Neural Evolutionary Architecture Search for Compact Printed Analog Neuromorphic Circuits

Haibin Zhao[1], Priyanjana Pal[2], Michael Hefenbrock[3], Yuhong Wang[1]
Michael Beigl[1], and Mehdi B. Tahoori[2], *Fellow, IEEE*

***Abstract*—Printed electronics (PE) is an additive fabrication technology for manufacturing electronic circuits which not only allows for a highly flexible printing of arbitrary circuit patterns, but also produce soft, non-toxic, and degradable electronics at an extremely low cost. These properties are unmatched by silicon-based electronics, making PE an enabler of new application domains, e.g., fast moving consumer goods, wearables, and disposable healthcare devices. A particularly promising class of circuits in this technology is the printed analog neuromorphic circuits, offering efficient and highly tailored computational functionalities. In this work, we leverage the highly flexible fabrication process of PE to address the bottleneck of PE, i.e., the large feature sizes and low device counts. This issue is crucial, as it impairs the integration of printed circuits into target applications with limited footprint, such as smart band-aids. We also propose an evolutionary algorithm (EA) to improve the circuit compactness through circuit architecture optimization. As baseline, we compare the proposed EA method with a state-of-the-art pruning method and a modified area-aware pruning method. All of them are able to optimize circuit architecture alongside the component values of printed neuromorphic circuits. Experimental simulation reveals that the proposed EA approach can effectively achieve compact circuits and outperform the pruning method by $3.1\times$ lower area with no loss of accuracy. As a byproduct, the power is reduced by $3.0\times$, paving the way to energy-harvested printed systems.**

***Index Terms*—printed electronics, neuromorphic computing, compact circuit design, evolutionary algorithm, gradient-based optimization, machine learning, neural architecture search**

I. Introduction

With the progression of the Internet of Things (IoT), next-generation electronics, such as wearable [1] and disposable [2] devices (as illustrated in Fig. 1), exhibit an imperative demand for soft, bio-compatible, and environmental friendly electronic solutions. In this regard, printed electronics (PE) is emerging as one of the most promising enablers [3]. Characterized by its additive manufacturing, PE facilitates the production of soft [4], non-toxic [5], and biodegradable [6] electronics at a significantly lower cost compared to silicon-based electronics. Moreover, additive manufacturing can easily adapt to any changes in the printed pattern for specific purpose. These features are unmatched by photolithography-based subtractive manufacturing processes.

Despite their benefits, PE has its inherent weaknesses, including larger feature sizes, reduced integration density, and diminished performance compared to their silicon-based counterparts. To address these, analog printed neuromorphic circuits (pNCs) in PE have emerged as a promising computing paradigm by enabling computation directly in analog domain, thus eliminating the costly analog-digital converters (ADCs) and consequently reducing device counts.

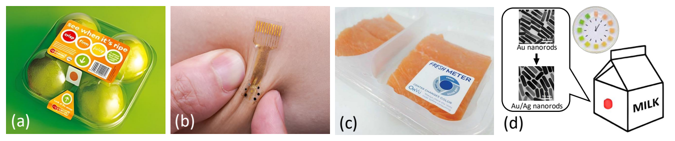


Fig. 1. Exemplary target applications domains of printed electronics: (a) smart fruit package (reprinted with permission from RipeSense®), (b) smart bandages (reprinted from [1] with permission under CC BY 4.0 license), (c) smart fish package (reprinted from [7] with permission under CC BY 4.0 license), (d) smart milk carton (adapted with permission from [8], copyright 2013 American Chemical Society).

Neuromorphic computing has shown remarkable computational abilities by leveraging basic operations like weighted sums and nonlinear activations within artificial neural networks (ANNs) or multilayer perceptrons (MLPs) [9]. These operations use analog circuit designs, like resistor crossbars and inverter circuits, thereby addressingtraditional PE limitations and open avenues to efficient computing in the analog domain for target applications.

As PE offers high flexibility, it can easily adapt to arbitrary circuit patterns though e.g., adaption of the printing trajectory, without additional cost. This allows highly bespoke printing for each specific task. So, a training framework [10] was proposed to train pNCs in a bespoke manner. Unfortunately, the existing gradient-based training method for pNCs prioritizes parameter optimization, such as crossbar conductances for weights and biases, while largely overlooking the exploration of neural architectures, specifically circuit topologies.

