- **0.8%**^{*16}
							1.02×10^{16}			ResNet50	**191.8-251.1**^{*13} **622.9-819.5**^{*14}	↓ **0.6%**^{*15}- **0.7%**^{*16}

SW - software, HW - hardware, DSE - design space exploration, co-opt. - co-optimization. ^{*1}: Quantization optimization: optimum weights and input precision search. ^{*2}: Device optimization - including device parameters search, e.g. bits per cell. ^{*3}: Circuit optimization - including circuit-level parameters optimization of crossbar macros, e.g. array size or ADC precision.^{*4}: Architecture optimization - including higher-level architecture parameters optimization, e.g. tiles, global buffer, etc. ^{*5}: higher = better, ^{*6}: ↓ - accuracy decreased, ↑ - accuracy increased, ^{*7}: Search space size is based on demonstrated HW search space, ^{*8}: CNNs: ResNet, VGG, AlexNet, MobileNet, ^{*9} Hardware-workload co-exploration (HWC), ^{*10}: varies depending on the dataset, ^{*11}: input precision only, ^{*12}: energy-latency optimized (area not considered), ^{*13}/^{*14}: comparing to Baseline 1 / Baseline 2 for 5 best EDAP-optimized designs (Section IV), ^{*15}/^{*16}: top1/top-5 performance accuracies of EDAP-optimized designs versus accuracies of baseline architectures (Section IV)

rations while maintaining competitive accuracy. Moreover, CIMNAS is robust, adaptable, and offers a high diversity among top-optimized designs, acknowledging the potential for multiple optimal parameter configurations within such a vast search space.

This paper is structured as follows. Section II provides background on Compute-In-Memory (CIM) architectures and reviews related work in neural architecture search and hardware-aware optimization. Section III presents the proposed CIMNAS framework, describing its joint search space formulation, optimization methodology, hardware configuration, neural network model used in the simulations, and the evaluation metrics for both software and hardware. Section IV details the experimental setup, including search configurations, baseline models, and comparison methods, and provides an in-depth analysis of the results, highlighting CIMNAS's performance in terms of accuracy, efficiency, and design diversity. Section V emphasizes the importance of jointly optimizing model architecture, quantization policies, and hardware parameters, and outlines how CIMNAS can be adapted for future co-design applications and emerging CIM technologies. Finally, Section VI concludes the paper and outlines potential directions for future research.

II. RELATED WORK

State-of-the-art HW-NAS methods for CIM-based architectures can be categorized into three main approaches: HW-NAS for fixed CIM architectures, CIM-based design space exploration for a fixed optimized neural network model, and HW-NAS for co-optimization of software-hardware parameters in CIM-based neural network designs [9]. In HW-NAS for fixed CIM architectures, neural network model parameters are optimized with consideration for CIM hardware (HW) feedback [14]–[17], [22], [26], [27]. This approach adapts the neural network model to fit the fixed hardware, helping to mitigate the effects of hardware non-idealities in CIM devices [28] Similarly, several frameworks optimize quantization policies to enhance the performance of CIM-based architectures [29]–[31]. In contrast, CIM-based design space exploration focuses on identifying the optimal CIM hardware parameters for deploying a fixed neural network model [18], [19], [23]. Finally, HW-NAS methods that co-optimize software and hardware parameters focus on jointly optimizing both the neural network model and CIM hardware parameters [20], [21], [24], [25]. This approach is particularly valuable in the initial stages of IMC-based AI accelerator design, where an optimized model and hardware setup are required for specific applications.

Table I presents a comparison between the most relevant state-of-the-art CIM-based HW-NAS approaches and the proposed CIMNAS framework, evaluating them in terms of optimized parameters, algorithms, search space size, optimization approach, and achieved optimization results. The energy-delay-area product (EDAP) metric is used to compare co-optimization frameworks. Due to variations in datasets and baseline CIM hardware across different frameworks, a direct quantitative comparison with the proposed approach is not feasible. To address this, we use two comparison metrics: (1) the ratio of the EDAP of the baseline design to that of the optimized design, $\frac{Baseline\,EDAP}{Optimized\,EDAP}$ [21], and (2) the accuracy drop observed in the EDAP-optimized design. Baseline EDAP refers to the EDAP of the reference design used as the starting point in the framework. Optimized EDAP denotes the EDAP of the best-performing architecture obtained after optimizing hardware parameters. Accuracy drop indicates the reduction in performance accuracy of the optimized design relative to the baseline, when optimized to reduce EDAP. These metrics are presented in Table I to provide a fair and consistent evaluation. Accuracy drop is commonly observed in software-hardware co-optimization, where accuracy is often traded off to achieve greater hardware efficiency.

AnalogNAS [15], NAS4RRAM [16], Flash [22], and NACIM [17] frameworks focus on HW-NAS for optimizing neural network models with hardware feedback. Both AnalogNAS and NAS4RRAM optimize ResNet-like models

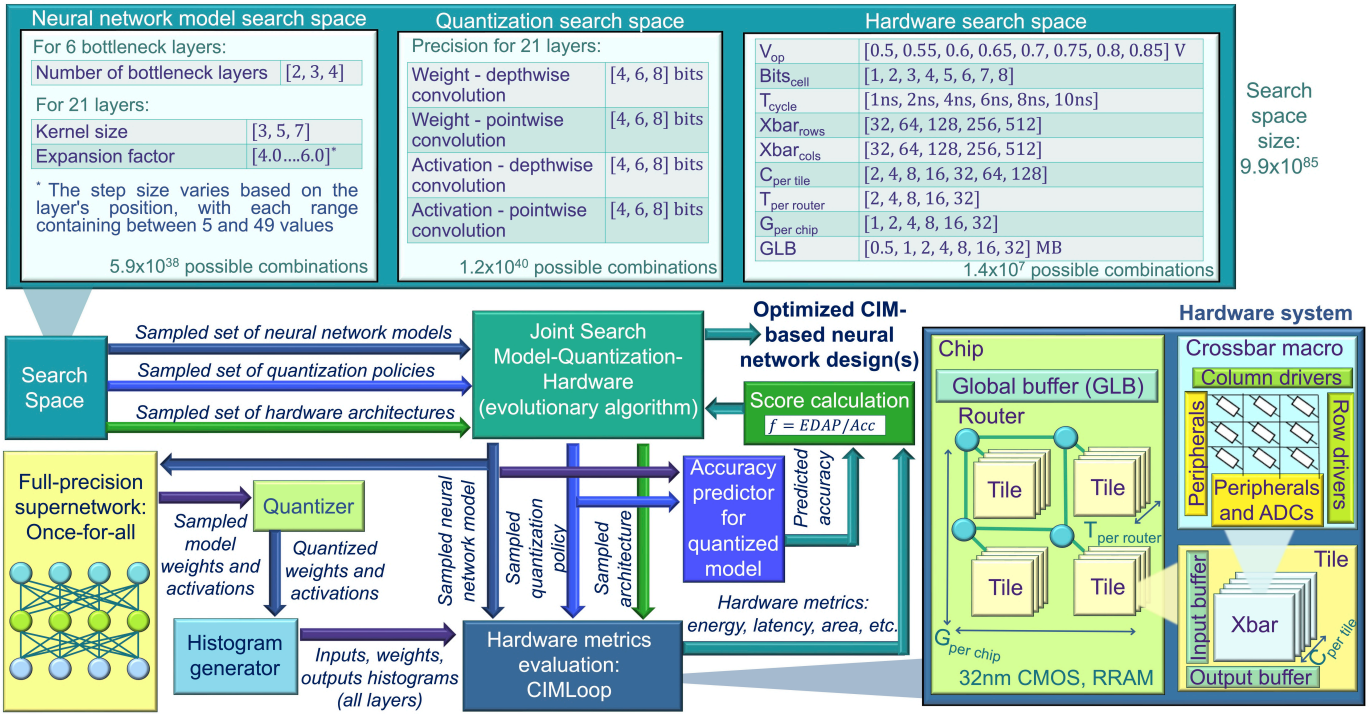

Fig. 1. CIMNAS: Joint co-optimization of software, precision, and hardware parameters for CIM-based neural network design.

using evolutionary algorithms (EA). Flash employs a coarse- and fine-grained search via a simplicial homology global optimization (SHGO)-based algorithm, utilizing an accuracy predictor and neural network degree metrics [22]. NACIM also optimizes model precision alongside parameters using reinforcement learning (RL) [17]. These approaches optimize neural network models for deployment on existing CIM-based hardware. In contrast, CoMN [18] and the Joint hardware-workload co-optimization (HWC) framework [23] focus on hardware design space exploration (DSE) for fixed neural network models. CoMN employs Bayesian optimization (BO) to search a large hardware space, optimizing mapping and architecture parameters for system performance. The HWC framework performs joint optimization across hardware workloads to create a generalized hardware solution optimized for diverse tasks. However, separately optimizing hardware parameters for high-accuracy models may result in suboptimal designs, as software-optimized models often lead to underutilized CIM-based hardware in deployment [25].

