nature reviews electrical engineering

https://doi.org/10.1038/s44287-024-00052-7

Review article

Check for updates

Neural architecture search for in-memory computing-based deep learning accelerators

Olga Krestinskaya[1], Mohammed E. Fouda[2], Hadjer Benmeziane[3], Kaoutar El Maghraoui[4], Abu Sebastian[3], Wei D. Lu[5], Mario Lanza[6], Hai Li[7], Fadi Kurdahi[8], Suhaib A. Fahmy[1], Ahmed Eltawil[1] & Khaled N. Salama[1]

Abstract

The rapid growth of artificial intelligence and the increasing complexity of neural network models are driving demand for efficient hardware architectures that can address power-constrained and resource-constrained deployments. In this context, the emergence of in-memory computing (IMC) stands out as a promising technology. For this purpose, several IMC devices, circuits and architectures have been developed. However, the intricate nature of designing, implementing and deploying such architectures necessitates a well-orchestrated toolchain for hardware–software co-design. This toolchain must allow IMC-aware optimizations across the entire stack, encompassing devices, circuits, chips, compilers, software and neural network design. The complexity and sheer size of the design space involved renders manual optimizations impractical. To mitigate these challenges, hardware-aware neural architecture search (HW-NAS) has emerged as a promising approach to accelerate the design of streamlined neural networks tailored for efficient deployment on IMC hardware. This Review illustrates the application of HW-NAS to the specific features of IMC hardware and compares existing optimization frameworks. Ongoing research and unresolved issues are discussed. A roadmap for the evolution of HW-NAS for IMC architectures is proposed.

Sections

[1]Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia. [2]Rain Neuromorphics, San Francisco, CA, USA. [3]IBM Research Europe, Ruschlikon, Switzerland. [4]IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. [5]Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI, USA. [6]Physical Science and Engineering Division, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia. [7]Electrical and Computer Engineering Department, Duke University, Durham, NC, USA. [8]Electrical Engineering and Computer Science Department, University of California at Irvine, Irvine, CA, USA. e-mail: ok@ieee.org; khaled.salama@kaust.edu.sa

Key points

- Hardware-aware neural architecture search (HW-NAS) is an efficient tool in hardware–software co-design, and it can be combined with other architecture-level and system-level optimization techniques to design efficient in-memory computing (IMC) hardware for deep learning accelerators.
- HW-NAS for IMC can be used for optimizing deep learning models for a specific IMC hardware, and co-optimizing a model and hardware design searching for the most efficient implementation.
- In HW-NAS, it is important to define a search space, select an appropriate problem formulation technique, and consider the trade-off between performance, search speed, computation demands and scalability when selecting a search strategy and a hardware evaluation technique.
- In addition to neural network model hyperparameters and quantization and pruning policies, HW-NAS for IMC can include the circuit-level and architecture-level hardware parameters in the search.
- The main challenges in HW-NAS for IMC include a lack of unified framework to support different types of neural network models and different IMC hardware architectures, HW-NAS benchmarks and efficient software–hardware co-design techniques and tools.
- Fully automated NAS methods capable of constructing new deep learning operations and algorithms suitable for IMC with minimal human design are needed.

Introduction

The proliferation of the Internet of things is fuelling an unprecedented surge in data generation and advanced processing capabilities to cater to the intricate demands of applications leading to rapidly developing artificial intelligence (AI) systems. The design and implementation of efficient hardware solutions for AI applications are critical, as modern AI models are trained using machine learning (ML) and deep learning algorithms processed by graphic processing unit (GPU) accelerators on the cloud[1]. However, high energy consumption, latency and data privacy issues associated with cloud-based processing have increased the demand for developing efficient hardware for deep learning accelerators, especially for on-edge processing[2]. One of the most promising hardware architectures executing deep learning algorithms and neural networks at the edge is in-memory computing (IMC)[3]. This entails carrying out data processing directly within or in close proximity to the memory. The cost related to data movement is thus reduced, and notable enhancements in both latency and energy efficiency for computation and data transfer operations are obtained[4–6].

Efficient and functional IMC systems require the optimization of the design parameters across devices, circuits, architectures and algorithms. Effective hardware–software co-design toolchains are needed to connect the software implementation of neural networks with IMC hardware design[7]. Hardware–software co-design requires optimizations at each level of the process, which involves hundreds of parameters and is difficult to perform manually. Hardware-aware neural architecture search (HW-NAS) is a hardware–software co-design method to optimize a neural network while considering the characteristics and constraints of the hardware on which this neural network is deployed. Depending on the goal and problem setting, HW-NAS can also be used to optimize the hardware parameters themselves, such as the design features contributing to hardware efficiency. The parameter space for state-of-the-art neural network models can easily reach the order of 10^{35} (ref. 8), making it impossible to search for the optimal parameters manually. Traditional NAS frameworks are efficient at automating the search for optimum network parameters (for example network layers, blocks and kernel sizes)[9]. However, they do not consider hardware constraints, nor can they optimize the parameters of the hardware itself. HW-NAS extends conventional NAS by seamlessly integrating hardware parameters and characteristics, streamlining the efforts of hardware designers and software programmers alike[10]. HW-NAS can automate the optimization of neural network models given hardware constraints such as energy, latency, silicon area and memory size. Moreover, certain HW-NAS frameworks have the capability to optimize the parameters of the hardware architecture that is best suited for deploying a given neural network. HW-NAS can also help to identify trade-offs between performance and other hardware parameters[11]. Moreover, conventional NAS frameworks often incorporate operations that are not suited to IMC hardware and disregard inherent IMC hardware non-idealities such as noise or temporal drift within the search framework[12]. The absence of established IMC NAS benchmarks compounds these challenges. To close these gaps, since the beginning of the 2020s, several HW-NAS frameworks for IMC architectures have been proposed[13–17], some of which support a joint search of neural network parameters and IMC hardware parameters. These IMC hardware parameters include crossbar size, the resolution of the analog-to-digital converter or digital-to-analog converter (ADC/DAC), buffer size and device precision.

Existing NAS surveys focus on the software and algorithmic perspectives, discussing search approaches, optimization strategies and search-space configurations[18–20]. Reviews related to HW-NAS discuss a taxonomy of HW-NAS methods, search strategies, optimization techniques, hardware parameters relating to different types of hardware, such as central processing units (CPUs), GPUs, field-programmable gate arrays (FPGAs) and traditional application-specific integrated circuits (ASICs)[9–11,20,21]. However, an in-depth review of HW-NAS specifically for IMC, with consideration for its unique properties and the available hardware frameworks is not available.

In this Review, we discuss HW-NAS methods and frameworks focusing on IMC architecture search space. We compare existing frameworks and identify research challenges and open problems in this area. A roadmap for HW-NAS for IMC is outlined. Moreover, we provide recommendations and best practices for effective implementation of HW-NAS frameworks for IMC architectures. Finally, we show where HW-NAS stands in the context of IMC hardware–software co-design, highlighting the importance of incorporating IMC design optimizations into HW-NAS frameworks.

In-memory computing background

In traditional von Neumann architectures (Supplementary Fig. 1a), the energy cost of moving data between the memory and the computing units is high. Processor parallelism can alleviate this problem to some extent by performing more operations per memory transfer, but the cost of data movement remains an issue[4]. Moreover, von Neumann architectures suffer from the memory wall problem — that is, the speed of processing has improved at a much faster rate than that of traditional memories (such as dynamic random access memory

(DRAM)), resulting in overall application performance being limited by memory bandwidth[4,22]. Power efficiency has also ceased scaling with technology node advances, resulting in stalled gains in performance computational density[23].

In in-memory computing (IMC), one approach proposed to overcome the von Neumann bottleneck, data processing is performed within memory, by incorporating processing elements (Supplementary Fig. 1b). This alleviates the cost of data movement and improves the latency and energy required for both computation and data transfer[4,24–26]. Tiled architectures for IMC[27] are based on a crossbar array of memory devices which can perform multiply–accumulate (MAC) or matrix–vector multiplication (MVM) operations efficiently. Because MVMs constitute the vast majority of their associated operations, the tiled architecture is ideal for hardware realization of deep neural networks[28]. To implement an efficient IMC system, device-level[4,29–32], circuit-level[33–35] and architecture-level[36] design aspects should be considered[37].

In-memory computing devices and technologies

IMC can be built using charge-based memories, such as static random access memory (SRAM) and flash memory, or using resistance-based memories, such as resistive random access memory (RRAM), phase-change memory (PCM), magnetoresistive random access memory (MRAM), ferroelectric random access memory (FeRAM) and ferroelectric field-effect transistors (FeFETs)[4] (Supplementary Fig. 1c). SRAM is a well-developed volatile memory used for IMC applications, whereas the other IMC devices mentioned are non-volatile. Overall, non-volatile memories are less mature than traditional SRAM-based memories but have promising potential for IMC-based neural network hardware due to high storage density, scalability and non-volatility properties. The configuration and operating principle of volatile and non-volatile memories are described in Supplementary Note 1.

The important criteria used to select devices for IMC architectures are access speed for read and write operations, write energy, endurance, scalability, cell area, and cost (Supplementary Fig. 1d). To implement state-of-the-art neural networks, both read and write operation speeds of IMC devices are important for both training and inference, especially considering dynamic operations in some models, such as transformer-based neural networks[38]. SRAM has the lowest write latency (<1 ns) and highest endurance (>10^{16} cycles), compared with non-volatile IMC memory devices (Supplementary Fig. 1d). SRAM and some non-volatile memories, such as MRAM, FeRAM and FeFET, have low write energy (<0.1 nJ for SRAM and <1 nJ for non-volatile memories), which can contribute to faster neural network training and dynamic operations. Non-volatile memories, such as RRAM, PCM and Flash memories, have the smallest cell area (approximately 10–16F^2, where F is the minimum lithography feature size) and higher scalability and storage density due to the possibility of multilevel storage. In most IMC architectures, memories are organized in a 2D array, but 3D integration and 3D stacking can offer higher storage density. Overall, the most common and scalable memory devices for IMC architectures are SRAMs, RRAMs and PCMs[39]. Flash-array-based IMC accelerators also show promising results for neural networks and ML applications[40]. Other possible IMC devices include spin-torque MRAM, electrochemical RAM and memtransistors[29]. However, they are less common and are at an early stage of development.

Conventional IMC architectures for neural networks

A typical IMC architecture has several layers of hierarchy[33,41–44] (Supplementary Fig. 1e). The highest layer is constituted by tiles typically connected through a network-on-chip that includes the routers to transmit the signal between the tiles. The weight matrix of a neural network can be stored inside a single tile or shared between several tiles. This layer also includes peripheral and interface circuits, such as the global accumulation unit, pooling unit and input–output interface circuits. A tile consists of several computing elements[42], also called MAC units or MVM units[41,43], and peripheral circuits, including accumulation and activation units. Each computing element contains several crossbar arrays (processing elements) and processing circuits, including multiplexers shared by several crossbar columns, shift-and-add circuits, ADC converters, local registers and control circuits. A crossbar array contains memory cells with one or more devices depending on IMC memory type used in the design.

Some device technologies, such as RRAM and PCM, use a device in series with a switching element to mitigate sneak path currents from cell to cell (which would result in false cell programming or reading) and limit the current in the low-resistance state (to avoid damage and improve variability and endurance). This device can be a two-terminal threshold-switching selector located above or below the resistance-based memory (namely 1S1R); or a complementary metal–oxide–semiconductor transistor (1T1R). This choice typically increases the size of each cell (the resistance-based memory is integrated on the via that comes from the drain and source contacts of the transistor)[30]. State-of-the-art IMC architectures include ISAAC[41], PUMA/PANTHER[43,45], TIMELY[46], RIMAC[47], PIMCA[48], HERMES[39] and SAMBA[49]. ISAAC[41] is a pipelined IMC accelerator with embedded DRAM buffer banks to store intermediate outputs of the pipeline stages. PUMA[45] is a programmable eight-core accelerator with a specialized instruction set architecture and compiler supporting complex workloads. PANTHER[43] is an extension of PUMA architecture supporting efficient training for RRAM-based IMC architectures. TIMELY[46] adopts analog local buffers inside the crossbar arrays to improve data locality, and time-domain interfaces to improve energy efficiency. RIMAC[47] is an ADC/DAC-free IMC accelerator with analog cache and computation modules. PIMCA[48] is a capacitive-coupling-based SRAM IMC accelerator with a flexible single-instruction, multiple-data processor for non-MVM operations. SAMBA[49] is a sparsity-aware RRAM-based IMC accelerator with load balancing and optimized scheduling.