This drawback strongly limits the highly flexible and agile manufacturing of PE to print pNCs with bespoke neural architectures, and further limits its performance from the following aspects:

- In terms of computing functions, architecture search can extend the search space of the networks, e.g., the number of neurons and their connections. In this way, bespoke architecture can further improve the circuit performance.
- In application scenarios with restricted areas, such as smart band-aids [11] or compact smart packaging [12], the circuit footprint must be considered. Since they are strongly related to circuit architecture, architectural optimization is required.

The limitations in gradient-based training for neural architecture search (NAS) stem from the discrete nature of de-

[1]Chair for Pervasive Computing Systems, Institute of Telematics, Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany.
[2]Chair of Dependable Nano Computing, Institute of Computer Engineering, Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany.
[3]RevoAI GmbH, Germany.
Manuscript received July 05, 2024.

cision variables, yielding non-informative gradients. Inspired by *NeuroEvolution of Augmenting Topologies (NEAT)* [13], we propose an evolutionary algorithm (EA)to train network parameters and search for optimal circuit architectures. We benchmark state-of-the-art gradient-based pruning methods to reduce circuit areas by pruning neurons and weights. Additionally, to examine gradient-based architectural design, we incorporate an area-aware training objective. We conduct experiments on 13 benchmark datasets to investigate each method's efficiency and effectiveness.

According to our SPICE simulations, EA provide $3.1\times$ and $3.0\times$ less area and power with no loss of accuracy. If 10% accuracy reduction (w.r.t. baseline) is acceptable, EA provide $5.8\times$ and $12.4\times$ less area and power respectively, which is $2.9\times$ more effective than gradient-based pruning. Meanwhile, to achieve this effectiveness, the EA method requires about few $(0.5\,\text{-}\,3)$ hours.

In sum, the contributions of this work are as follows:

- Through strategic encoding, we introduce an EA capable of training both circuit parameters and architectures of pNCs. This algorithm is likely to encourage the trained pNCs having high compactness.
- We analyze the state-of-the-art neural network compression techniques and employ an existing pruning method as baseline to compare with the proposed EA method. We further enhanced this baseline by utilizing a compactness-oriented objective function, thereby serving as another stronger benchmark pruning method.
- We established a variational autoencoder (VAE)-based area estimator that supports precise calculation of the area footprint of pNCs from their architectures. This area estimator is then used to enable the area-aware training of the pNC.
- Experiments on 13 benchmark datasets are conducted, with 50 different trade-off configurations between area and classification accuracy being evaluated for each dataset. This process is repeated with 10 different random seeds to account for the random process during training. Finally, Pareto-fronts are drawn to show the superiority of the EA over the gradient-based pruning methods.

The rest of this paper is structured as follows: Sec. II introduces PE, pNC, and other preliminaries. Sec. III describes the formulation of the EA method for training compact pNCs, while Sec. IV introduces the baselines of this work, i.e., the gradient-based network pruning and its adaption to the aware-aware training objective. In Sec. VI, the proposed approaches are evaluated on the training of compact pNCs. Finally, Sec. VII concludes this work and discusses future work.

II. Preliminaries

A. Printed Electronics

Printed electronics (PE) is an emerging technology for the next-generation of electronics. Unlike conventional CMOS, which involves complex silicon-based fabrication, PE utilizes additive manufacturing to directly print electronic devices on flexible substrates, employing materials like conductive inks and semiconducting polymers. Thus, PE find applications in lightweight flexible electronics such as wearable sensors and displays, whereflexibility and ultra-low-cost production are crucial. Moreover, PE supports a variety of substrates, allowing for characteristics like flexibility and biocompatibility. These distinct benefits are essential for advancing IoT applications, such as wearables, Radio-Frequency Identification (RFID) systems, disposable electronics, and implantable sensors.

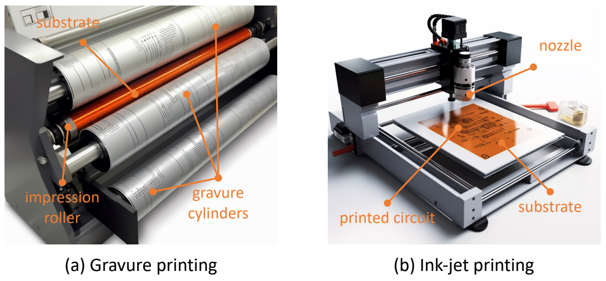


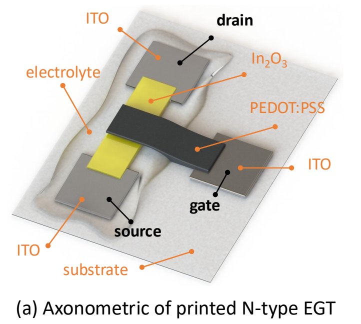
Fig. 2. Typical printing technologies: (a) gravure based high-volume printing, and (b) ink-jet based low-volume printing.