Achieving truly optimal CIM chip design for neural network applications, particularly for low-power edge devices, requires co-optimizing software and hardware parameters. Frameworks like NAX [20], CIMNet [24], Gibbon [25], and XPert [21]. illustrate such co-optimization approaches. NAX optimizes kernel and crossbar sizes for energy efficiency using differential search (DS), but focuses only on limited crossbar array trade-offs, which is insufficient for full CIM architecture optimization. CIMNet jointly optimizes model architecture, quantization levels, and hardware parameters by leveraging an accuracy predictor and a layer-wise look-up table (LUT)-based hardware metrics estimator, assuming a constrained hardware search space limited to device precision and crossbar size. While this approach significantly reduces search time, the reliance on layer-wise LUT estimators becomes impractical as the hardware search space expands, particularly when architectural and system-level parameters are included. These parameters often depend on inter-layer data transmission and mapping strategies, making LUT-based estimation inefficient and less scalable. Gibbon co-optimizes a large search space including software precision and CIM hardware parameters, yet restricts exploration to crossbar macro settings, such as crossbar and converter precision, leading to a notable accuracy drop due to modifications in ResNet for better CIM compatibility. To reduce the search space from 4.3×10^{84} to 1.3×10^{22}, Gibbon's algorithm prunes lower-priority parameters. Applying a similar strategy to the search space in this work could result in missed optimal configurations due to strong interdependencies among parameters such as circuit settings, layer count, and precision. XPert employs a two-stage optimization, first optimizing channel depth, ADC type, and column sharing to reduce latency and area, then adjusting input and ADC precision for energy and accuracy. Although effective for initial parameters, this approach may get trapped in local minima, limiting the diversity of designs. Both Gibbon and XPert vary hardware parameters across network layers, which, while enhancing efficiency, complicates fabrication and limits the chip's reusability for other tasks due to inconsistencies like differing crossbar and ADC designs.

In this work, we address these issues by jointly co-optimizing software model parameters (e.g. number of layers, kernel sizes, expansion factor for each layer), quantization settings (e.g. precisions for weights and activations for depth-wise and point-wise convolution in each layer), and hardware parameters, including device-, circuit-, and architecture-level

configurations, within a vast search space of 9.9×10^{85} possible combinations of parameters. We explore diverse hardware parameters while maintaining consistency across the architecture to facilitate easier transistor-level design, layout, and CIM chip fabrication based on these optimized configurations. We jointly co-optimize these parameters within a single search to avoid local minima, fully explore the search space, and ensure diversity among the optimized designs generated by the framework. CIMNAS preserves the high performance accuracy of CIM-based neural networks while achieving optimized EDAP, avoiding a drop in accuracy relative to the baseline design.

III. CIMNAS

The proposed CIMNAS framework for joint software-quantization-hardware co-exploration in CIM-based neural network optimization is illustrated in Fig. 1. The CIMNAS search space is built upon the MobileNetV2 baseline architecture [32] for an ImageNet-based [33] image classification task. The MobileNet backbone was used as it is often overlooked in in-memory computing research and poses challenges due to its depthwise separable convolutions and compact design [34]. Despite these complexities, our results demonstrate that our algorithm performs effectively, highlighting its robustness and adaptability. To avoid local minima and ensure comprehensive search space coverage, CIMNAS jointly optimizes all parameters in the search space. The framework samples sets of neural network parameters, quantization policies, and hardware architectures, feeding them into an evolutionary joint search algorithm. In each iteration, the algorithm evaluates each neural network implementation based on performance accuracy and hardware metrics. An accuracy predictor, trained on quantized neural network models, is used to quickly estimate performance accuracy, as quantizing and retraining for each combination is impractical. Hardware metrics are evaluated using sampled hardware parameters and layer-specific histograms of quantized inputs, weights, and outputs. These histograms are generated by sampling a candidate neural network from a pre-trained full-precision supernetwork and applying the corresponding sampled quantization policy (Section III-C). CIMNAS outputs a set of optimized CIM-based mixed-precision neural network models and hardware parameters. The algorithm and overall workflow of CIMNAS are presented in Algorithm 1.

Algorithm 1 CIMNAS algorithm (in this work $P = 150$, $G = 70 - 100$).

Initial population sampling :
while (population $p < P$) **do**
  Randomly **sample** neural network model from S_M
  Randomly **sample** quantization policy from S_Q
  Randomly **sample** candidate hardware c_H from S_H
  Find M memory elements required for the sampled quantized model
  if Number of memory elements of $c_h \geq M$ **then**
    Keep the sample in the initial population
    $p = p + 1$
for g **in** G generations **do**
  Evaluation phase :
  if $g > 1$ **then**
    Exclude samples, where hardware design does not fit corresponding sampled model and quantization
  for Each sample α in a population **do**
    Obtain full precision model weights and activations from the supernetwork
    Quantize weights and activation values
    Generate corresponding histograms
    Evaluate hardware metrics
    Obtain accuracy for quantized model from accuracy predictor
    Calculate score f_α
  Selection, *crossover*, *and mutation* :
  Sort the designs
  Select designs to participate in crossover (crossover probability $\mathbb{P}_c$)
  Perform crossover with distribution η_c, constructing new $''$offsprings$''$
  Execute mutation with probability $\mathbb{P}_m$ and distribition η_m, constructing new population with P samples

A. Combined model-quantization-hardware search space

The complete search space is illustrated in Fig. 1. It consists of the neural network model (S_M), quantization (S_Q), and hardware parameters S_H search spaces. The neural network model search space includes 5.9×10^{38} possible configurations, adjusting parameters such as the number of bottleneck layer blocks (depth/repetition) in the MobileNetV2 architecture, kernel sizes for depthwise convolutions, and expansion factors across six bottleneck layers, while keeping the first convolution, first bottleneck, last convolution, and final linear classification layers fixed. The quantization search space comprises 1.2×10^{40} possible combinations, covering weight and input precision for depthwise and pointwise convolutions across all bottleneck layers.

The CIM hardware search space is based on a hierarchical resistive random access memory (RRAM)-based CIM architecture, as shown in Fig. 1. This hierarchy includes crossbar macros, CIM tiles, tile groups, and the interconnections between them. The CIM chip is composed of crossbar macros with peripheral circuits, row/column drivers, and converters. Each tile consists of multiple crossbar macros ($\mathrm{C_{per\,tile}}$) with $\mathrm{Xbar_{rows}}$ rows and $\mathrm{Xbar_{cols}}$ columns, along with input/output buffers. The chip contains a global SRAM-based buffer (GLB) for storing input and output data, with groups of tiles connected by routers. The CIM chip architecture includes $\mathrm{G_{per\,chip}}$ tile groups, each with $\mathrm{T_{per\,router}}$ tiles and $\mathrm{G_{per\,chip}}$ routers. All of the parameters mentioned above are incorporated into the search space. Additional parameters like device precision ($\mathrm{Bits_{cell}}$), operating voltage ($\mathrm{V_{op}}$), and cycle time ($\mathrm{T_{cycle}}$) are also considered, resulting in a hardware search space with 1.4×10^7 possible parameter combinations.