An alternative to hierarchical architectures is one that combines spatially distributed analog IMC tiles and heterogeneous digital computing cores[50]. Such an architecture, based on 2D-mesh interconnect[51], is highly programmable and supports a wide range of workloads (mapping and pipelining). TAICHI[51] is another example of a tiled RRAM-based accelerator with mesh interconnect, local arithmetic units and global co-processor targeting reconfigurability and efficiency.

Since the beginning of the 2020s, several fabricated IMC macros have been demonstrated: a reconfigurable 48-core RRAM-based IMC chip (NeuRRAM) suitable for various applications, such as image classification, speech recognition and image reconstruction[52]; eight-core RRAM-based IMC macros[53,54]; a PCM-based fabricated eight-core chip[55]; a flash-memory-based pipelined 76-core chip with analog computing engine tiles[40]; a SRAM-based mixed-signal 16-core IMC accelerator with configurable on-chip network, flexible dataflow and scalable cores[56]; and a MRAM-based IMC core with readout circuits[57]. In 2023, a mixed-signal IMC chip called the IBM HERMES Project Chip, comprising 64 cores of PCM-based crossbar arrays with integrated data converters, on-chip digital processing and a digital communication fabric, was presented[39].

Weight mapping, computing precision and non-idealities

Several software and hardware parameters need to be considered to map a software-based neural network model to IMC hardware. For example, when mapping neural network weight matrices and inputs or activations to IMC crossbars, important parameters are the matrix size, the crossbar size, the precision of weights and inputs, the precision of IMC devices, the resolution of the converters (ADCs and DACs) and the peripheral circuits. In this case, the concept of partial sums should be considered. Partial sums in IMC architectures are applied in three different cases (Supplementary Fig. 1f): (1) when a large weight matrix does not fit into a single crossbar array; (2) when high-precision weights are mapped to low-precision crossbar cells; and (3) when high-precision inputs are streamed to the crossbar sequentially[58] (discussed in detail in Supplementary Note 2). Partial sums require specific ADC resolution to maintain the desirable computing precision (Supplementary Note 2), which contributes to on-chip area and energy consumption overhead of peripheral circuits[43].

In IMC architectures with non-volatile memory devices, computing precision is also affected by non-idealities, including device-to-device variability, circuit nonlinearity, and conductance variation with time[38,59]. Such degradation of computing precision can be predicted and mitigated using hardware-aware training methods leading to robust IMC hardware designs[38,60]. The accuracy degradation caused by non-idealities of the devices can also be improved by periodically calibrating batch normalization parameters during the inference[59]. Overall, it is necessary to conduct a comprehensive analysis of noise effects when designing IMC hardware and include mitigation and compensation techniques for non-idealities as part of the design.

Model compression for IMC architectures

Model compression techniques used for neural network optimization, such as quantization and pruning[61], can be applied in implementations on IMC architectures to reduce hardware costs. It is too expensive to deploy full-precision neural network weights to IMC devices[62], so quantization is often used, reducing occupied memory, data transmission and computation latency[63]. Network pruning, which removes unnecessary neural network weights or groups of weights, can reduce energy and latency[64].

Quantization for IMC architectures. Quantization methods are divided into uniform quantization and non-uniform quantization[65]. In uniform quantization, the quantization intervals are equally distributed across the quantized range[58]. An example of non-uniform quantization is logarithmic quantization, that is the power-of-two method[66] commonly used for SRAM-based IMC hardware or RRAM-based binarized IMC architectures. More complex quantized weight representations use bit-level sparsity to increase the number of zeros in the bit representation of weights and activations to improve energy efficiency in MAC operation during quantization[67,68]. Most quantization-related RRAM-based architectures focus on fixed-precision quantization with a uniform quantizer[62].

An alternative approach is based on mixed precision quantization, where different quantization configurations are chosen for different layers[63,69]. This method is effective because different neural network layers and convolution filters have different sensitivity to quantization[62] and contribute differently to overall accuracy[69]. Flexible word length quantization improves compression and reduces accuracy loss compared with uniform quantization[70].

In IMC architectures, quantization is performed by either changing the number of crossbar cells per weight or the number of bits per crossbar cell. Analog weight storage (≥2 bits per cell) allows for higher effective cell density (bits mm^{-2}), leading to higher storage density[39]. However, increasing the number of bits per cell increases the effect of RRAM non-idealities and variabilities. ADC precision, and hence overhead, also increases with the precision of crossbar weights[63].

Pruning in IMC architectures. Pruning is divided into unstructured and structured pruning. In unstructured pruning, the individual connections (weights) are removed at a fine granularity. Structured pruning implies coarse-grained pruning of groups of weights (kernel or channel pruning)[64,71]. In IMC hardware, weight pruning (usually unstructured pruning of individual weights) disconnects unnecessary crossbar cells, leading to sparse neural network hardware implementation. Structured pruning is implemented by removing or disconnecting whole rows or columns of the crossbar array and corresponding peripheral circuits.

Sparsity due to unstructured pruning in IMC architectures can improve energy efficiency and throughput, but it can also lead to unnecessary computation overhead, difficulty in parallelizing processing, and low hardware utilization. Nevertheless, mapping structured row-wise or column-wise pruning to IMC architecture leads to higher crossbar utilization than unstructured pruning[64]. One of the ways to reach a desired compression ratio via structured pruning is to incorporate several rounds of weight grouping, determining the importance of these groups, followed by fine-grained row-wise pruning[64].

Most work on RRAM-based neural network pruning uses heuristics or rules to prune network weights. This can sometimes prune non-trivial weights and preserve trivial weights, leading to sub-optimal solutions. Also, hardware constraints and hardware feedback are not always considered in RRAM-based network pruning[72].

Hardware-aware neural architecture search

In the hardware-aware neural architecture search (HW-NAS) process, the inputs are neural network parameters, model compression parameters and hardware parameters (Fig. 1a). In some cases, the search space includes hardware search space and model compression options to be optimized. Optimal neural network designs and optimal hardware parameters are searched for within this space using a search strategy (algorithm or heuristic). The main difference between HW-NAS and traditional NAS is the consideration of hardware limitations and constraints in the search. Problem formulation methods are used to define an objective function for the search and a method to incorporate optimization constraints. Some frameworks search only for optimum neural network designs considering hardware constraints, whereas others can incorporate the search of optimal hardware parameters to find the most efficient hardware design. Performance is evaluated using performance metrics and hardware metrics. In this Review, performance metrics refer to a neural network performance characteristic such as accuracy or performance error, while hardware metrics refer to the metrics describing hardware efficiency, such as energy, latency and on-chip area. To evaluate hardware performance, various hardware cost estimation methods can be used.

HW-NAS basics

HW-NAS for IMC incorporates four efficient deep learning methods for design space exploration (Fig. 1b), which allow the optimal design of neural network model and hardware to be found: model compression,

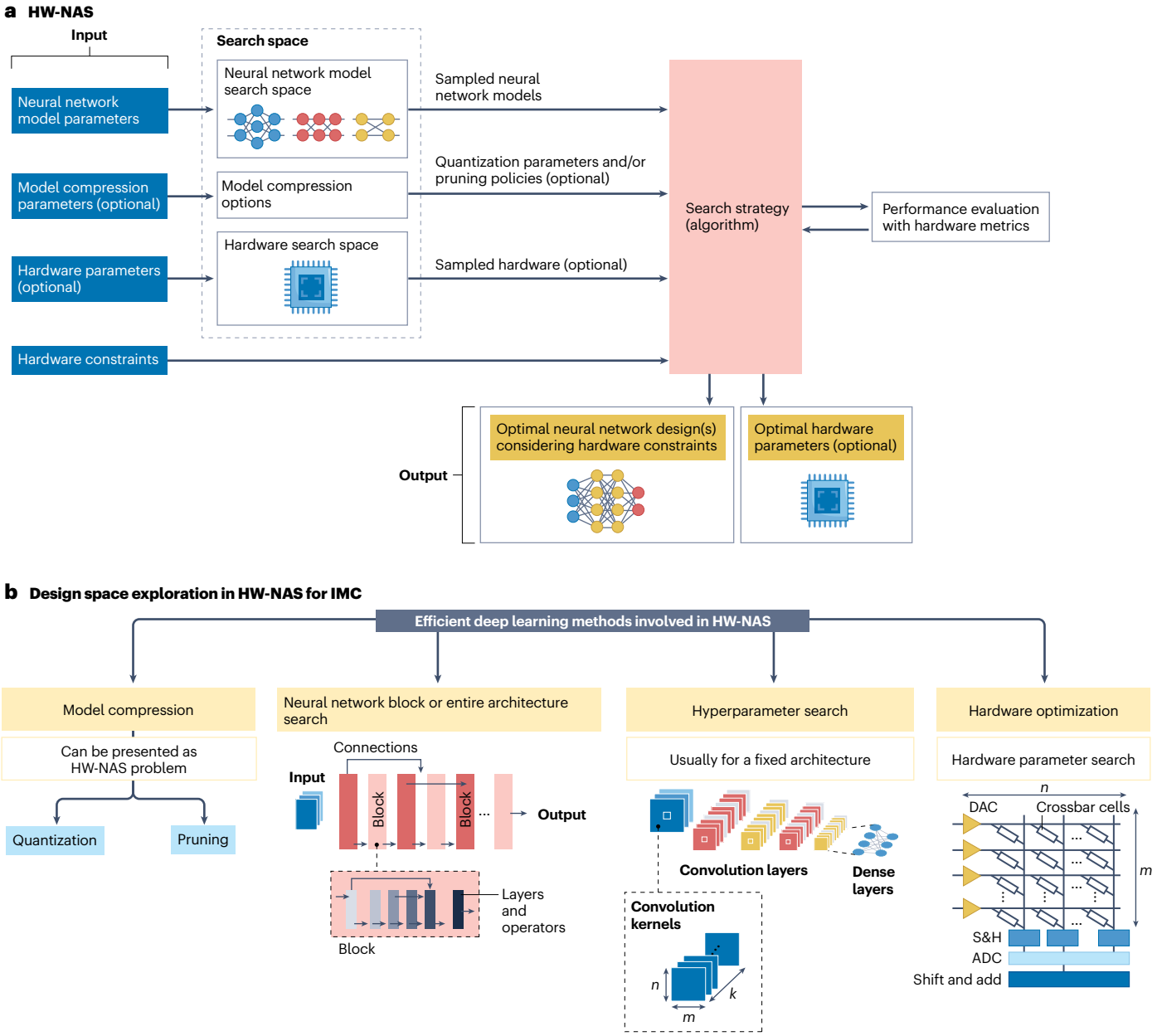

Fig. 1 | Fundamentals of hardware-aware neural architecture search. a, Overview of hardware-aware neural architecture search (HW-NAS). **b**, Efficient deep learning methods for the design space exploration involved in HW-NAS for in-memory computing (IMC). ADC, analog-to-digital converter; DAC, digital-to-analog converter; S&H, sample and hold.

neural network model search, hyperparameter search and hardware optimization. Model compression techniques, such as quantization and pruning, can be viewed as HW-NAS problems and are often included in HW-NAS flows[9]. Neural network model search implies searching for neural network layers and corresponding operations, as well as the connections between them[73,74]. Hyperparameter search includes searching for the optimized parameters for a fixed network — that is, the number of filters in a convolution layer or the kernel size[9]. Hardware optimization is the optimization of hardware components such as tiling parameters, buffer sizes and other parameters included in the hardware search space. For IMC architectures, hardware optimization may include crossbar-related parameters (such as ADC/DAC precision and crossbar size) that can have an effect on the performance of the architecture, in terms of energy consumption, processing speed, on-chip area and performance accuracy[15,75].

The search space in HW-NAS refers to the set of network operations and hyperparameters searched to optimize the network and hardware architecture. The search space can be fixed or hardware-aware[11]. In a fixed search space, neural network operations are designed manually without considering the hardware. In hardware-aware search,

the interlayer and intralayer operations in the network are adapted depending on the hardware.