Most advancements in ink-jet printed field-effect transistors (FETs) use organic materials, forming channels between source and drain electrodes through lithographically structured organic semiconductors. Such organic FETs (OFETs), primarily based on P-type materials, exhibit low field-effect mobility and require high operation voltages ($\geq 25\,\mathrm{V}$), thus limiting their use in the target application areas of PE powered by small energy harvesters or power sources [14]. Although organic materials offer ease of processing, they also exhibit lower environmental stability. Conversely, oxide-based inorganic inks boast superior conductivity and environmental resilience but present challenges in printing and are prone to impurities attributed to surfactants [3]. Current research within inorganic PE focuses on via ink-jet printing techniques (as shown in Fig. 3), due to the absence of reliable P-type EGTs. This preference may be attributed to the band structure of metallic oxides, which favor higher field-effect mobility for electrons. Due to their substantial gate capacitance, nEGTs can operate at supply voltages below $1\,\mathrm{V}$, rendering them ideally suited for target applications.

While best performance in PE is typically achieved with vacuum-deposited, highly purified molecular substrates, solution-based fabrication methods such as spin-coating and ink-jet printing offer simpler and more cost-effective manufacturing. Printing technologies can be categorized based on scale of production: (Fig. 2a) replication printing (e.g., gravure printing for high-volume output, and (Fig. 2b) jet printing(e.g., ink-jet printing) for highly customized circuits in smaller volumes.

Furthermore, the use of contact-less printing methods, such as ink-jet printers, in conjunction with optimized functional inks comprising conductive, semiconductive, and non-conductive materials, facilitates the development of electronics on flexible substrates, supporting both organic [15] or oxide-based inorganic [16] transistors. This versatility allows printed electronics to cater to various manufacturing needs.

Nevertheless, additive manufacturing also brings drawbacks into PE, i.e., large feature sizes and high parasitic capacitance, which lead to low functional densities and high device latencies. Therefore, the target of PE is not to compete with



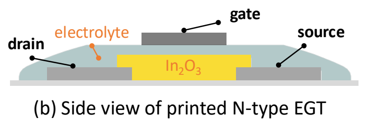


Fig. 3. Structure of the printed N-type inorganic electrolyte-gated transistor (EGT): (a) axonometric view, and (b) side view.

silicon-based electronics in the VLSI applications, but rather to supplement them in the realm of resource-limited edge computing scenarios as exemplified in Fig. 1. This favors low-complexity circuit designs with a restricted transistor count, opting preferably for analog components to reduce area utilization. This is also justified, because in the target scenarios of PE, computing tasks have low complexity and modest computational precision requirements, requiring only small-scale circuits with fewer device counts.

B. Printed Analog Neuromorphic Circuits

To address fundamental computational demands in the target applications of PE, such as fruit ripening assessment or human stress detection [17], pNCs have received increasing attention, as they can implement the operations in neuromorphic computing paradigms through the simple-structured circuit primitives, and possess advanced expressiveness.

a) Neuromorphic computing: The concept of neuromorphic computing, as envisioned by pioneers like Alan Turing [18] and John von Neumann [19], involves neurally inspired hardware implementations [9]. Unlike the classic von Neumann architecture [20], which separates CPU and memory, neuromorphic systems collocate memory and processing units. This design alleviates the CPU-memory bandwidth bottleneck [21], enabling parallel processing and energy efficiency [9]. As a result, neuromorphic systems outperform von Neumann architectures in fields such as neuroscience [22] and machine learning (ML) acceleration [23].