In total, the jointly optimized search space includes 9.9×10^{85} possible configurations. The relationships between

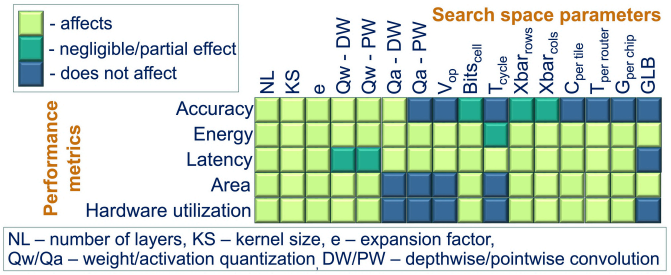

Fig. 2. Effect of search space parameters on hardware performance and accuracy and correlation between them.

search space parameters and their impact on performance metrics are illustrated in Fig. 2, showing strong interdependencies among software, quantization, and hardware parameters. For instance, the sizes of crossbar macros, the number of macros, and the number of tiles are interrelated, allowing for multiple optimal design configurations; reducing one parameter may require increasing another to ensure compatibility between the model and hardware. Additionally, crossbar size and the number of bits per cell can impact performance accuracy in the presence of device non-idealities and noise variations; however, this topic is beyond the scope of this work. Overall, most parameters influence all performance metrics, making joint optimization essential to avoid local minima. Furthermore, in Section IV, we show that two-stage approaches tend to fall into local minima within a given search space, resulting in designs that prioritize either high accuracy or low EDAP, but not both.

B. Search algorithm and optimization function

1) Search algorithm selection: CIMNAS is based on a genetic algorithm, a specific type of evolutionary algorithm (EA). Common NAS and HW-NAS algorithms include EA, differential search (DS), reinforcement learning (RL), and Bayesian optimization (BO). Table II compares EA with RL, BO, and DS, justifying the choice of EA for the large and complex CIMNAS search space.

RL is computationally expensive and scales poorly with hyperparameters due to its sequential controller updates. RL-based NAS frameworks NASNet and MnasNet require 22,400 GPU-hours and 288 TPU days, respectively [35], [36]. RL typically involves 4,000–20,000 episodes per architecture and struggles with sparse rewards in large spaces, leading to suboptimal convergence. BO is also sequential, where each iteration involves architecture generation, evaluation, and surrogate model updates. It often requires hundreds of surrogate samples (e.g., 283.2 GPU-hours in the BANANAS framework [37]), but performance degrades in high-dimensional search spaces due to the breakdown of surrogate modeling. Compared to RL- and BO-based methods, EA is faster and more practical for large search spaces [9], [23].

DS, as in the DARTS framework, trains a supernetwork over a few days [38], but relies on differentiable parameters. When extended to discrete hardware parameters, model duplication leads to exponential memory demands, and relaxation techniques fail to scale. DS is unsuitable for joint software-hardware searches, as the supernetwork of software models alone requires at least 48GB of GPU memory per model for training and fine-tuning. Simultaneously co-optimizing software and hardware using DS and expanding this supernetwork to include all possible hardware combinations would require an excessive amount of GPU resources for training and convergence [9], [39]–[41].

EA, in contrast, scales linearly with generations and supports parallel evaluation of populations. It is derivative-free, handles discrete parameters without relaxation, and achieves good coverage via crossover and mutation. Typical configurations use 30–100 generations with populations of 50–150. EA's diversity helps escape local minima and converge efficiently, even in large search spaces [9], [39]–[41]. Therefore, EA is selected for CIMNAS due to its scalability, parallelism, and compatibility with discrete hardware-aware search. Additionally, we use a pre-trained supernetwork-based accuracy predictor, similar to DS, to efficiently estimate candidate performance.

2) CIMNAS algorithm: Preliminary experiments showed that unconstrained optimization often results in CIM designs with infeasibly large on-chip areas. To address this, CIMNAS focuses on area-constrained optimization, which serves two main purposes: it excludes designs with excessively large areas that are impractical to fabricate as a single chip, and it drives the evolutionary algorithm (EA) to converge more quickly toward reasonably sized CIM hardware compared to unconstrained optimization. The network score of a given CIM-based neural network design, α, is calculated using an objective function, f, that incorporates energy E_α, delay D_α (latency across all layers), on-chip area A_α, and predicted performance accuracy Acc_α for the sampled design:

$$f_\alpha = f(E_\alpha, D_\alpha, A_\alpha, Acc_\alpha) \quad \text{s.t. } A_\alpha \leq A_{constr} \tag{1}$$

where A_{constr} represents the area constraint. We focus on EDAP optimization by minimizing the objective function $f_\alpha = \frac{E_\alpha \times D_\alpha \times A_\alpha}{Acc_\alpha}$. Accuracy is included directly in the objective function to ensure that EDAP optimization does not compromise performance accuracy. We avoid setting accuracy as a constraint because the search is based on a pre-trained supernetwork, which inherently prevents the sampling of low-accuracy designs, eliminating the need to filter out designs with insufficient accuracy. The most compact supernetwork configuration—comprising 2 bottleneck blocks per bottleneck layer, a kernel size of 3, and an expansion factor of 4 in all layers—achieves 64.9% accuracy on the ImageNet dataset when using the lowest bit precision of 4 bits for both weights and inputs. Additionally, we incorporate on-chip area as both an objective and a constraint to ensure that the algorithm not only minimizes area but also avoids converging to designs with unreasonably large on-chip areas, which often tend to reduce energy consumption. In Section IV, we present experiments with other area-constrained objective functions for comparison.

TABLE II
COMPARISON OF THE STATE-OF-THE-ART NAS ALGORITHMS AND ADVANTAGES OF EA.

Algorithm	**RL**	**BO**	**DS**	**EA**
Computational complexity	$O(a^n \cdot t \cdot C)$	$O(k \cdot C + M(k,n))$	$O(D^{n_d+1} \cdot C)$	$O(g \cdot p \cdot C)$
Scaling with hyperparameters	Exponential	Exponential (surrogate degrades with high n)	Linear-moderate (memory overhead increases with n)	**Linear per generation** (robust to large n)
Parallelization	Poor (sequential episodes)	Moderate (if batch BO used)	Poor (single graph)	**Good** (population-level parallelism)
GPU memory requirement	High (train agents and models)	**Moderate** (depends on M(k,n) and batches)	Very high (to store and train supernet)	**Moderate**
Applicability to large discrete search space	Hard to scale	Not applicable (surrogate models breaks down)	Not effective for discrete search space	**Applicable**

n – number of hyperparameters; a – number of actions per reinforcement learning (RL) step; t – trajectory length (i.e., the number of decisions made per architecture, approximately equal to n); C – cost of evaluating one candidate architecture; k – number of candidates to evaluate; $M(k,n)$ – complexity of the surrogate model as a function of k and n; g – number of generations in evolutionary algorithms (EA); p – population size in EA; n_d – number of discrete hardware parameters (e.g., hardware-specific parameters that require architecture duplication).

During the initial sampling to construct the starting population for the EA, design samples that cannot accommodate the neural network model are removed (Algorithm 1). As a result, the initial population consists only of feasible designs, while any samples that do not meet the initial design constraints are discarded. This approach reduces the presence of infeasible designs in the search space and minimizes the likelihood of EA mutations generating and propagating unfit designs throughout the search process. In addition, if an infeasible design sample is produced after crossover and mutation, it is discarded from the population in the new generation.