From the perspective of the search parameters, the search space can be divided into the neural network search space and hardware architecture search space[9]. The first covers the selection of neural network operations, blocks, layers and the connections between them. The second considers the hardware design, for example IP core reuse on FPGA, quantization schemes, tiling, or selection of buffer sizes. The hardware architecture search space depends either on hardware platform configurations or on predefined templates for different operations to optimize certain hardware parameters[9]. For IMC architectures, the search space should be extended to include specific hardware-related details, such as crossbar size and precision of the converters (discussed later).

Depending on the HW-NAS objectives, frameworks can optimize a specific neural network or set of network models for a specific or multiple hardware architectures[11]. Depending on the target hardware, hardware constraints can vary. Hardware constraints can be categorized into implicit constraints and explicit constraints. Implicit constraints are those that do not describe desired hardware metrics directly but affect them implicitly, such as bits per operation. Explicit constraints are the evaluated metrics related to hardware deployment, including energy consumption, latency, on-chip area, memory and available hardware blocks. Typical constraints for IMC architectures include energy consumption, latency and the number of available resources (for example crossbar tiles on a chip).

Problem formulation in HW-NAS

A problem formulation in HW-NAS defines the objective function, optimization constraints and how the problem is formulated. The problem formulation method is selected according to the target output and available information about hardware resources. For example, the HW-NAS target can be either a single architecture with optimized performance and hardware parameters or a set of architectures optimizing hardware parameters with a certain priority. HW-NAS problem formulation is divided into single-objective optimization and multi-objective optimization methods (Fig. 2). The selection of a HW-NAS problem formulation method depends on the objectives of the search. Two-stage methods are suitable for deploying well-performing models to a specific hardware or getting a sub-model from an existing model followed by a specific hardware deployment. Constrained optimization is useful for the case of specified hardware constraints or designing a neural network model for a specific hardware platform. Scalarization methods could be used when setting up the priority of a certain objective function and the Pareto-based optimization for finding the trade-off between the performance metrics and hardware metrics.

Single-objective optimization. These methods are categorized into two-stage methods and constrained optimization[9]. In two-stage optimization, HW-NAS first selects a set of well-performing high-accuracy network models and then performs hardware optimization and selects the most hardware-efficient design. It is useful to transform well-performing neural networks for implementation on different hardware platforms, or to optimize networks for a specific hardware platform. Hardware constraints are included in the second stage of HW-NAS. The drawback of such methods is that the selected network models in the first stage tend to be large to maximize accuracy and may not always fit the hardware constraints of a specific platform. In constrained optimization, hardware parameters are considered when searching for a neural network model. This allows filtering out of network models that do not fit within hardware constraints during the search process, thus speeding up HW-NAS. The challenge of constrained optimization is the difficulty of including hardware constraints directly in the search algorithms, especially in gradient-based methods or reinforcement learning. Therefore, the problem is often transformed into an unconstrained optimization that includes the hardware constraints in the objective function[9,76].

Multi-objective optimization. These methods are categorized into scalarization methods and Pareto optimization typically using the NSGA-II algorithm[9]. The first approach is a multi-objective optimization method when several objective functions are combined via weighted summation, weighted exponential sum or weighted product to set up the significance of the objective term. This approach is useful to set the priority for certain objective terms while not ignoring others, and modifying this weighting based on requirements. During a search, the weights are usually fixed, and multiple runs are required to find the Pareto optimal set or to get a 'truly' optimum network model. Therefore, speed is slow and depends on the number of search iterations with different weights. In the second method, a set of Pareto optimal neural network models is searched. The search can be implemented with the evolutionary algorithm NSGA-II[9], where the problem is formulated as a linear combination of different objective functions. This method is useful to find trade-offs between performance, accuracy and hardware metrics, especially when searching for the optimal network model for different hardware platforms. Search speed is slow compared with the previous methods, as the whole set of network models on the Pareto curve is searched.

Search strategies: algorithms for HW-NAS

After defining a problem formulation method, a search strategy should be selected (Fig. 3). The search algorithm is a core component of NAS and defines the flow of parameter search. There are three main optimization algorithms used for HW-NAS: reinforcement learning, evolutionary algorithms, and gradient-based methods, such as differentiable search. Less common algorithms include random search and Bayesian optimization. The search algorithm is independent of the problem formulation methods shown in Fig. 2. For example, two different search algorithms can be used in a two-stage problem formulation method in different optimization stages, in a hybrid approach[9]. Constrained optimization and scalarization method-based problem formulation can be combined with most search strategies, for example reinforcement learning, evolutionary algorithms and Bayesian optimization. Differentiable search is easier to apply for differentiable parameters search, for example, in the first stage of two-state methods when optimizing the neural network model parameters, while using any other algorithm (reinforcement learning or evolutionary algorithm) to optimize the hardware or fine-tuning the model to fit hardware constraints[8]. Pareto optimization problems are mainly addressed in the literature by evolutionary algorithm approaches, such as NSGA-II[77]; however, the Pareto optimal set can also be found using other methods[78].

In reinforcement-learning-based NAS search, an agent interacts with the environment and learns the best policy to take an action using a trial-and-error approach. The environment is modelled with a Markov decision process[63]. The algorithm aims to maximize the reward function. The main drawback of reinforcement learning is slow search speed and high computational cost. Reinforcement learning can be applied to RRAM-based architecture[63] or used to find the best-performing

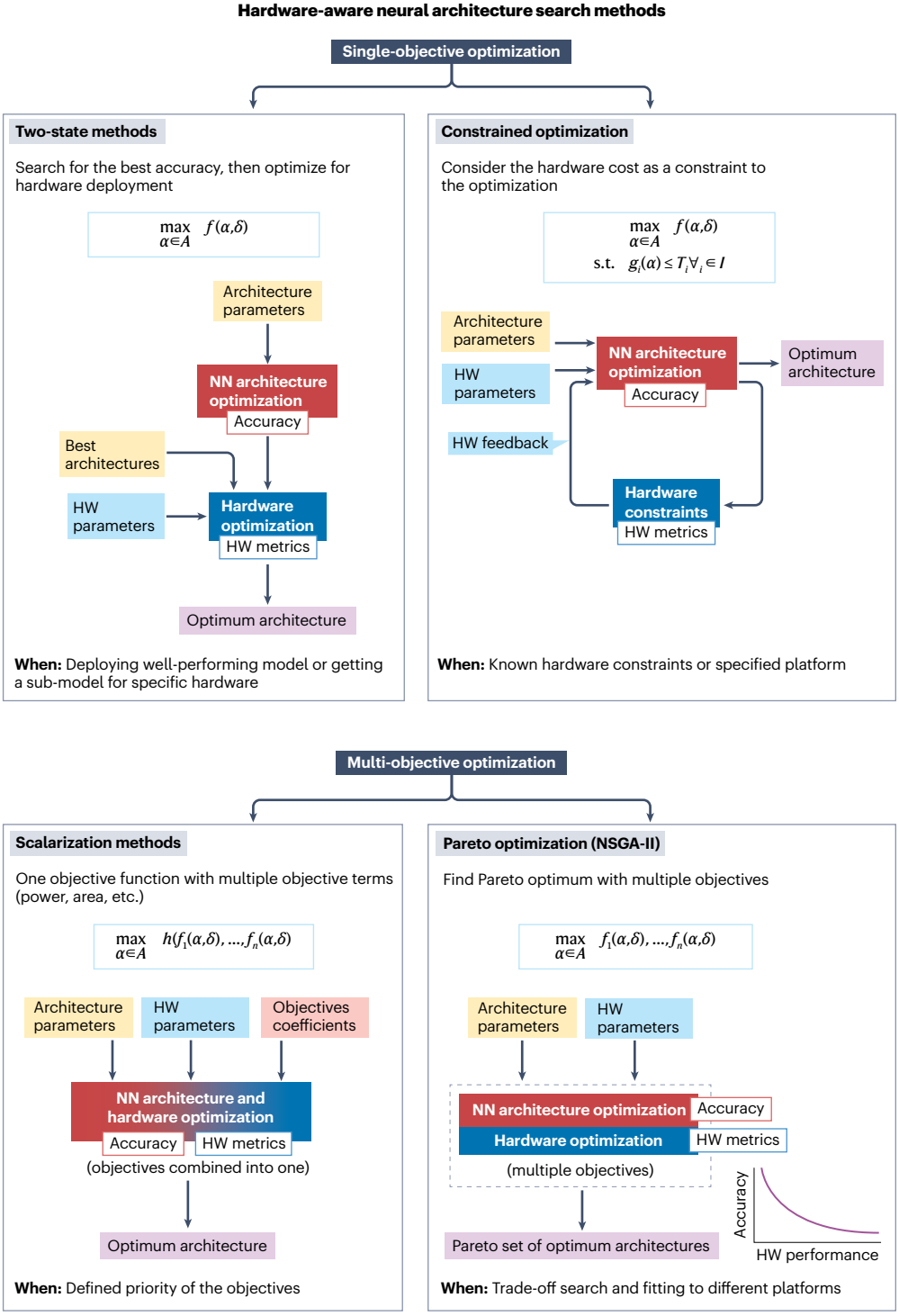

Fig. 2 | Problem formulation methods in hardware-aware neural architecture search. The methods search for a neural network (NN) model α from the search space A that maximizes a performance metric f for several performance metrics f_n for a dataset δ. Constrained optimization uses a hardware (HW) constraint g_i with a threshold T_i from a set of hardware constraints I. $h()$ is a combination of several performance metrics that can represent weighted summation, weighted exponential sum or weighted product. Data derived from ref. 9.

network models (considering the optimization of hardware resource utilization in an RRAM-based architecture)[69]. Reinforcement-learning-based automated weight pruning can also be applied for RRAM-based crossbars[72,79].

One of the best-known evolutionary algorithms is the genetic algorithm, which is used in several HW-NAS frameworks[9]. The first step of an evolutionary algorithm is the initialization of a population of networks with a random combination of parameters to start the search. The performance of the networks is then evaluated and scored based on an objective function (also called the fitness function). The best-performing networks are used in a mutation and crossover process, where the parameters of these networks are mixed and some of the