Although the most initial intention of neuromorphic computing was to emulate biological nervous systems, the progression of artificial intelligence (AI) has proven the excellent power of ANNs and MLPs. These models are originated from the McCulloch-Pitts neuromorphic model [24]. Consequently, research on neuromorphic computing for feedforward MLPs has outpaced that of the biological plausible paradigms, such as spiking neural networks (SNNs) (reported in 7th figure of [9]). To follow this major trend, this work focuses on the printed neuromorphic circuits implementing MLP paradigm.

b) Circuit primitives of analog pNCs: The left part of Fig. 4(b) sketches a 3-input resistor crossbar employed to emulate the weighted-sum operation in ANNs. Following Ohm's Law and Kirchhoff's Laws, the resulting output voltage of the crossbar V_z can be calculated as

$$V_\mathrm{z} = \frac{g_1}{G}V_1 + \frac{g_2}{G}V_2 + \frac{g_3}{G}V_3 + \frac{g_\mathrm{b}}{G}V_\mathrm{b}, \tag{1}$$

where g_i signifies the conductances of the resistors R_i and G represents the aggregate conductance $\sum_j g_j + g_\mathrm{b} + g_\mathrm{d}$. Evidently, the output voltage V_z is the weighted-sum of the input voltages V_i, with the weights embodied by the ratio of conductance values between g_i, g_b and G. Therefore, analogous to ANNs, by training and printing bespoke conductances, desired weights can be achieved to perform target tasks.

In pNCs, weights represented by conductance are limited to positive values, challenging the expression of negative relationships in neuromorphic computing. To overcome this drawback, inverter-based *negation circuits* have been proposed [25], as shown in Fig. 4(c). Whenever a negative weight is required, the corresponding input voltage is inverted through a negation circuit with the transfer characteristic

$$\mathrm{neg}(V_\mathrm{z}) = -\left(\eta_1^\mathrm{N} + \eta_2^\mathrm{N} \cdot \tanh\left(\left(V_\mathrm{z} - \eta_3^\mathrm{N}\right) \cdot \eta_4^\mathrm{N}\right)\right),$$

where $[\eta_1^\mathrm{N}, \eta_2^\mathrm{N}, \eta_3^\mathrm{N}, \eta_4^\mathrm{N}] = [-0.006, 1.024, 0.016, 1.006]$ are obtained by fitting the circuit characteristics based on experimental measurements [26].

Additionally, resistor crossbars constrain weights and their sum below 1, thus degrading the output. This is mitigated by connecting crossbars to an inverter-based printed tanh-like (ptanh) circuit with amplification [26] (as shown in Fig. 4b). Its transfer characteristic is given by

$$V_\mathrm{a} = \mathrm{ptanh}(V) = \eta_1^\mathrm{A} + \eta_2^\mathrm{A} \cdot \tanh\left(\left(V - \eta_3^\mathrm{A}\right) \cdot \eta_4^\mathrm{A}\right) \tag{2}$$

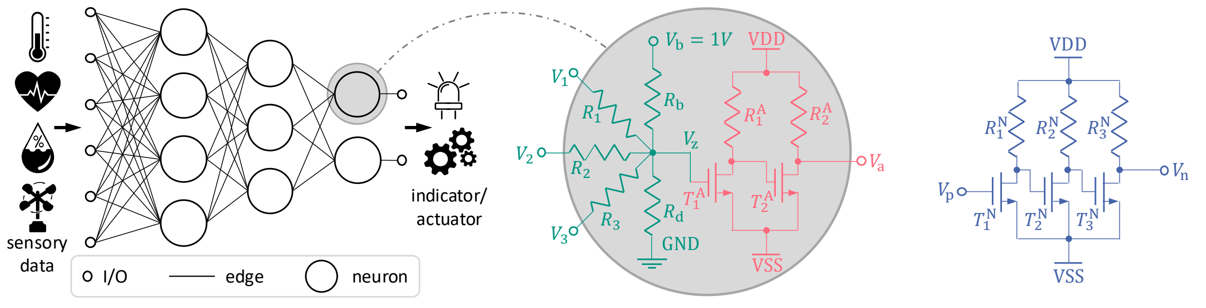
with $[\eta_1^\mathrm{A}, \eta_2^\mathrm{A}, \eta_3^\mathrm{A}, \eta_4^\mathrm{A}] = [0.29, 0.71, -0.017, 20]$. The ptanh circuit also emulates the nonlinear activation functions employed in ANNs.

Analogous to MLP, through the interconnection of the aforementioned circuit primitives, circuits with more complicated architectures can be realized to execute more complex computational tasks. Fig. 4(a) exemplifies a neuromorphic circuit resembling an ANN with a 6-4-3-2 architecture.

Furthermore, to respect the variation (e.g., printing errors) sensitivity of analog computing paradigms, variation-aware training has already been proposed to improve the robustness of pNCs [10, 26]. Nevertheless, as the study on variation-aware training is independent of the contribution in this work, we do not include variation-aware training in this work.