After initializing the population (Algorithm 1), the evolutionary algorithm is executed over G generations. For each sampled design, the full precision weights and activations from the supernetwork are quantized to generate the corresponding histograms. Hardware metrics are then evaluated, and design accuracy is estimated using an accuracy predictor. The design samples are subsequently ranked based on the objective function defined in Eq. 1. Selected designs undergo binary crossover and polynomial mutation [42], [43], with crossover and mutation probabilities set to $\mathbb{P}_c = 0.95$ and $\mathbb{P}_m = 0.95$, respectively. The distribution indices for these operations are $\eta_c = 3$ and $\eta_m = 3$, values within the typical range of 3 to 30. These parameters promote exploration and maintain population diversity, enabling a broad search of the design space. Following mutation, the evaluation process is repeated for the newly generated population. After each generation, all design samples and their associated metrics are stored. The final optimized designs are selected from the entire set of stored samples based on the objective function in Eq. 1. This approach not only ensures comprehensive optimization but also avoids redundant evaluations of identical samples, thereby reducing computational overhead and search time.

C. Evaluation of software and hardware metrics

1) Performance accuracy evaluation and accuracy predictor: For software metrics evaluation, we use the pretrained full-precision "Once-for-all" supernetwork from [44] and an accuracy predictor for quantized designs from [45]. This accuracy predictor is trained by sampling models from the supernetwork, quantizing them, and performing a single epoch of quantization-aware fine-tuning. The accuracy predictor is a three-layer neural network with hidden layers of size 400. First, a full-precision accuracy predictor is trained. The input to this predictor consists of all neural network model parameters from the search space illustrated in Fig. 1, encoded using a one-hot encoding scheme. The output is the predicted performance accuracy corresponding to the given model parameters. The full-precision predictor is trained using 80,000 samples of neural network configurations and their corresponding performance accuracies. Once the full-precision predictor is trained, it is extended to support quantization parameters from the quantization search space, which are also encoded using one-hot encoding. This quantized accuracy predictor is derived by expanding the input layer of the full-precision model to include the quantization policy representations and fine-tuning this predictor using 2,500 sampled neural network architectures, each evaluated with 10 different quantization policy configurations. The quantized accuracy predictor outputs the predicted performance accuracy of a quantized neural network, considering both the sampled model, and quantization parameters [45].

The accuracy predictor greatly reduces search time; while it takes just 5 GPU seconds per network, directly fine-tuning each quantized design would require at least 4-5 GPU hours. Without the accuracy predictor, exploring such an extensive search space would be impractical. Although the accuracy predictor does not provide exact accuracy values, its relative predictions are consistent enough to compare sampled designs during the search. Final accuracy for the top-selected networks in Section IV is obtained through fine-tuning these models.

2) Hardware metrics evaluation: Hardware metrics evaluation is performed using the CiMLoop simulator [46], a highly flexible, cycle-accurate simulation framework designed for architecture-level evaluation of CIM-based hardware accelerators. CiMLoop supports detailed modeling of architecture, circuit, and device parameters, making it well suited for evaluating diverse CIM-based design configurations. It integrates the Timeloop simulator [47] for model-to-hardware mapping and employs the Accelergy framework [48] for energy estimation. CiMLoop was selected for its relatively high simulation speed and accuracy close to that of NeuroSim [49], while

offering enhanced flexibility. Specifically, CiMLoop achieves an average error of approximately 3% in hardware estimations [46]. In contrast to NeuroSim, CiMLoop supports parallel processing of model layers, significantly reducing runtime. For instance, while NeuroSim requires approximately 945 seconds to process a MobileNetV2 architecture on a single core, CiMLoop completes the same task in 220.5 seconds. When parallelized across 64 CPU cores, the runtime is further reduced to approximately 25 seconds. Therefore, we achieve almost 40× faster hardware metrics estimation compared to NeuroSim. This parallelism is especially advantageous in our CIMNAS framework, where evolutionary algorithm (EA) search generations and CiMLoop-based hardware evaluations are both parallelizable. This enables highly efficient large-scale search, fully utilizing multithreading capabilities. For a single EA generation, we achieve at least a 10× speedup compared to using NeuroSim. The combination of fast simulation speed and low estimation error makes CiMLoop a practical and scalable choice for evaluating thousands of candidate architectures in CIMNAS.

Instead of directly processing the inputs and weights of each layer, as done in time-consuming cycle-accurate simulators that simulate every operation, CiMLoop leverages histograms of these values. This approach significantly reduces simulation time while maintaining reliable, data-dependent performance estimates. As "Once-for-all" is a pre-trained full-precision supernetwork, we quantize the inputs, weights, and outputs of each layer and convert them into corresponding histograms for CiMLoop to ensure performance estimation based on realistic data. The histogram generation function was implemented as described in [46]. Histograms for the inputs and outputs of each layer are generated using a randomly selected image from the ImageNet dataset. Each design evaluation takes about 30 seconds on a 64-core CPU, allowing the search to proceed efficiently without additional speed-up methods, such as performance predictors used in [25].

D. Computational complexity, memory cost, and runtime of CIMNAS

The computational complexity of CIMNAS is determined by two main components: the EA-based search and the supernetwork-based accuracy predictor training. The complexity of the EA search is outlined in Table II, and is primarily governed by the number of generations and the population size. Importantly, EA scales linearly with the number of hyperparameters per generation and does not suffer from exponential growth. Within each generation, candidate evaluations can be fully parallelized. Moreover, the EA search has low computational overhead and does not require GPU resources, making it suitable for parallel execution across multiple CPU cores. GPU resources are mainly utilized to accelerate the accuracy evaluations performed by the supernetwork-based predictor. The training of the accuracy predictor, based on a supernetwork, is more computationally demanding. It requires substantial GPU memory and compute resources for both initial training and fine-tuning to support quantization (as described in Section III-C1). However, once trained, this predictor can be reused across multiple hardware configurations, enabling efficient evaluation of diverse hardware–software co-design scenarios without retraining.

IV. Results

A. Simulation setup

CIMNAS searches were conducted with a population size of $P = 150$ architectures over $G = 70$ generations. Each search takes approximately 1.75 to 2.5 days, utilizing 64 CPU cores for parallel layer-wise hardware performance evaluation and a single GPU for the accuracy predictor. The initial sampling phase requires 8–10 minutes, while each generation takes around 50 minutes to complete. Hardware evaluations were performed using 32nm CMOS technology and RRAM devices from [50], with an area constraint of $A_{constr} = 800$ mm^2, reflecting a reasonable die size for single-chip fabrication [51]. To ensure fair comparison across search techniques, we fixed the initial seed to start each search with a similar initial population. Accuracy was evaluated on the ImageNet dataset with 1000 classes. Since accuracy evaluation during the search relies on the accuracy predictor, the final accuracy for each quantized architecture is obtained by fine-tuning the quantized model.

In this work, we define two primary baselines for comparison, differing in hardware parameters. In both baseline architectures, the software model chosen for reference is a standard-size 8-bit MobileNetV2 [32] neural network. It should be noted that the search space includes configurations with larger kernel sizes and additional layers, allowing NAS to potentially discover networks with higher performance accuracy than the baseline. Baseline 1 is constructed by sampling the median value of each hardware parameter, with specifications: $\mathrm{V_{op}} = 0.7V$, $\mathrm{Bits_{cell}} = 4$, $\mathrm{T_{cycle}} = 4ns$, $\mathrm{Xbar_{rows}} = \mathrm{Xbar_{cols}} = 256$, $\mathrm{C_{per\,tile}} = 16$, $\mathrm{T_{per\,router}} = 8$, $\mathrm{G_{per\,chip}} = 16$, and $\mathrm{GLB} = 4$ MB. To ensure unbiased representation, a second baseline is defined by randomly sampling 1000 hardware configurations from the search space and calculating the mean performance metrics, providing a fair representation of the search space.