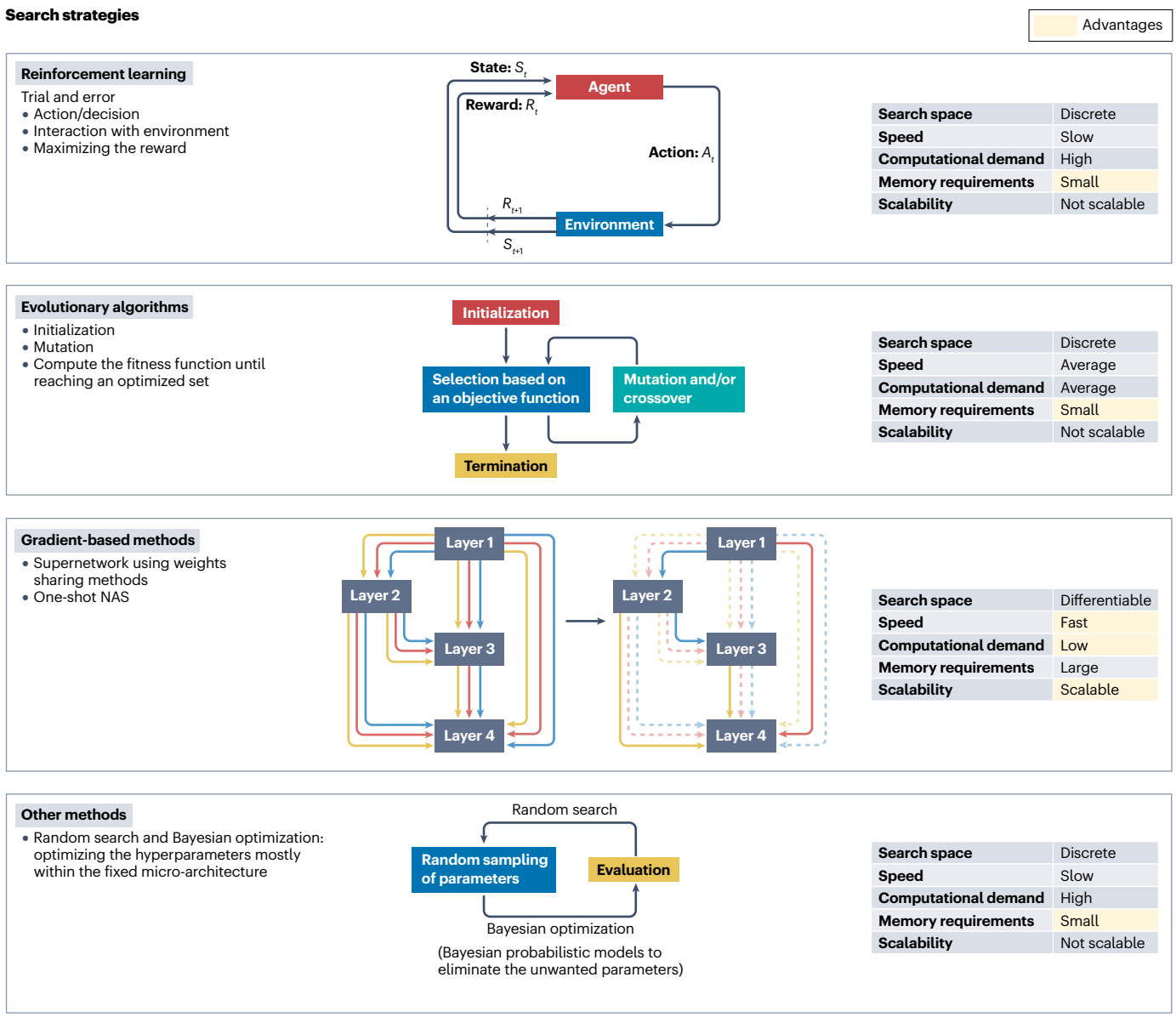

Fig. 3 | Search strategies in hardware-aware neural architecture search. Summary and comparison of search strategies and their performance parameters, computation and resource requirements. Search strategies in hardware-aware neural architecture search (HW-NAS) refer to the algorithms and search techniques used in the search for the optimum parameters. Scalability refers to how scalable the algorithm is when the search space grows. Even though the evolutionary algorithms are not scalable when the search space increases, they are often used as a second step in a two-state optimization, where the first step is a scalable gradient-based method with a super-network search space. Gradient-based methods are scalable, as the search time does not increase exponentially with search space size. Computational demand and memory requirements refer to the resources required to run the algorithm. The scalability of each algorithm in terms of memory consumption depends on the specific algorithm and may vary within the same subset of methods.

parameters are randomly changed to new ones to create a new population. The process is repeated starting from evaluation for several iterations, called generations, until convergence to a set of well-performing network models[70,80].

Gradient-based and differentiable search methods use the one-shot NAS approach and weight sharing in a super-network (an over-parametrized network)[81]. This approach combines all possible neural network models with different parameters in a single super-network using a weight-sharing technique. Compared with reinforcement learning and evolutionary algorithm methods, where the evaluated networks are randomly sampled at the beginning of a search from the existing parameters, the weights in a super-network in gradient-based methods are trained at the same time as the search is performed. After training, the super-network is pruned to eliminate inefficient connections

and blocks, creating a single optimum network. Differentiable search is the only scalable approach with a differentiable search space, where the search space does not grow exponentially with the increased number of searched hyperparameters. This approach is faster than other methods, as it does not require the training of every single neural network model and therefore has low computational demand. However, differentiable search has high memory requirements for storing an over-parameterized network. An example of differentiable NAS applied to RRAM-based architecture is CMQ[62]. The super-network search can be performed by evolutionary algorithm or Bayesian optimization methods, which are also suitable for a discrete search space[8].

Bayesian optimization[82] excels in managing complex search spaces by constructing and iteratively refining a probabilistic model of the objective function. It balances exploration of new architectures and exploitation of known effective configurations. This makes Bayesian optimization a strong contender for optimizing hyperparameters within a set macro-architecture of a neural network. However, it is important to note that the search speed of Bayesian optimization is relatively slow, comparable to that of reinforcement learning methods. Complementing Bayesian optimization, other strategies, such as random search[83], the multi-armed bandit approach[84] and simulated annealing[85], also contribute to the field. Random search, with its simplicity and unpredictability, offers a baseline performance metric and can be effective in certain high-dimensional search spaces. The multi-armed bandit approach, adept at efficiently navigating decisions under uncertainty, and simulated annealing, inspired by metallurgical cooling processes, both provide unique mechanisms for exploring the search space. These methods are valuable for their distinctive ways of handling search space complexities and often find use in scenarios where more advanced techniques may not be as suitable or necessary.

Overall, reinforcement learning[16] and evolutionary algorithms[12] can produce good results, but they are slow and not scalable. The search space and search time required for reinforcement learning and evolutionary algorithms increase exponentially with the number of searched hyperparameters. This problem is addressed by differentiable search[14], which is scalable and faster than reinforcement learning and evolutionary algorithms. Therefore, differentiable search is useful for a large search space with many parameters. However, it is important to consider the gradient estimation techniques for non-differentiable parameters. Bayesian optimization is also a promising search strategy. Nevertheless, the application of Bayesian optimization for IMC hardware has not been explored yet.

Hardware cost estimation methods

A major factor in HW-NAS is the estimation methods for hardware performance. There are four different methods for evaluation of a hardware cost (Fig. 4): real-time estimation, lookup table (LUT)-based methods, analytical estimation, and prediction models.

In real-time measurement-based methods, the hardware evaluation is performed directly on target hardware in real time. In FPGA-based or microcontroller-based designs, this implies the deployment of the network model to real or simulated hardware[9]. For IMC architectures, this can be performed directly on IMC chip or using circuit-level simulations such as SPICE, which is difficult to automate. This method ensures highly accurate performance estimation, and it is also scalable for different models and across different hardware platforms; however, it is slow, inefficient and impractical.

LUT-based methods involve the separate evaluation of hardware metrics for every hardware parameter and its storage in a large LUT[8,86]. During hardware evaluation in HW-NAS, LUTs are used to calculate the total hardware metrics, which are the total energy, the latency or the on-chip area, using the stored results. LUT-based methods are less accurate than real-time measurements and prediction methods, especially when the communication between crossbar tiles or other hardware blocks in IMC architecture is not considered. LUT-based methods are less scalable than other methods, as with the increased search space, the number of required measurements grows combinatorically with the number of parameters. In addition, LUT-based methods are moderately scalable across different neural network models and require regeneration when transferred to other hardware platforms.

Analytical estimation methods imply computing rough estimates of hardware metrics using mathematical equations (such as DNN+NeuroSim[87], MNSIM[88,89], AIHWKit[90] and PUMAsim[45] for IMC-based hardware). Such a method is fast and highly scalable when the search space is increased. The scalability of this method across different neural network models depends on how similar a new model is to the already estimated one. The transferability across hardware platforms also depends on the similarity of the hardware architectures. However, it still is not as accurate as real-time measurements. It also requires the initial estimation of hardware metrics from real-time hardware or circuit-level simulations.

Prediction-based methods are based on ML and are trained to use a linear regression or neural-network-based approach to predict hardware metrics[91–93]. These methods require an initial set of hardware parameters and hardware metrics stored in LUTs to train ML models and are fast and highly scalable when adding new hyperparameters to the search space. The scalability across neural network models depends on the similarity of the models. Prediction-based methods support differentiable NAS methods and are more accurate than analytical estimation and LUT-based methods.

Other HW-NAS considerations

Each search strategy and algorithm contains a sampling part, where the neural network models are sampled. In evolutionary algorithm and reinforcement learning-based optimization frameworks, the network models are sampled before the search. In contrast, in differentiable NAS, the best-performing models are sampled after training a supernet. The most common sampling method is uniform and random-uniform sampling, which is simple and effective[94]. Other methods include Monte Carlo sampling, which guarantees good performance and diversity owing to the randomness of the sampling process, and the Stein variational gradient descent algorithm with regularized diversity, which provides a controllable trade-off between the single-model performance and the predictive diversity[95]. More advanced sampling methods include attentive sampling aiming to produce a better Pareto front, such as AttentiveNAS framework[96], and dynamic adaptive sampling[97], where the sampling probability is adjusted according to the hardware constraints.

As search parameters of a neural network models can be non-differentiable, one of the main issues in differentiable NAS is the relaxation of non-differentiable parameters when applying differentiable search methods. As these methods require gradient calculation, the search space should be differentiable. When it comes to HW-NAS and searching for the hardware parameters, this becomes an even more critical issue, as most of the hardware parameters and hardware metrics are non-differentiable[98]. These relaxation methods allow gradient computation over discrete variables. The most common methods include estimated continuous function, the REINFORCE algorithm, and

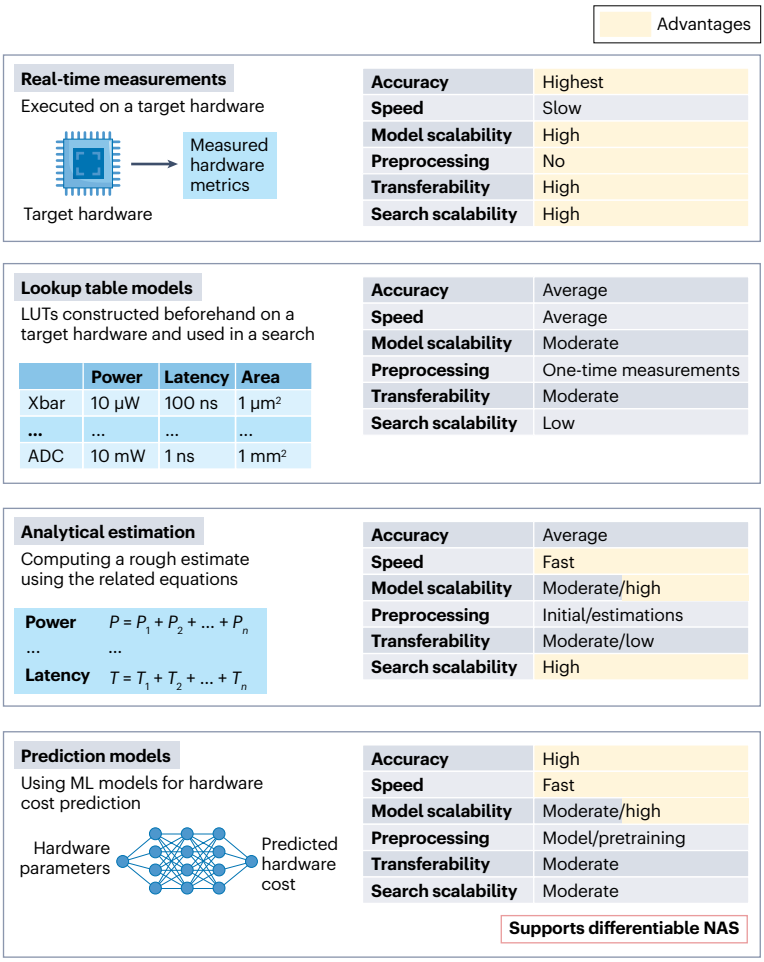

Fig. 4 | Hardware cost estimation methods for hardware-aware neural architecture search. Model scalability refers to the scalability across different neural network models. In analytical estimation, model scalability depends on how similar a new model is to the one with already estimated hardware cost metrics. Transferability refers to transferability across different hardware platforms. Transferability of lookup table (LUT) models and prediction models is moderate, as it requires regeneration of LUTs and machine learning (ML) models for prediction, respectively. Transferability of the analytical estimation depends on how similar new hardware is to the already estimated one. Search scalability refers to the scalability with increasing size of the search space, when the number of search hyperparameters grows. ADC, analog to digital converter; NAS, neural architecture search; Xbar, crossbar. Data derived from ref. 9.

application of the Gumbel Softmax function using random Gumbel noise in a computation[9].

The main challenge in HW-NAS is search speed and runtime performance. To improve the search speed of HW-NAS, several techniques, including early stopping, hot start, proxy datasets and accurate prediction models, are used[9]. In early stopping, the change in a loss function is monitored for the first few training iterations instead of training the neural network modes completely. In the hot start technique, the search starts from the well-performing efficient network models rather than random ones. In the application of the proxy dataset, small simple datasets are used in the search first, and then the search results are fine-tuned for more complex datasets. To speed up the search, accuracy prediction methods can also be used for accuracy estimation instead of training every sampled network[9].