We test three optimization cases with CIMNAS: (1) EDAP and accuracy, (2) delay and accuracy, and (3) energy, area, and accuracy. The corresponding objective functions for the CIMNAS search are $f = \frac{E \times D \times A}{Acc}$, $f = \frac{D}{Acc}$, and $f = \frac{E \times A}{Acc}$. We compare CIMNAS with two methods: two-stage search and XPert-like search. The two-stage search approach sequentially optimizes software and precision parameters for high accuracy without hardware considerations, followed by CIM hardware optimization in the second stage [9], this approach is similar to the one in [18]. The XPert-like search method, based on [21], also uses a two-stage optimization, with the first stage focusing on latency and area, and the second stage targeting energy and accuracy. For EDAP and accuracy optimization, the first stage focuses on model and hardware parameters to optimize area and delay, followed by quantization parameter optimization in the second stage for accuracy and energy efficiency. For delay and accuracy, latency is optimized in the first stage, with accuracy refinement in the second. For energy, area, and accuracy, area is optimized initially, followed by energy and accuracy in the second stage.

TABLE III

COMPARISON OF CIMNAS WITH BASELINE AND STATE-OF-THE-ART SEARCH METHODS. THE PROPOSED FRAMEWORK IS EVALUATED AGAINST A TWO-STAGE APPROACH AND AN XPERT-STYLE SEARCH, FOLLOWING STRATEGIES FROM [18] AND [21].

Optimization approach	Accuracy			Energy (mJ)	Delay (us)	Area (mm^2)	EDAP (mJ*ms*mm^2)	Search score	TOPS/W	TOPS/mm^2	Hardware utilization	Design diversity*2
	Top-1	Top-5	Change*1									
Baseline 1 (median of the parameters)	73.00%	91.20%	-	0.95	6.94	3691	24.33	EDAP/Acc: 0.34 L/Acc: 95.1 EA/Acc: 48.02	1.36	0.71	0.26	-
Baseline 2 (random 1000)	73.00%	91.20%	-	1.15	3.80	14033	28.19*3	EDAP/Acc: 0.38 L/Acc: 52.1 EA/Acc: 237.19	1.4	0.8	0.35	-
EDAP - focused optimization: search score EDAP/Acc (mJ*ms*mm^2/%)												
Two-stage search	74.25%	92.01%	↑ 1.25%	0.43	8.80	465	1.76	0.0250	8.41	2.42	0.53	medium
XPert-like search	69.60%	88.90%	↓ 3.4%	0.27	2.72	235	0.17	0.0025	6.56	8.91	0.68	low
CIMNAS	73.81%	91.80%	↑ 0.81%	0.33	3.55	234	0.27	0.0037	6.56	9.08	0.60	high
Latency - focused optimization: search score D/Acc (ms/%)												
Two-stage search	74.25%	92.01%	↑ 1.25%	0.39	2.10	797	0.65	28.3	9.41	9.82	0.53	medium
XPert-like search	72.76%	90.99%	↓ 0.24%	0.27	1.40	464	0.18	19.2	7.67	13.55	0.59	low
CIMNAS	73.71%	91.74%	↑ 0.71%	0.30	1.47	467	0.20	19.9	6.45	11.00	0.57	high
Energy/area - focused optimization: search score EA/Acc (mJ*mm^2/%)												
Two-stage search	74.25%	92.01%	↑ 1.25%	0.56	2.14	472	0.57	3.56	5.92	13.30	0.31	medium
XPert-like search	69.57%	89.21%	↓ 3.43%	0.20	1.21	799	0.19	3.44	8.79	8.48	0.30	low
CIMNAS	71.85%	90.58%	↓ 1.5%	0.22	2.90	234	0.14	0.71	6.75	1.63	0.52	high

*1: accuracy change compared to the baseline top-1, ↓ - accuracy decreased, ↑ - accuracy increased. *2: In the context of a large search space with several potential optima, design diversity reflects the degree of uniqueness among the top five optimized designs. *3: average across 1000 randomly sampled hardware configurations

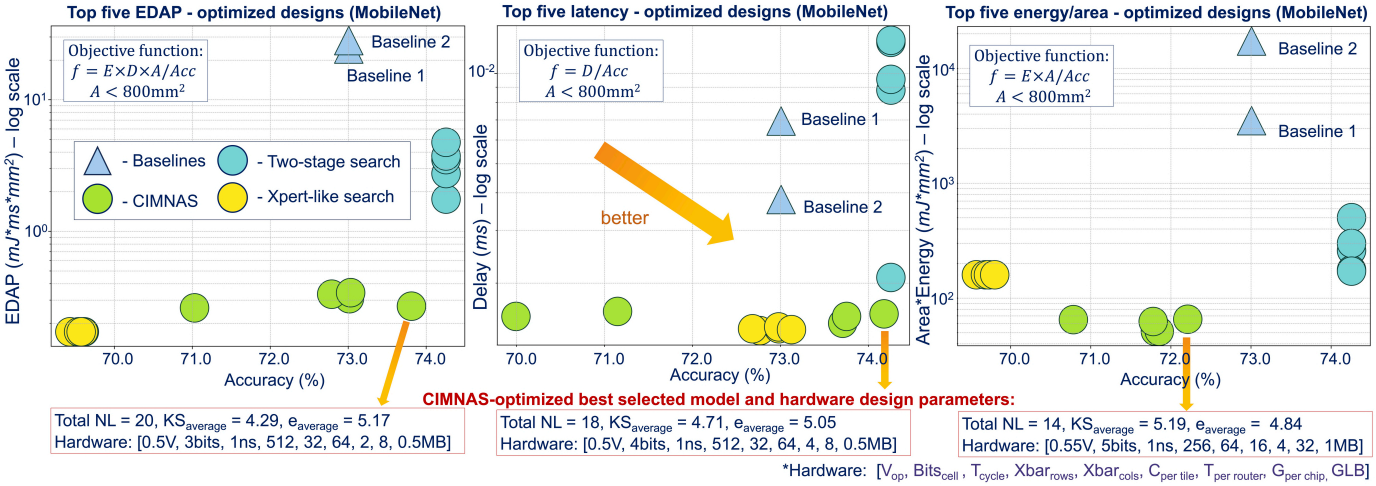

Fig. 3. Trade-offs between hardware performance and accuracy for the top 5 selected designs, along with the parameters of the highest-scoring CIMNAS-selected designs for each experiment.

B. CIMNAS simulation results

Table III and Fig. 3 show the simulation results and a comparison of the proposed methods with baseline and state-of-the-art approaches. For EDAP optimization, CIMNAS achieves a reduction of $90.1\times$ to $104.5\times$ in the energy-delay-area product, along with a $4.68\times$ to $4.82\times$ increase in energy efficiency, an $11.35\times$ to $12.78\times$ improvement in area efficiency, a $1.7\times$ to $2.3\times$ boost in hardware utilization, and a 0.81% gain in performance accuracy compared to the baseline architectures. With EDAP optimization, CIMNAS identifies architectures that reduce hardware metrics without compromising performance accuracy. Latency-focused optimization follows a similar trend, reducing delay by a factor of $2.5\times$ to $4.7\times$ compared to the baseline while increasing performance accuracy by 0.71%. For energy-area optimization, CIMNAS achieves a $68\times$ to $313\times$ improvement in the $E \times A$ score, with only a 1.5% accuracy reduction—significantly lower than the 3.43% drop seen with XPert-like search.

In contrast, the two-stage search achieves the highest accuracy by optimizing software parameters first to maximize performance; however, it fails to reduce EDAP and optimize hardware efficiency. The XPert-like search achieves superior hardware performance by minimizing latency and/or area in the first stage, but it cannot attain high accuracy afterward due to hardware restrictions imposed in the initial stage. Consequently, CIMNAS achieves the best balance between accuracy and hardware metrics. This trend is evident in Fig. 3, which shows the trade-offs between hardware metrics and performance accuracy for the top five selected designs from each of the three approaches. The same trends are observed in the other two experiments focused on latency and on energy/area optimization.