HW-NAS for IMC architectures

State-of-the-art HW-NAS frameworks for IMC

Manually searching for both the optimum design and processing of in-memory architecture is unrealistic, as a search space becomes huge when adding the architecture parameters[16] (Supplementary Fig. 2a). Besides neural network blocks, hyperparameter search and optimized compression techniques, the search space for IMC architectures can be expanded to search for IMC crossbar-related hardware components[15–17,63]. The IMC hardware search space considered in these frameworks includes IMC crossbar size, ADC/DAC precision, device precision and buffer size.

Between 2020 and 2023, several HW-NAS frameworks for IMC-based neural network architectures have been introduced (Supplementary Fig. 2b–d and Table 1). Based on which parameters are searched (Supplementary Fig. 2b), the HW-NAS methods for IMC architectures can be divided into three main categories: (1) frameworks containing the 'true' NAS searching for the neural network components and hyperparameters[12–17,80,99–101]; (2) frameworks in which quantization is presented as an HW-NAS problem, and optimum bit-width is searched considering the hardware feedback[62,69,70]; and (3) frameworks searching for optimum pruning, formulating the problem as HW-NAS[72,79]. Compared with 'true' NAS approaches, frameworks focused on only quantization or pruning search for optimized model compression techniques while using HW-NAS problem formulation techniques.

Based on the consideration of hardware parameters in a search (Supplementary Fig. 2c), HW-NAS frameworks for IMC can be divided into three categories: (1) frameworks for a fixed IMC architecture optimizing a neural network model for a fixed hardware[12–14,62,69,70,72,79,80,99–101], (2) frameworks with hardware parameters search for a fixed model optimizing IMC hardware for a certain application[102], and (3) frameworks for optimum model and architecture search optimizing both

Table 1 | State-of-the-art hardware-aware neural architecture search frameworks for in-memory computing

Framework	Q	P	N	Network model search space	Hardware search space	Hardware considerations	Algorithm	Hardware non-idealities	C	Performance gains and outcomes
AnalogNAS[12] (2023)	—	—	✓	Number of blocks, channels, branches, kernel size, widening factor	—	AIHWKit[90]	EA	Device variations, conductance drift	✓	2% ↑ accuracy, 4× ↓ time, 1.2× ↑ energy eff. (vs ResNet32[a,b])
Pareto-based NAS[14] (2022)	—	—	✓	Convolution layer width, depth, expansion ratio	—	DNN+NeuroSim[87]	DS	—	—	Pareto front (accuracy vs latency)
NAS4RRAM[13] (2021)	—	—	✓	Layers, output channels (residual blocks)	—	Simulator for RRAM-based accelerator	EA	Device variations (thermal/shot noises, RTN)	—	>6% ↑ accuracy (vs ResNet32[b])
FLASH[99] (2021)	—	—	✓	Number of skip connections, cells, layers per cell, channels per layer	—	NeuroSim[87] and BookSim[119]	SHGO	—	—	>27,729× ↑ speed than RL
UAE[101] (2021)	✓	—	✓	Number of channels, filter height/width, weights bit-width (integer/fraction bits)	—	Analytical	RL	Device variations (thermal/shot noises, RTN), programming errors	—	6.3% ↑ accuracy[b] (vs NACIM[17]), robust to device variations
GA for IMC AI hardware[80,100] (2020)	—	—	✓	Number of layers, neurons, channels, activation functions, kernel size	—	Analytical	GA	Device variations (Gaussian), conductance deviation, device failure	—	~2× ↑ speed vs grid search for small networks, ↑↑ accuracy vs using NAS[c]
NAX[15] (2021)	—	—	✓	Kernel size	Crossbar size	GENIEx[112] (non-idealities)	DS	Wire resistances, source/sink resistances	—	0.8% ↑ accuracy 17% ↓ EDAP (vs ResNet20[b])
Gibbon[16] (2022)	✓	—	✓	Number of blocks, output channels, groups (for group convolution), kernel size, weights/activations bit-width	Crossbar size, ADC/DAC resolution, device precision	MNSIM[89]	EA	Device variations	—	8.4–41.3× ↑ speed, 10.7% ↑ accuracy[b], 6.48x ↓ EDP (vs refs. 13,17,101)
NACIM[17] (2020)	✓	—	✓	Network hyperparameters, weights bit-width (integer/fraction bits)	Tile size, buffer size, bandwidth	DNN+NeuroSim[87]	RL	Device variations	—	3% ↑ accuracy[b] (vs VGG11), high accuracy with device variations
CF-MESMO for RRAM[102] (2021)	—	—	—	—	Crossbar size, device precision, frequency	DNN+NeuroSim[87]	CF-MESMO	Device variations (thermal/shot noises, RTN)	—	Pareto front and 90.91% ↓ in computation cost (vs NSGA-II)
CMQ[62] (2022)	✓	—	—	Quantization threshold and weights bit-width	—	MINT[120]	DS	Device variations (Gaussian)	—	2.04% ↑ accuracy[b] (vs fixed-precision model for ResNet20)
Mixed-precision quantization[63] (2021)	✓	—	—	Weights bit-width (total+fraction), inputs bit-width (total+fraction)	ADC precision	PUMAsim[45]	RL	—	—	4.84× ↓ energy, 3.98× ↓ latency (vs 16-bit LeNet model)
EGQ[70] (2021)	✓	—	—	Weights/activations bit-width	—	DNN+NeuroSim[87]	GA	—	—	1.2–1.6 ↑ TOPs/W[d], 5–25% ↓ area[b] (vs fixed-precision model for VGG8)
RaQU[69] (2021)	✓	—	—	Weights/kernels bit-width	—	Analytical	RL	—	—	18% ↑ utilization, 3.3% ↑ accuracy[b] (vs fixed-precision model for ResNet18)
ASBP[79] (2021)	—	✓	—	Bits of weights	—	Analytical	RL	—	—	79% ↓ energy, 55% ↓ area[b] (vs unpruned bit model for ResNet18)
Auto-prune[72] (2021)	—	✓	—	Weights (pruned unimportant columns)	—	MNSIM[89]	RL	—	—	9x ↑ area eff., 12× ↑ energy eff.[b] (vs unpruned bit model for VGG16)

ADC/DAC, analog to digital or digital to analog converters; AI, artifical intelligence; C, available open-source code; DS, differentiable search; EA, evolutionary algorithm; EDAP, energy–delay–area product; EDP, energy–delay product; eff., efficiency; GA, genetic algorithm; HW-NAS, hardware-aware neural architecture search; IMC, in-memory computing; N, neural architecture search; P, automated pruning; Q, quantization search; RL, reinforcement learning; RRAM, resistive random-access memory;RTN, random telegraph noise. [a]Hardware implementation of ResNet32 with phase-change memory devices. [b]For CIFAR-10. [c]Without consideration of hardware non-idealities. [d]TOPs/W, tera operations per second per watt (energy efficiency).

neural network model parameters and hardware parameters[15–17,63]. The frameworks for a fixed architecture are designed to optimize the neural network model for a specific IMC hardware. This approach is the most widespread method employed for adjusting a neural network model for a specific ready IMC architecture considering hardware constraints[12–14,62,69,70,72,79,80,99–101]. Frameworks for IMC hardware parameters search for a fixed neural network model formulate a hardware optimization problem as a single- or multi-objective optimization problem, rather than optimizing the design manually or using brute force approaches[102]. The HW-NAS frameworks for both neural network model and IMC hardware parameters search perform co-optimization of software and hardware parameters. This approach is useful to obtain optimum hardware solutions for a particular application, especially at the initial design stages.

In addition, state-of-the-art HW-NAS frameworks can be categorized based on the algorithm used in a search (Supplementary Fig. 2d). The detailed description of each framework and mathematical representation of the problem formulation can be found in Supplementary Note 3.

An HW-NAS framework for IMC that can simultaneously prune, quantize and perform NAS in one flow has not been reported yet. The baseline and optimization functions for the state-of-the-art HW-NAS frameworks for IMC are different, and these frameworks focus on different neural network models and different search strategies (Table 1). Therefore, comparing the performance and search speed of these frameworks is difficult. It is important to note that the state-of-the-art frameworks for HW-NAS for IMC are mostly designed for different types of convolutional neural networks, and it is still an open problem to apply HW-NAS techniques to other types of neural network architectures implemented on IMC hardware.

Two-stage optimization versus joint optimization

The HW-NAS can be divided into the search of an optimized model for a specific hardware architecture taking hardware constraints into account, and co-optimization of a neural network model and hardware parameters (Supplementary Fig. 2c). The second is useful when designing IMC hardware for a specific application, especially when hardware efficiency is critical. As illustrated in Table 1, only a few HW-NAS frameworks for IMC include hardware parameters in the search and perform hardware–software co-optimization. The hardware parameter search can help to design a more efficient hardware implementation of an IMC architecture. To include IMC hardware parameters in the search, there are two possible scenarios of HW-NAS frameworks: two-stage optimization and joint optimization. Comparing it to the problem formulation techniques of HW-NAS shown in Fig. 2, two-stage optimization falls into the category of two-state methods, whereas joint optimization refers to the rest of the HW-NAS problem formulation methods.

In the two-stage optimization, a neural network model search space and hardware search space are separated. After defining the neural network model search space, the set of networks is sampled followed by HW-NAS to select a set of models with high-performance accuracy using a certain search algorithm. When the best-performing networks are selected, the set of networks is passed to the second stage of optimization. In the second optimization stage, the optimum hardware parameters are searched from the set of sampled hardware parameters. Finally, the second search stage outputs the optimum neural network model(s) and optimum hardware parameters.

In joint optimization, a large joint search space consisting of neural network models and hardware parameters is sampled to create a set of random neural network models. Then, HW-NAS is performed, searching for the optimum neural network model and hardware parameters simultaneously. Both performance accuracy and hardware metrics are used to evaluate the performance sampled networks and find the most optimum design.

Two-stage optimization can simplify the search, as the best-performing models are selected in the first stage only based on performance accuracy. This makes the search space smaller in the second stage, where hardware parameters are selected. However, this approach can lead to local optimization and might not explore the search space fully. In joint optimization, the search space is large, which can make the search slower and more complex. However, it also allows the selection of the best-performing models considering design parameters and has more probability of reaching the global solution. Also, as shown in ref. 15, there is a correlation between the hardware parameters and performance accuracy. In addition, the problem formulation methods and end goal of HW-NAS should be considered when selecting the methods to add the hardware parameters to the search.

Outlook and recommendations

Even though methods and frameworks for hardware–software co-design techniques for IMC, and HW-NAS in particular, have already been developed, there are still several open challenges in HW-NAS for IMC to be addressed. This section covers the open issues and future research directions for HW-NAS for IMC and provides recommendations for hardware evaluation techniques, mapping neural network models to hardware, and IMC system co-optimization.

Open problems and challenges in HW-NAS for IMC

A roadmap for HW-NAS for IMC architectures, including state-of-the-art frameworks, open problems and future development, is illustrated in Fig. 5. One of the main challenges is the lack of a unified framework searching for both neural network design parameters and hardware parameters. Moreover, none of the reported HW-NAS frameworks for IMC can prune, quantize and perform NAS in one flow. Combining these three optimizations in a single framework and optimizing a search time for such a large search space is an open challenge for IMC architectures. One example of a similar existing framework is APQ, which targets a constrained NAS problem but for a digital neural network accelerator[8].

Different frameworks focus on different hardware implementations and parameters, and different neural network designs (Table 1). Most of the frameworks focus only on specific issues without considering various HW aspects, such as the study of the correlation between crossbar size and convolution kernel sizes in the search engine NAX[15]. Therefore, a fair comparison between methods for HW-NAS for IMC is not possible, which leads to a lack of benchmarking of various HW-NAS techniques and search strategies. For the end user, it is still challenging to understand which search algorithm will perform better, what possible speed-up could be provided by certain algorithms, and which techniques of HW-NAS are the most efficient for IMC applications. There is a lack of quantitative comparison of HW-NAS methods, especially considering various hardware parameters in the search.