Given the large search space used in these experiments, multiple optimal parameter combinations with similar scores are possible. To assess this, we evaluate the design diversity of each method, where design diversity indicates how well a search algorithm captures various high-scoring combinations. Fig. 3 shows how close the top five designs are to

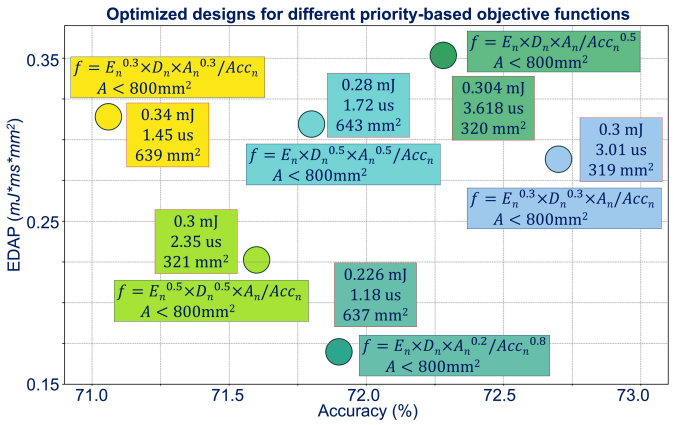

Fig. 4. Demonstration of priority-based optimization.

each other for each approach. The XPert-like search yields the least diverse architectures, as it optimizes model and hardware parameters in the first stage, and the subsequent quantization policy optimization in the second stage does not generate a wide variety of designs. The two-stage approach exhibits slightly more diversity since hardware parameters are adjusted in the second stage. Both the two-stage and XPert-like approaches tend to fall into local minima and fail to achieve the best balance between accuracy and hardware performance. Capturing a diverse set of optimal designs within the search space is important for HW-NAS, as it benefits later design stages, such as transistor-level simulations, layout development, and the fabrication of the final CIM chip.

C. Priority-based optimization

For hardware design, it is crucial to evaluate trade-offs carefully, as one hardware metric can be more critical than another when optimizing system performance. Therefore, we also tested the proposed framework for the priority-based objective function shown in Fig. 4. In priority-based optimization, the objective function $f = \frac{E_n^a \times D_n^b \times A_n^c}{Acc_n^d}$ uses the hardware metrics E_n, D_n, A_n, and accuracy Acc_n normalized to the values of the first obtained sample, where a, b, c, and d are priority coefficients in the range of 0 to 1. Depending on the optimization priorities, hardware designs with similar EDAP can exhibit distinct performance in terms of hardware metrics; for example, on-chip area and accuracy-focused optimization with objective function $f = \frac{E_n^{0.3} \times D_n^{0.3} \times A_n}{Acc_n}$ favors designs with reduced area and enhanced accuracy, while energy and delay-focused optimization with objective function $f = \frac{E_n \times D_n \times A_n^{0.2}}{Acc_n^{0.8}}$ targets designs with minimized energy consumption and delay. In addition, we demonstrate that the search is also sensitive to the objective coefficients. For example, in an area and accuracy-focused optimization setting, $a = b = 0.3$ for energy and delay priorities leads to designs with enhanced accuracy and smaller on-chip area compared to when $a = b = 0.5$. This highlights the importance of fine-tuning priority coefficients to achieve optimal hardware design outcomes based on specific performance goals.

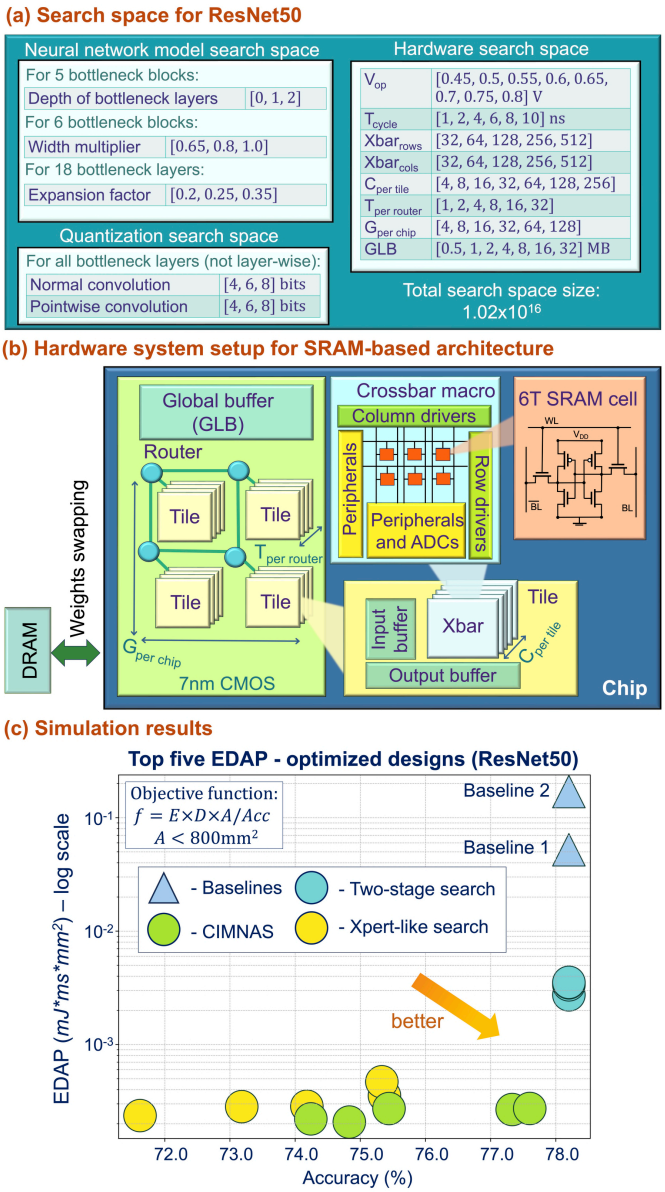

Fig. 5. (a) Search space and (b) hardware system configuration of SRAM-based ResNet50 architecture. (c) Performance demonstration of the proposed CIMNAS framework applied to the ResNet50 model within the SRAM-based CIM design space.

D. Adaptability and robustness of the framework

To demonstrate the adaptability of the CIMNAS framework and the robustness of its results beyond MobileNet and the RRAM-based weight-stationary CIM design, we extend our evaluation to a different neural network architecture, a new CIM hardware configuration, and a different CMOS technology node. Specifically, we apply CIMNAS to the ResNet50 architecture using an SRAM-based CIM design with weight swapping, evaluated at the 7nm CMOS technology node.

In this experiment, a new model search space, illustrated in Fig. 5(a), is based on the ResNet50 architecture. The model search space includes the depth of the bottleneck blocks, which defines how many times each block is repeated. It also includes a width multiplier that scales the number of

channels across the network and an expansion factor that controls the ratio between the number of channels in the intermediate (expanded) layer versus the input/output channels in each bottleneck block. The quantization search space includes precision settings for both normal and pointwise convolutions within the bottleneck layers, while the input convolution and output linear layers are fixed to 8-bit precision. The full-precision accuracy predictor is derived from a supernetwork trained on all parameter configurations within this ResNet50-based search space [44]. The quantized accuracy predictor is then obtained by fine-tuning the full-precision accuracy predictor for the quantization search space shown in Fig.5(a).

The SRAM-based hardware configuration is shown in Fig. 5(b). Unlike the RRAM-based weight-stationary architecture used in Fig. 1, which requires all neural network weights to fit on-chip, the SRAM-based design in this experiment allows weight swapping. This enables sequential processing of layer groups on the same hardware: initial layers are executed first, followed by loading the next set of weights from DRAM, and so on until the final layer. The simulation includes the energy and latency overheads associated with transferring weights from DRAM to the chip. A standard 6T SRAM cell is used as in [52]. The hardware search space is similar to the one in previous RRAM-based experiments, but slightly extended to support the larger architecture (Fig.5(a)).