Moreover, state-of-the-art HW-NAS frameworks for IMC architectures focus mostly on different types of convolutional neural networks for computer vision applications, such as ResNet or VGG. However, there are many other neural network types to which the HW-NAS approach for IMC architectures has not yet been applied, such as transformer-based networks[103] or graph networks[104].

There are open challenges related to hardware–software co-design of such models and IMC architectures for various applications, for example biomedical tasks[105], language processing[106] or recommender systems[81].

In addition to the lack of HW-NAS frameworks for IMC focusing on diverse neural network models, the same applies to HW-NAS benchmarks. In NAS, benchmarks are the datasets describing the accuracy of all the sampled architectures in a certain search space[107]. NAS benchmarks are important to ensure reproducibility of the experiments and comparison of different methods and algorithms, avoiding extensive computations when generating a search space. These include NAS-Bench-101 (ref. 108), NAS-Bench-201 (ref. 109) and NAS-Bench-360 (ref. 110). It is also important to extend such benchmarks to the hardware domain. For example, HW-NAS benchmarks, including energy and latency, for different types of edge devices, such as mobile devices, ASIC and FPGA, have been demonstrated[86]. An HW-NAS benchmark for IMC architectures is still an open problem, which is essential to be addressed for further development of HW-NAS algorithms for IMC applications.

From the hardware perspective, most existing frameworks for HW-NAS for IMC focus on the standard mixed-signal IMC architecture[14,17,70,99], because of the availability of open-source frameworks for hardware evaluation, such as DNN+NeuroSim[87]. However, there are a lot of other IMC architectures, where different design parameters, devices and technologies are used. The main issue is the adaptation of state-of-the-art HW-NAS frameworks for the other IMC hardware architectures without implementing the frameworks from scratch.

Further development is required in hardware–software co-design techniques to transfer the neural network model from software to hardware. Techniques to speed up the hardware simulation are needed. Even though most HW-NAS frameworks for IMC use software-level simulations (such as Python or C++) to approximate and speed up simulations of circuits and architectures (as for SPICE-level simulations), further development and improvements in hardware–software co-design frameworks are required[7]. Moreover, hardware–software co-design includes IMC-related compiler optimizations, which can translate deep learning operations to IMC-friendly hardware implementations.

The complexity and runtime issues of the search algorithms should also be addressed. Although there are various methods to speed up HW-NAS, the architecture search is still complex and slow, especially when a lot of search parameters are considered. Moreover, the complexity of HW-NAS increases even further when IMC hardware search space is considered. In most of the search strategies, except the differentiable search, the search time and search space increase exponentially with the number of search parameters.

Finally, the step further is to create fully automated NAS methods capable of constructing new deep learning operations and algorithms suitable for IMC with minimal human design. One such approach for general ML algorithms and neural networks is illustrated by AutoML-Zero[111], which automatically searches for the whole ML algorithm, including the model, optimization procedures and initialization techniques, with minimum restriction on the form and only simple mathematical operations. Such algorithms aim to reduce human intervention in the design and allow constructing of an algorithm and neural network model without predefined complex building blocks, which can adapt to different tasks. Adding IMC-awareness to such methods is a step forward in fully automating IMC-based neural network hardware design.

Hardware evaluation frameworks

The frameworks for HW-NAS require the estimation of the hardware metrics, even when not searching for the hardware parameters. Several open-source hardware estimation frameworks can be used to estimate hardware metrics for standard IMC architecture, such as DNN+NeuroSim[87] and PUMAsim[45]. Both frameworks allow setting up of several computation cores (crossbars) and architecture-related parameters. NeuroSim also includes several hardware non-idealities and precision parameters. In addition, adding hardware non-idealities and noise effects to the search framework is an essential step toward developing highly effective surrogate models, as they can affect the performance

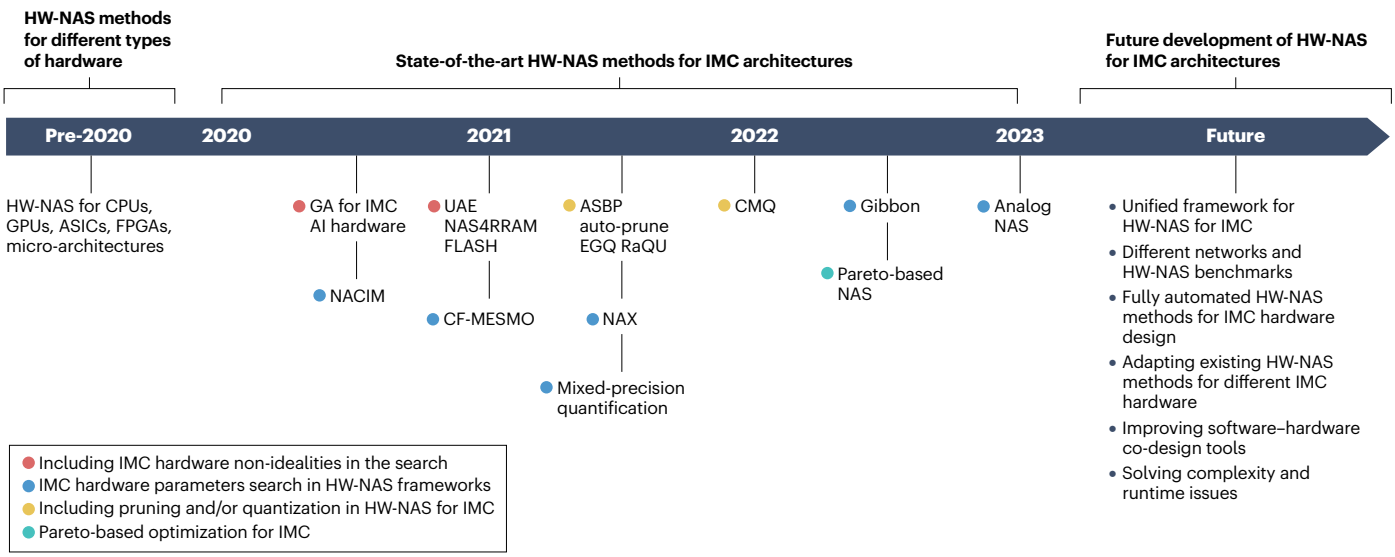

Fig. 5 | Roadmap for hardware-aware neural architecture search for in-memory computing. Summary of what has been accomplished by state-of-the-art hardware-aware neural architecture search (HW-NAS) frameworks for in-memory computing (IMC), and main perspectives and directions for future development. AI, artificial intelligence; ASIC, application-specific integrated circuit; CPU, central processing unit; FPGA, field-programmable gate array; GPU, graphical processing unit; GA, genetic algorithm.

accuracy and hardware metrics, and also can be compensated by the selected neural network model parameters[15,80,100]. One of the open-source frameworks that can be used for IMC hardware non-idealities simulations is GENIEx[112]. For the non-standard designs, there is still a lack of open-source frameworks, so designers are still required to create custom hardware evaluation frameworks for non-trivial IMC architecture or customize the existing ones. This challenge can be addressed by developing a framework generating IMC hardware description automatically from a neural network model.

Mapping deep neural network models to IMC hardware

Another drawback of the state-of-the-art HW-NAS models for IMC is the lack of consideration of dataflow optimization methods and mapping techniques of ML model to the IMC hardware. Dataflow optimization involves consideration of data movement, including transmission of inputs, outputs, intermediate values and control instructions within the IMC architecture and to the external system components, while mapping covers the split of the neural network model across the available hardware resources. In addition to the several layers of the hierarchy of the IMC architecture, including processing elements, computing elements, and tiles (Supplementary Fig. 1), which should be considered for efficient mapping, the IMC accelerator is also connected to the external memory and cache system (Supplementary Fig. 3). Global registers, SRAM global buffer, cache and DRAM are used for storing and fetching inputs, outputs and neural network weights (in the case of larger deep neural networks when all the layers cannot fit into the accelerator processing elements). Some IMC systems can also have a local cache, such as the RIMAC IMC accelerator that contains a local analog cache to reduce the overhead of ADCs and DACs[47]. The benefits of IMC — energy efficiency and low latency — can fully be exploited only if the data-path and data-mapping to the external memory are optimized[113].

Depending on the workloads and types of layers, the optimum mapping varies. For example, there are several ways to map the convolution layer to the crossbar, including dense and sparse mapping[75]. However, depth-wise separable convolution layers should be converted to a dense layer form, which yields highly inefficient performance[114]. Mapping of neural network layers to processing elements, computing elements and tiles is considered in the existing IMC hardware simulators. The design space exploration framework, for example, supports different types of mapping and investigates its effects on the RRAM-based IMC architecture performance[75]. HW-NAS frameworks based on NeuroSim[87] and AIHWKit[90] consider the mapping to the real hardware models, and NeuroSim performs the optimization of hardware utilization. Nevertheless, the types of supported layers are still limited, and the hardware architecture is fixed.

In contrast, optimization of data movement from and to the external memory, fetching and storing inputs and outputs considering different levels of the external memory hierarchy, is rarely done in the existing frameworks[113]. The trade-off between the global buffer or cache sizes and the data movement time affects IMC hardware efficiency. Spatial and temporal data reuse can also reduce latency and improve energy efficiency. Therefore, mapping ML and neural network models to IMC hardware considering the data path and the external memory based on the workloads is a separate optimization problem. One of the existing frameworks that can be used for these purposes is ZigZag[115], which aims to optimize even and uneven mapping to large balanced and unbalanced memory hierarchy design space and is suitable for IMC hardware accelerators[113].

HW-NAS and IMC co-optimization

As of the beginning of 2024, HW-NAS frameworks for IMC applications mostly cover device-related, circuit-related and algorithm-related optimizations (Fig. 6). However, to design the optimized IMC hardware, the architecture- and system-level design should also be optimized. Therefore, it is important to combine HW-NAS with other optimization techniques.

In most state-of-the-art HW-NAS optimization frameworks, the architecture is fixed, including mapping and communication between tiles. However, architecture-related details in IMC architectures should also be optimized. For example, effective on-chip communication and optimized interconnect choice are critical for IMC hardware[44].

The system-level design is the part responsible for translating the algorithm to the hardware, which also requires optimization. From the programming perspective, there are various high-level hardware–software co-design concepts requiring design and optimization, including communication of the IMC accelerator with the CPU, programming commands and instructions, issues with shared off-chip memory, and automated scheduling. These challenges related to accelerator programming are rarely considered by IMC hardware designers, even though there are many design aspects to optimize[116].

In most cases, IMC accelerators are not used as a standalone system and are considered as co-processors that need to communicate

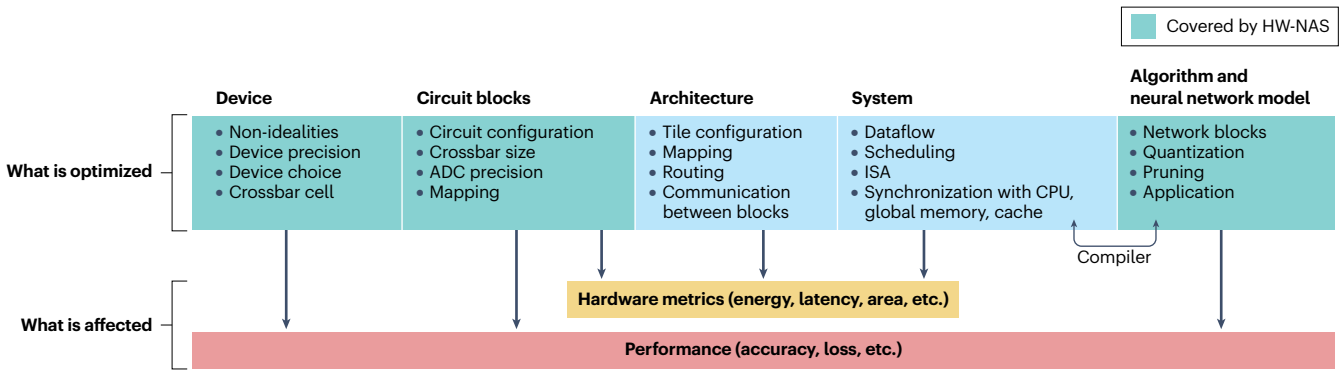

Fig. 6 | Place of hardware-aware neural architecture search in hardware–software co-design. Hardware–software co-design flow covers the optimization of devices, circuits, architectures, systems and algorithms. Hardware-aware neural architecture search (HW-NAS) can be involved in device-level, circuit-level and algorithm-level optimizations. Including optimization of architecture and system-related parameters in HW-NAS is a potential direction of future research, which could help to automate hardware–software co-design and further improve the optimized solutions.

with the host CPU. Control and instruction units on both sides, CPU and IMC accelerator, and extending instruction set architecture are thus needed. Extending instruction set architecture is an abstraction of hardware–software interface defining the main hardware operations and control commands. This interface is a key to support the implementation of different algorithms on hardware[117], including mapping different types of neural networks to IMC hardware. Creating instruction set architectures and optimizing the programming of an IMC architecture using high-level languages is important for IMC architectures[118]. Hardware compilers and automated translation of the software commands to hardware implementation are also open challenges for IMC architectures.

Often external DRAM should be shared between the CPU and IMC accelerator to support storage of large models and large tensors that do not fit on on-chip memories. This point brings in other design issues to consider and optimize, including data sharing, virtual address translation (to translate the memory address used by IMC architecture to the global DRAM address) and cache coherence (if the IMC accelerator shares the cache with the CPU). Also, as swapping and re-loading neural network weights to IMC architecture can reintroduce data movement issues, it can be more practical to split the large network models between the CPU and weight-stationary IMC architecture when it does not fit an IMC accelerator. This point might require additional optimization. Another optimization challenge is the automated runtime scheduling of IMC tasks[116]. Therefore, consideration of the higher-level programming perspective, automating and simplifying the translation of the software algorithms to hardware, is the next optimization step in IMC hardware accelerators for ML and neural network applications. In general, it is crucial to combine HW-NAS with other optimization techniques to design efficient IMC hardware for AI and ML applications.

Published online: 20 May 2024

References

1. Bengio, Y., Lecun, Y. & Hinton, G. Deep learning for AI. *Commun. ACM* **64**, 58–65 (2021).
   This work provides an overview of deep learning methods for artificial intelligence applications and related future directions.
2. Krestinskaya, O., James, A. P. & Chua, L. O. Neuromemristive circuits for edge computing: a review. *IEEE Trans. Neural Netw. Learn. Syst.* **31**, 4–23 (2019).
3. Ielmini, D. & Wong, H.-S. P. In-memory computing with resistive switching devices. *Nat. Electron.* **1**, 333–343 (2018).
4. Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R. & Eleftheriou, E. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* **15**, 529–544 (2020).
   This work explains the importance of and highlights the application landscape of in-memory computing, and also includes an overview of in-memory computing devices.
5. Xia, Q. & Yang, J. J. Memristive crossbar arrays for brain-inspired computing. *Nat. Mater.* **18**, 309–323 (2019).
6. Yang, J. J., Strukov, D. B. & Stewart, D. R. Memristive devices for computing. *Nat. Nanotechnol.* **8**, 13–24 (2013).
7. Zhang, W. et al. Neuro-inspired computing chips. *Nat. Electron.* **3**, 371–382 (2020).
   This work benchmarks in-memory computing architectures, presents the requirements for device metrics based on different applications and provides an in-memory computing roadmap.
8. Wang, T. et al. Apq: joint search for network architecture, pruning and quantization policy. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 2078–2087 (IEEE/CVF, 2020).
9. Benmeziane, H. et al. A comprehensive survey on hardware-aware neural architecture search. Preprint at https://doi.org/10.48550/arXiv.2101.09336 (2021).
10. Chitty-Venkata, K. T. & Somani, A. K. Neural architecture search survey: a hardware perspective. *ACM Comput. Surv.* **55**, 1–36 (2022).
11. Benmeziane, H. et al. Hardware-aware neural architecture search: survey and taxonomy. In *Proc. Thirtieth International Joint Conference on Artificial Intelligence* 4322–4329 (IJCAI, 2021).
12. Benmeziane, H. 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)* 233–244 (IEEE, 2023).
13. Yuan, Z. et al. NAS4RRAM: neural network architecture search for inference on RRAM-based accelerators. *Sci. China Inf. Sci.* **64**, 160407 (2021).
14. Guan, Z. et al. 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)* 1–4 (IEEE, 2022).
15. Negi, S., Chakraborty, I., Ankit, A. & Roy, K. NAX: neural architecture and memristive xbar based accelerator co-design. In *Proc. 59th ACM/IEEE Design Automation Conference* 451–456 (IEEE, 2022).
16. Sun, H. et al. Gibbon: efficient co-exploration of NN model and processing-in-memory architecture. In *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)* 867–872 (IEEE, 2022).
17. Jiang, W. et al. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Comput.* **70**, 595–605 (2020).
18. Elsken, T., Metzen, J. H. & Hutter, F. Neural architecture search: a survey. *J. Mach. Learn. Res.* **20**, 1997–2017 (2019).
19. Ren, P. et al. A comprehensive survey of neural architecture search: challenges and solutions. *ACM Comput. Surv.* **54**, 1–34 (2021).
    This work provides a survey on neural architecture search from the software, algorithms and frameworks perspective.
20. Sekanina, L. Neural architecture search and hardware accelerator co-search: a survey. *IEEE Access* **9**, 151337–151362 (2021).
21. Zhang, X., Jiang, W., Shi, Y. & Hu, J. When neural architecture search meets hardware implementation: from hardware awareness to co-design. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* 25–30 (IEEE, 2019).
22. Efnusheva, D., Cholakoska, A. & Tentov, A. A survey of different approaches for overcoming the processor-memory bottleneck. *Int. J. Comput. Sci. Inf. Technol.* **9**, 151–163 (2017).
23. Liu, B. et. al. Hardware acceleration for neuromorphic computing: An evolving view. In *15th Non-Volatile Memory Technology Symposium (NVMTS)* 1–4 (IEEE, 2015).
24. Yantır, H. E., Eltawil, A. M. & Salama, K. N. IMCA: an efficient in-memory convolution accelerator. *IEEE Trans. Very Large Scale Integr. Syst.* **29**, 447–460 (2021).
25. Fouda, M. E., Yantır, H. E., Eltawil, A. M. & Kurdahi, F. In-memory associative processors: tutorial, potential, and challenges. *IEEE Trans. Circuits Syst. II Express Briefs* **69**, 2641–2647 (2022).
26. Yantır, H. E., Eltawil, A. M. & Salama, K. N. A hardware/software co-design methodology for in-memory processors. *J. Parallel Distrib. Comput.* **161**, 63–71 (2022).
27. Lotfi-Kamran, P. et al. Scale-out processors. *ACM SIGARCH Comput. Archit. N.* **40**, 500–511 (2012).
28. Ali, M. et al. Compute-in-memory technologies and architectures for deep learning workloads. *IEEE Trans. Very Large Scale Integr. Syst.* **30**, 1615–1630 (2022).
29. Ielmini, D. & Pedretti, G. Device and circuit architectures for in-memory computing. *Adv. Intell. Syst.* **2**, 2000040 (2020).
    This work provides an extensive overview of in-memory computing devices.
30. Lanza, M. et al. Memristive technologies for data storage, computation, encryption, and radio-frequency communication. *Science* **376**, eabj9979 (2022).
    This work reviews in-memory computing devices, related computations and their applications.
31. Mannocci, P. et al. In-memory computing with emerging memory devices: status and outlook. *APL Mach. Learn.* **1**, 010902 (2023).
32. Sun, Z. et al. A full spectrum of computing-in-memory technologies. *Nat. Electron.* https://doi.org/10.1038/s41928-023-01053-4 (2023).
33. Smagulova, K., Fouda, M. E., Kurdahi, F., Salama, K. N. & Eltawil, A. Resistive neural hardware accelerators. *Proc. IEEE* **111**, 500–527 (2023).
    This work provides an overview of in-memory computing-based deep learning accelerators.
34. Rasch, M. Neural network accelerator design with resistive crossbars: opportunities and challenges. *IBM J. Res. Dev.* **63**, 10:11–10:13 (2019).
35. Ankit, A., Chakraborty, I., Agrawal, A., Ali, M. & Roy, K. Circuits and architectures for in-memory computing-based machine learning accelerators. *IEEE Micro* **40**, 8–22 (2020).
36. Gebregiorgis, A. et al. A survey on memory-centric computer architectures. *ACM J. Emerg. Technol. Comput. Syst.* **18**, 1–50 (2022).
37. Aguirre, F. et al. Hardware implementation of memristor-based artificial neural networks. *Nat. Commun.* **15**, 1974 (2024).
38. Rasch, M. J. et al. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nat. Commun.* **14**, 5282 (2023).
39. Le Gallo, M. et al. A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference. *Nat. Electron.* **6**, 680–693 (2023).
40. Fick, L., Skrzyniarz, S., Parikh, M., Henry, M. B. & Fick, D. Analog matrix processor for edge AI real-time video analytics. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)* 260–262 (IEEE, 2022).
41. Shafiee, A. et al. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Comput. Archit. Netw.* **44**, 14–26 (2016).
42. Krishnan, G. et al. SIAM: chiplet-based scalable in-memory acceleration with mesh for deep neural networks. *ACM Trans. Embedded Comput. Syst.* **20**, 1–24 (2021).
    This work provides an overview of the hierarchical system-level design of in-memory computing accelerators for deep neural networks.
43. Ankit, A. et al. Panther: a programmable architecture for neural network training harnessing energy-efficient reram. *IEEE Trans. Comput.* **69**, 1128–1142 (2020).
44. Krishnan, G. et al. Impact of on-chip interconnect on in-memory acceleration of deep neural networks. *ACM J. Emerg. Technol. Comput. Syst.* **18**, 1–22 (2021).