The performance of CIMNAS on this ResNet50 search space using a 7nm SRAM-based CIM system is shown in Fig.5(c). Two baseline architectures are defined following the same methodology as in Section IV-A, both being conventional 8-bit ResNet50 models. For the hardware parameters, Baseline 1 represents a median configuration with $\mathrm{V_{op}} = 0.7\mathrm{V}$, $\mathrm{T_{cycle}} = 4\mathrm{ns}$, $\mathrm{Xbar_{rows}} = \mathrm{Xbar_{cols}} = 128$, $\mathrm{C_{per\,tile}} = 32$, $\mathrm{T_{per\,router}} = 8$, $\mathrm{G_{per\,chip}} = 32$, and $\mathrm{GLB} = 4\mathrm{MB}$. Baseline 2 reflects the mean hardware performance across 1,000 randomly sampled configurations from the search space.

Fig.5(c) compares CIMNAS with the other state-of-the-art frameworks. The 8-bit ResNet50 baseline achieves a Top-1 accuracy of 78.2% and a Top-5 accuracy of 91.8%. The best architecture discovered by CIMNAS reaches 77.6% Top-1 accuracy and 91.2% Top-5 accuracy, with only a 0.6% drop, an insignificant degradation, while achieving significant EDAP improvements. Specifically, CIMNAS achieves 251.1× and 819.5× reductions in EDAP compared to Baseline 1 and Baseline 2, respectively. The results in Fig.5(c) show that CIMNAS outperforms two-stage methods and Xpert-like approaches in this new setting. While the accuracy remains close to the baseline, the substantial gains in EDAP are due to the fact that the search space is dominated by hardware parameters, which have a greater impact on the hardware performance and EDAP optimization outcome, compared to the model or quantization parameters.

This experiment confirms that CIMNAS is highly adaptable to different neural network models, hardware configurations, and technology nodes, while maintaining strong performance. It demonstrates CIMNAS's ability to generalize and consistently outperform both baselines and state-of-the-art alternatives.

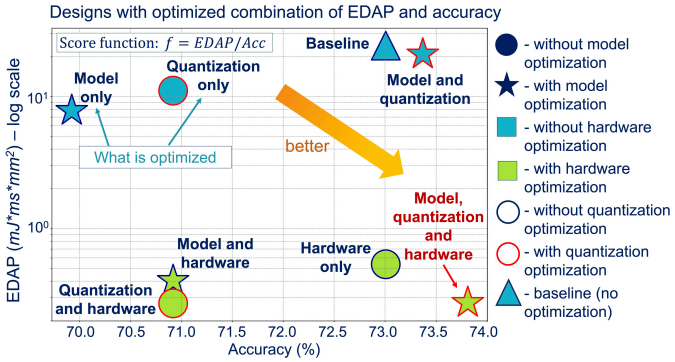

Fig. 6. Impact of parameter co-optimization on hardware efficiency and performance accuracy. Designs are generated using CIMNAS with an EDAP/Accuracy objective function (minimizing EDAP while maximizing accuracy) for MobileNet search space. The details of the baseline comparison are discussed in Section IV.

V. Discussion

The results presented in this work highlight the critical importance of performing joint optimization across the model, quantization, and CIM hardware design spaces. In addition, Fig.6 highlights the importance of an integrated model-quantization-hardware co-optimization to achieve optimal hardware efficiency and performance accuracy. The graph represents seven different CIMNAS-based optimization experiments, each aimed at identifying CIM-based neural network implementations with an optimal combination of energy-delay-area product (EDAP) and performance accuracy. These experiments involved various parameter configurations, demonstrating that optimizing hardware parameters plays a significant role in enhancing hardware performance. The results indicate that focusing solely on neural network model optimization, even with hardware feedback, is insufficient to achieve a hardware-efficient solution. To obtain the optimal combination of EDAP and performance accuracy, it is essential to co-optimize model, quantization, and hardware parameters.

Moreover, traditional sequential or isolated optimization approaches often lead to suboptimal trade-offs, as decisions made in one domain (e.g., model architecture) can significantly impact the effectiveness and efficiency of others (e.g., hardware implementation). By contrast, CIMNAS enables a holistic co-design process, capturing complex interdependencies between the algorithmic and hardware parameters to find globally optimal configurations. This joint optimization is especially critical for CIM systems, where quantization settings, memory cell behavior, circuit-level constraints, architecture-level design choices, and model performance are tightly interdependent and governed by complex non-linear relationships. CIMNAS's ability to explore an extremely large and diverse search space (on the order of 9.9×10^{85} configurations) while still identifying EDAP-efficient and accurate solutions demonstrates its scalability and robustness.

CIMNAS serves as an initial tool in CIM hardware design automation, supporting the creation of energy- and area-efficient CIM chips for AI applications. As illustrated in Fig.6 and Fig.4, the framework also supports flexible optimization modes, including the ability to optimize specific groups of

parameters independently or to perform priority-based optimization, where certain design objectives are prioritized over others based on application needs. Section IV-D demonstrates the adaptability of CIMNAS to various neural network architectures, CIM hardware configurations, and CMOS technology nodes. The framework can support diverse design settings, including both weight-stationary and weight-swapping memory schemes, without requiring fundamental modifications. Moreover, CIMNAS shows strong robustness, maintaining effective optimization performance across search spaces dominated by different factors, whether hardware, model, or quantization parameters, highlighting its generalizability across different design scenarios. Looking ahead, CIMNAS can serve as a general-purpose framework for the automated co-design of energy-efficient edge AI systems. It can be extended to support other emerging memory technologies (e.g., FeFETs, RRAM), new quantization schemes (e.g., mixed-precision, non-uniform quantization), larger neural network models, or task-specific model constraints (e.g., latency-bound or memory-limited designs). By bridging the gap between algorithm design and hardware realization, CIMNAS lays the foundation for next-generation CIM-aware neural architecture search frameworks that are adaptable, scalable, and efficient.

VI. Conclusion

We introduced CIMNAS, a CIM-aware NAS framework that jointly optimizes model, quantization, and hardware parameters across a comprehensive search space, including device-, circuit-, and architecture-level CIM hardware configurations. We introduced CIMNAS, a CIM-aware NAS framework that jointly optimizes model, quantization, and hardware parameters across a comprehensive search space, including device-, circuit-, and architecture-level CIM hardware configurations. CIMNAS achieves an optimal balance of hardware efficiency and performance without sacrificing accuracy, producing a diverse set of model-quantization-hardware parameter combinations. For RRAM-based MobileNet architecture, CIMNAS achieves up to a 104.5× reduction in EDAP, and improvements of 4.82× and 12.78× in energy and area efficiency, respectively, compared to the selected baseline. While for SRAM-based ResNet50 architecture, CIMNAS achieves an even greater EDAP reduction of up to 819.5×. As part of future work, we aim to extend the algorithm to support a broader range of workloads and tasks beyond image classification. Additionally, we plan to develop prediction models to adapt search results to different hardware technologies, eliminating the need to rerun the search when migrating to new hardware and a new technology node.

References

[1] A. Mehonic, D. Ielmini, K. Roy, O. Mutlu, S. Kvatinsky, T. Serrano-Gotarredona, B. Linares-Barranco, S. Spiga, S. Savel'ev, A. G. Balanov, *et al.*, "Roadmap to neuromorphic computing with emerging technologies," *APL Materials*, vol. 12, no. 10, 2024.

[2] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[3] O. Krestinskaya, L. Zhang, and K. N. Salama, "Towards efficient in-memory computing hardware for quantized neural networks: State-of-the-art, open challenges and perspectives," *IEEE Transactions on Nanotechnology*, vol. 22, pp. 377–386, 2023.

[4] D. Ielmini and G. Pedretti, "Device and circuit architectures for in-memory computing," *Advanced Intelligent Systems*, vol. 2, no. 7, p. 2000040, 2020.