45. Ankit, A. et al. PUMA: a programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proc 24th International Conference on Architectural Support for Programming Languages and Operating Systems* 715–731 (ACM, 2019).
46. Li, W. et al. TIMELY: pushing data movements and interfaces in PIM accelerators towards local and in time domain. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* 832–845 (IEEE, 2020).
47. Chen, P., Wu, M., Ma, Y., Ye, L. & Huang, R. RIMAC: an array-level ADC/DAC-free ReRAM-based in-memory DNN processor with analog cache and computation. In *Proc. 28th Asia and South Pacific Design Automation Conference* 228–233 (ACM, 2023).
48. Zhang, B. et al. PIMCA: a programmable in-memory computing accelerator for energy-efficient dnn inference. *IEEE J. Solid-State Circ.* **58**, 1436–1449 (2022).
49. Kim, D. E., Ankit, A., Wang, C. & Roy, K. SAMBA: sparsity aware in-memory computing based machine learning accelerator. *IEEE Trans. Comput.* **72**, 2615–2627 (2023).
50. Jain, S. et al. A heterogeneous and programmable compute-in-memory accelerator architecture for analog-AI using dense 2-D mesh. *IEEE Trans. Very Large Scale Integr. Syst.* **31**, 114–127 (2022).
51. Wang, X. et al. TAICHI: a tiled architecture for in-memory computing and heterogeneous integration. *IEEE Trans. Circ. Syst. II Express Briefs* **69**, 559–563 (2021).
52. Wan, W. et al. A compute-in-memory chip based on resistive random-access memory. *Nature* **608**, 504–512 (2022).
53. Hung, J.-M. et al. A four-megabit compute-in-memory macro with eight-bit precision based on CMOS and resistive random-access memory for AI edge devices. *Nat. Electron.* **4**, 921–930 (2021).
54. Xue, C.-X. et al. A 22nm 4Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices. In *IEEE International Solid-State Circuits Conference (ISSCC)* 245–247 (IEEE, 2021).
55. Khwa, W.-S. et al. A 40-nm, 2M-cell, 8b-precision, hybrid SLC-MLC PCM computing-in-memory macro with 20.5-65.0 TOPS/W for tiny-Al edge devices. In *IEEE International Solid-State Circuits Conference (ISSCC)* 1–3 (IEEE, 2022).
56. Jia, H. et al. Scalable and programmable neural network inference accelerator based on in-memory computing. *IEEE J. Solid State Circuits* **57**, 198–211 (2021).
57. Jung, S. et al. A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature* **601**, 211–216 (2022).
58. Krestinskaya, O., Zhang, L. & Salama, K. N. Towards efficient RRAM-based quantized neural networks hardware: state-of-the-art and open issues. In *IEEE 22nd International Conference on Nanotechnology (NANO)* 465–468 (IEEE, 2022).
59. Joshi, V. et al. Accurate deep neural network inference using computational phase-change memory. *Nat. Commun.* **11**, 2473 (2020).
60. Cao, T. et al. A non-idealities aware software–hardware co-design framework for edge-ai deep neural network implemented on memristive crossbar. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 934–943 (2022).
61. Wen, W., Wu, C., Wang, Y., Chen, Y. & Li, H. Learning structured sparsity in deep neural networks. *Adv. Neural Inf. Proc. Syst.* **29**, 2074–2082 (2016).
62. Peng, J. et al. CMQ: crossbar-aware neural network mixed-precision quantization via differentiable architecture search. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **41**, 4124–4133 (2022).
63. Huang, S. et al. Mixed precision quantization for ReRAM-based DNN inference accelerators. In *Proc. 26th Asia and South Pacific Design Automation Conference* 372–377 (ACM, 2021).
64. Meng, F.-H., Wang, X., Wang, Z., Lee, E. Y.-J. & Lu, W. D. Exploring compute-in-memory architecture granularity for structured pruning of neural networks. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **12**, 858–866 (2022).
65. Krestinskaya, O., Zhang, L. & Salama, K. N. Towards efficient in-memory computing hardware for quantized neural networks: state-of-the-art, open challenges and perspectives. *IEEE Trans. Nanotechnol.* **22**, 377–386 (2023).
66. Li, Y., Dong, X. & Wang, W. Additive powers-of-two quantization: an efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations (ICRL)* (ICRL, 2020).
67. Karimzadeh, F., Yoon, J.-H. & Raychowdhury, A. Bits-net: bit-sparse deep neural network for energy-efficient RRAM-based compute-in-memory. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **69**, 1952–1961 (2022).
68. Yang, H., Duan, L., Chen, Y. & Li, H. BSQ: exploring bit-level sparsity for mixed-precision neural network quantization. In *International Conference on Learning Representations (ICRL)* (ICRL, 2020).
69. Qu, S. et al. RaQu: an automatic high-utilization CNN quantization and mapping framework for general-purpose RRAM accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)* 1–6 (IEEE, 2020).
70. Kang, B. et al. Genetic algorithm-based energy-aware CNN quantization for processing-in-memory architecture. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **11**, 649–662 (2021).
71. Li, S., Hanson, E., Li, H. & Chen, Y. Penni: pruned kernel sharing for efficient CNN inference. In *International Conference on Machine Learning* 5863–5873 (PMLR, 2020).
72. Yang, S. et al. AUTO-PRUNE: automated DNN pruning and mapping for ReRAM-based accelerator. In *Proc. ACM International Conference on Supercomputing* 304–315 (ACM, 2021).
73. Zhang, T. et al. Autoshrink: a topology-aware NAS for discovering efficient neural architecture. In *Proc. AAAI Conference on Artificial Intelligence* 6829–6836 (AAAI, 2020).
74. Cheng, H.-P. et al. NASGEM: neural architecture search via graph embedding method. in *Proc. AAAI Conference on Artificial Intelligence* 7090–7098 (AAAI, 2021).
75. Lammie, C. et al. Design space exploration of dense and sparse mapping schemes for RRAM architectures. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)* 1107–1111 (IEEE, 2022).
76. Fiacco, A. V. & McCormick, G. P. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques* (SIAM, 1990).
77. Lu, Z. et al. NSGA-Net: neural architecture search using multi-objective genetic algorithm. In. *Proc. Genetic and Evolutionary Computation Conference* 419–427 (ACM, 2019).
78. Guo, Y. et al. Pareto-aware neural architecture generation for diverse computational budgets. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 2247–2257 (IEEE, 2023).
79. Qu, S., Li, B., Wang, Y. & Zhang, L. ASBP: automatic structured bit-pruning for RRAM-based NN accelerator. In *2021 58th ACM/IEEE Design Automation Conference (DAC)* 745–750 (IEEE, 2021).
80. Krestinskaya, O., Salama, K. & James, A. P. Towards hardware optimal neural network selection with multi-objective genetic search. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2020).
81. Zhang, T. et al. NASRec: weight sharing neural architecture search for recommender systems. In *Proc. ACM Web Conference* 1199–1207 (ACM, 2023).
82. Stolle, K., Vogel, S., van der Sommen, F. & Sanberg, W. in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* 463–479 (Springer).
83. Li, L. & Talwalkar, A. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence* 367–377 (PMLR, 2020).
84. Huang, H., Ma, X., Erfani, S. M. & Bailey, J. Neural architecture search via combinatorial multi-armed bandit. In *2021 International Joint Conference on Neural Networks (IJCNN)* 1–8 (IEEE, 2021).
85. Liu, C.-H. et al. FOX-NAS: fast, on-device and explainable neural architecture search. In *Proc. IEEE/CVF International Conference on Computer Vision* 789–797 (IEEE, 2021).
86. Li, C. et al. HW-NAS-Bench: hardware-aware neural architecture search benchmark. In *2021 International Conference on Learning Representations (ICRL, 2021)*.
87. Peng, X., Huang, S., Luo, Y., Sun, X. & Yu, S. DNN+ NeuroSim: an end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *2019 IEEE International Electron Devices Meeting (IEDM)* 32.35.31–32.35.34 (IEEE, 2019).
88. Xia, L. et al. MNSIM: Simulation platform for memristor-based neuromorphic computing system. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **37**, 1009–1022 (2017).
89. Zhu, Z. et al. MNSIM 2.0: a behavior-level modeling tool for memristor-based neuromorphic computing systems. In *Proc. 2020 on Great Lakes Symposium on VLSI* 83–88 (ACM, 2020).
90. Rasch, M. J. et al. A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)* 1–4 (IEEE, 2021).
91. Lee, H., Lee, S., Chong, S. & Hwang, S. J. Hardware-adaptive efficient latency prediction for nas via meta-learning. *Adv. Neural Inf. Process. Syst.* **34**, 27016–27028 (2021).
92. Laube, K. A., Mutschler, M. & Zell, A. What to expect of hardware metric predictors in NAS. In *International Conference on Automated Machine Learning* 13/11–13/15 (PMLR, 2022).
93. Hu, Y., Shen, C., Yang, L., Wu, Z. & Liu, Y. A novel predictor with optimized sampling method for hardware-aware NAS. In *2022 26th International Conference on Pattern Recognition (ICPR)* 2114–2120 (IEEE, 2022).
94. Guo, Z. et al. Single path one-shot neural architecture search with uniform sampling. In *Proc. Computer Vision — ECCV 2020: 16th European Conference* Part XVI 16, 544–560 (Springer, 2020).
95. Shu, Y., Chen, Y., Dai, Z. & Low, B. K. H. Neural ensemble search via Bayesian sampling. In *38th Conference on Uncertainty in Artificial Intelligence (UAI)* 1803-1812 (PMLR, 2022).
96. Wang, D., Li, M., Gong, C. & Chandra, V. AttentiveNAS: improving neural architecture search via attentive sampling. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* 6418–6427 (IEEE, 2021).
97. Yang, Z. & Sun, Q. Efficient resource-aware neural architecture search with dynamic adaptive network sampling. In *IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2021).
98. Lyu, B. & Wen, S. TND-NAS: towards non-differentiable objectives in differentiable neural architecture search. In *Proc. 3rd International Symposium on Automation, Information and Computing* (INSTICC, 2022).
99. Li, G., Mandal, S. K., Ogras, U. Y. & Marculescu, R. FLASH: fast neural architecture search with hardware optimization. *ACM Trans. Embedded Comput. Syst.* **20**, 1–26 (2021).
100. Krestinskaya, O., Salama, K. N. & James, A. P. Automating analogue AI chip design with genetic search. *Adv. Intell. Syst.* **2**, 2000075 (2020).
101. Yan, Z., Juan, D.-C., Hu, X. S. & Shi, Y. Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search. In *Proc. 26th Asia and South Pacific Design Automation Conference* 859–864 (ACM, 2021).
102. Yang, X. et al. Multi-objective optimization of ReRAM crossbars for robust DNN inferencing under stochastic noise. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)* 1–9 (IEEE, 2021).
103. Chitty-Venkata, K. T., Emani, M., Vishwanath, V. & Somani, A. K. Neural architecture search for transformers: a survey. *IEEE Access* **10**, 108374–108412 (2022).
104. Oloulade, B. M., Gao, J., Chen, J., Lyu, T. & Al-Sabri, R. Graph neural architecture search: a survey. *Tsinghua Sci. Technol.* **27**, 692–708 (2021).

105. Al-Sabri, R., Gao, J., Chen, J., Oloulade, B. M. & Lyu, T. Multi-view graph neural architecture search for biomedical entity and relation extraction. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **20**, 1221–1233 (2022).
106. Klyuchnikov, N. et al. NAS-Bench-NLP: neural architecture search benchmark for natural language processing. *IEEE Access* **10**, 45736–45747 (2022).
107. Chitty-Venkata, K. T., Emani, M., Vishwanath, V. & Somani, A. K. Neural architecture search benchmarks: insights and survey. *IEEE Access* **11**, 25217–25236 (2023).
108. Ying, C. et al. NAS-Bench-101: towards reproducible neural architecture search. In *International Conference on Machine Learning* 7105–7114 (PMLR, 2019).
109. Dong, X. & Yang, Y. NAS-Bench-201: extending the scope of reproducible neural architecture search. In *2020 International Conference on Learning Representations (ICLR)* (ICLR, 2020).
110. Tu, R. et al. NAS-Bench-360: benchmarking neural architecture search on diverse tasks. *Adv. Neural Inf. Process. Syst.* **35**, 12380–12394 (2022).
111. Real, E., Liang, C., So, D. & Le, Q. AutoML-Zero: evolving machine learning algorithms from scratch. In *International Conference on Machine Learning* 8007–8019 (PMLR, 2020).
112. Chakraborty, I., Ali, M. F., Kim, D. E., Ankit, A. & Roy, K. GENIEx: a generalized approach to emulating non-ideality in memristive xbars using neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)* 1–6 (IEEE, 2020).
113. Houshmand, P. et al. Assessment and optimization of analog-in-memory-compute architectures for DNN processing. In *IEEE International Electron Devices Meeting* (IEEE, 2020).
114. Zhou, C. et al. ML-HW co-design of noise-robust tinyml models and always-on analog compute-in-memory edge accelerator. *IEEE Micro* **42**, 76–87 (2022).
115. Mei, L., Houshmand, P., Jain, V., Giraldo, S. & Verhelst, M. ZigZag: enlarging joint architecture-mapping design space exploration for DNN accelerators. *IEEE Trans. Comput.* **70**, 1160–1174 (2021).
116. Ghose, S., Boroumand, A., Kim, J. S., Gómez-Luna, J. & Mutlu, O. Processing-in-memory: a workload-driven perspective. *IBM J. Res. Dev.* **63**, 1–3 (2019).
117. Liu, R. et al. FeCrypto: instruction set architecture for cryptographic algorithms based on FeFET-based in-memory computing. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **42**, 2889–2902 (2023).
118. Mambu, K., Charles, H.-P. & Kooli, M. Dedicated instruction set for pattern-based data transfers: an experimental validation on systems containing in-memory computing units. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **42**, 3757–3767 (2023).
119. Jiang, N. et al. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* 86–96 (IEEE, 2013).
120. Jiang, H., Huang, S., Peng, X. & Yu, S. MINT: mixed-precision RRAM-based in-memory training architecture. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)* 1–5 (IEEE, 2020).

Acknowledgements
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.

Author contributions
O.K. researched data and wrote the article. O.K., M.E.F., K.E.M., A.S., A.M.E. and K.N.S. contributed substantially to discussion of the content. O.K., M.E.F., H.B., K.E.M., A.S., W.D.L., M.L., H.L., F.K., S.A.F., A.M.E. and K.N.S. reviewed and/or edited the manuscript before submission.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s44287-024-00052-7.

Peer review information *Nature Reviews Electrical Engineering* thanks Arun Somani and Zheyu Yan for their contribution to the peer review of this work.