[5] H. E. Yantır, A. M. Eltawil, and K. N. Salama, "A hardware/software co-design methodology for in-memory processors," *Journal of Parallel and Distributed Computing*, vol. 161, pp. 63–71, 2022.

[6] K. Smagulova, M. E. Fouda, F. Kurdahi, K. N. Salama, and A. Eltawil, "Resistive neural hardware accelerators," *Proceedings of the IEEE*, vol. 111, no. 5, pp. 500–527, 2023.

[7] F. Aguirre, A. Sebastian, M. Le Gallo, W. Song, T. Wang, J. J. Yang, W. Lu, M.-F. Chang, D. Ielmini, Y. Yang, *et al.*, "Hardware implementation of memristor-based artificial neural networks," *Nature communications*, vol. 15, no. 1, p. 1974, 2024.

[8] W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M.-F. Chang, H.-J. Yoo, H. Qian, and H. Wu, "Neuro-inspired computing chips," *Nature electronics*, vol. 3, no. 7, pp. 371–382, 2020.

[9] O. Krestinskaya, M. E. Fouda, H. Benmeziane, K. El Maghraoui, A. Sebastian, W. D. Lu, M. Lanza, H. Li, F. Kurdahi, S. A. Fahmy, A. Eltawil, and K. N. Salama, "Neural architecture search for in-memory computing-based deep learning accelerators," *Nature Reviews Electrical Engineering*, pp. 1–17, 2024.

[10] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–36, 2022.

[11] M. Rakka, M. E. Fouda, P. Khargonekar, and F. Kurdahi, "A review of state-of-the-art mixed-precision neural network frameworks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[12] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[13] Y. Xu, H. Shi, and Z. Wang, "Nash: Neural architecture and accelerator search for multiplication-reduced hybrid models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[14] Z. Guan, W. Zhou, Y. Ren, R. Xie, H. Yu, and N. Wong, "A hardware-aware neural architecture search pareto front exploration for in-memory computing," in *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pp. 1–4, IEEE, 2022.

[15] H. Benmeziane, C. Lammie, I. Boybat, M. Rasch, M. Le Gallo, H. Tsai, R. Muralidhar, S. Niar, O. Hamza, V. Narayanan, *et al.*, "Analognas: A neural network design framework for accurate inference with analog in-memory computing," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 233–244, IEEE, 2023.

[16] Z. Yuan, J. Liu, X. Li, L. Yan, H. Chen, B. Wu, Y. Yang, and G. Sun, "Nas4rram: neural network architecture search for inference on rram-based accelerators," *Science China Information Sciences*, vol. 64, no. 6, p. 160407, 2021.

[17] W. Jiang, Q. Lou, Z. Yan, L. Yang, J. Hu, X. S. Hu, and Y. Shi, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 595–605, 2020.

[18] L. Han, R. Pan, Z. Zhou, H. Lu, Y. Chen, H. Yang, P. Huang, G. Sun, X. Liu, and J. Kang, "Comn: Algorithm-hardware co-design platform for non-volatile memory based convolutional neural network accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[19] X. Yang, S. Belakaria, B. K. Joardar, H. Yang, J. R. Doppa, P. P. Pande, K. Chakrabarty, and H. H. Li, "Multi-objective optimization of reram crossbars for robust dnn inferencing under stochastic noise," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.

[20] S. Negi, I. Chakraborty, A. Ankit, and K. Roy, "Nax: neural architecture and memristive xbar based accelerator co-design," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 451–456, 2022.

[21] A. Moitra, A. Bhattacharjee, Y. Kim, and P. Panda, "Xpert: Peripheral circuit & neural architecture co-search for area and energy-efficient xbar-based computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[22] G. Li, S. K. Mandal, U. Y. Ogras, and R. Marculescu, "Flash: F ast neura l a rchitecture s earch with h ardware optimization," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–26, 2021.

[23] O. Krestinskaya, M. E. Fouda, A. Eltawil, and K. N. Salama, "Towards efficient imc accelerator design through joint hardware-workload co-optimization," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2025.

[24] X.-J. Chen and C.-L. Yang, "Cimnet: Joint search for neural network and computing-in-memory architecture," *IEEE Micro*, 2024.

[25] H. Sun, Z. Zhu, C. Wang, X. Ning, G. Dai, H. Yang, and Y. Wang, "Gibbon: An efficient co-exploration framework of nn model and processing-in-memory architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4075–4089, 2023.

[26] O. Krestinskaya, K. N. Salama, and A. P. James, "Automating analogue ai chip design with genetic search," *Advanced Intelligent Systems*, vol. 2, no. 8, p. 2000075, 2020.

[27] Z. Yan, D.-C. Juan, X. S. Hu, and Y. Shi, "Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 859–864, 2021.

[28] O. Krestinskaya, K. Salama, and A. P. James, "Towards hardware optimal neural network selection with multi-objective genetic search," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[29] S. Huang, A. Ankit, P. Silveira, R. Antunes, S. R. Chalamalasetti, I. El Hajj, D. E. Kim, G. Aguiar, P. Bruel, S. Serebryakov, *et al.*, "Mixed precision quantization for reram-based dnn inference accelerators," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 372–377, 2021.

[30] B. Kang, A. Lu, Y. Long, D. Kim, S. Yu, and S. Mukhopadhyay, "Genetic algorithm-based energy-aware cnn quantization for processing-in-memory architecture," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 649–662, 2021.

[31] J. Peng, H. Liu, Z. Zhao, Z. Li, S. Liu, and Q. Li, "Cmq: Crossbar-aware neural network mixed-precision quantization via differentiable architecture search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4124–4133, 2022.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[34] C. Zhou, F. G. Redondo, J. Büchel, I. Boybat, X. T. Comas, S. Nandakumar, S. Das, A. Sebastian, M. L. Gallo, and P. N. Whatmough, "Analognets: Ml-hw co-design of noise-robust tinyml models and always-on analog compute-in-memory accelerator," *arXiv preprint arXiv:2111.06503*, 2021.

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[36] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

[37] C. White, W. Neiswanger, and Y. Savani, "Bananas: Bayesian optimization with neural architectures for neural architecture search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 10293–10301, 2021.

[38] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[39] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[40] M. Poyser and T. P. Breckon, "Neural architecture search: A contemporary literature review for computer vision applications," *Pattern Recognition*, vol. 147, p. 110052, 2024.

[41] Y. Guo, Y. Chen, Y. Zheng, P. Zhao, J. Chen, J. Huang, and M. Tan, "Breaking the curse of space explosion: Towards efficient nas with curriculum search," in *International Conference on Machine Learning*, pp. 3822–3831, PMLR, 2020.

[42] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on genetic and evolutionary computation*, pp. 1187–1194, 2007.

[43] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *Ieee access*, vol. 8, pp. 89497–89509, 2020.

[44] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[45] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2078–2087, 2020.

[46] T. Andrulis, J. S. Emer, and V. Sze, "Cimloop: A flexible, accurate, and fast compute-in-memory modeling tool," *arXiv preprint arXiv:2405.07259*, 2024.

[47] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pp. 304–315, IEEE, 2019.

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

[49] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306–2319, 2020.

[50] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, "Neurosim simulator for compute-in-memory hardware accelerator: Validation and benchmark," *Frontiers in artificial intelligence*, vol. 4, p. 659060, 2021.

[51] J. Choquette, E. Lee, R. Krashinsky, V. Balan, and B. Khailany, "3.2 the a100 datacenter gpu and ampere architecture," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, pp. 48–50, IEEE, 2021.

[52] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, *et al.*, "A 65nm 4kb algorithm-dependent computing-in-memory sram unit-macro with 2.3 ns and 55.8 tops/w fully parallel product-sum operation for binary dnn edge processors," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 496–498, IEEE, 2018.