C. Algorithmic Level Design and Optimization

The additive manufacturing process can easily allow PE to flexibly produce the desired component values and circuit architectures through specification of their printing trajectories. Therefore, by printing appropriate conductance values in the resistor crossbar, pNCs can perform intended computational tasks. To leverage this capability, it is imperative to model the circuit for training conductance values. Given the target application of PE such as the disposable electronics in Fig. 1, pNCs do not employ reconfigurable components. Rather, they are designed in a bespoke manner and then printed.



(a) exemplary printed neuromorphic circuit (b) printed neuron (c) printed negation circuit

Fig. 4. Schematic of a printed neuromorphic circuit (pNC). (a) Exemplary analog pNC with topology being 6-4-3-2, receiving sensor signals and yields outputs to subsequent devices. (b) Left: a 3-input printed resistor crossbar. Right: schematic of an inverter-based printed tanh-like (ptanh) activation circuit. (c) Schematic of an inverter-based printed negation circuit.

a) Modeling of Printed Neuromorphic Circuit: Based on printed neurons described in Sec. II-B, the mathematical model of a printed neuron is given by

$$\text{ptanh}\Big(\sum_i w_i \left(V_i \cdot \mathbb{1}_{\{\theta_i \geq 0\}} + \text{neg}(V_i) \cdot \mathbb{1}_{\{\theta_i < 0\}}\right)\Big), \quad (3)$$

where θ_i is the learnable parameter encoding conductance by $g_i = |\theta_i|$ and the presence of a *negation circuit* via its sign. Moreover, the indicator function $\mathbb{1}_{\{\cdot\}}$ returns 1 if its condition is true, else 0. The weights w_i are given by

$$w_i = \frac{|\theta_i|}{\sum_j |\theta_j|}.$$

With this mathematical model, the conductance parameters in pNC is trained to perform specific tasks in a bespoke manner.

b) Gradient-Based Optimization: Gradient-based learning with backpropagation [27] forms the backbone of training modern ANNs. However, gradient-based training is unable to handle non-differentiable problems, e.g., the hyperparameters related to neural architecture. Additionally, some functions do not provide useful update information through their gradient, e.g., piece-wise constant functions. ThoughNAS has received increasing attention, existing methods [28, 29] may be beneficial for optimizing circuit architecture but have not been adapted for pNCs yet. This work bridges this gap by proposing a gradient-based area-aware training framework in Sec. IV.

c) Evolutionary-Based Optimization: EAs are inspired by the natural selection and biological evolution [30]. By leveraging operations like crossover and mutation, the solutions are optimized incrementally. The distinct advantage of EA lies in their versatility and adaptability, e.g., it does not necessitate a problem to be differentiable. Although EAs are generally less efficient than gradient-based methods, especially for large scale problems, this drawback is mitigated by the fact that PE generally targets small-scale circuits in edge scenarios. This work introduced an EA capable of simultaneously training both circuit parameters and architectures through strategic encoding, and showcases its superiority over state-of-the-art gradient-based methods within the context of parameter-architecture co-design of analog pNCs.

D. Related Work

Compact circuit design not only reduces the circuit costs via less material and printing time, but also opens up the scope for area-scarce scenarios. Efforts to reduce device counts include employing simpler computing paradigms like support vector machine (SVM) [31] or approximate computing [32, 33].

Neural network compression [34], including NAS [35], parameter quantization [36], and network pruning [28], may also aid in compact designs. Nevertheless, NAS primarily suits block-based deep neural networks (DNNs) like convolutional neural networks (CNNs), and is less suitable for MLP-like architectures. Consequently, network pruning emerges as a viable option for producing lightweight and compact pNCs. Therefore, we adopt an existing and a modified task-specific pruning method to investigate the potential of the gradient-based training in compact circuit design. They are then compared with the proposed EA method to serve as a benchmark.

Moreover, several efforts have leveraged EAs for training neural networks or neuromorphic circuits, as these algorithms facilitate not only the optimization of weights [37], but also the discrete decision variables like neural architectures. Notable examples include the NEAT [13] and EONS [38]. Although extensive work exists on employing EAs for network training, these efforts have largely focused on hardware-agnostic models [39, 40], digital circuits [38], or specific hardware like FPGA [41]. Nevertheless, applying these frameworks directly to pNCs is challenging, as they typically fail to account for the unique characteristics of PE, such as interference between analog circuits, device variation, aging, and other manufacturing constraints.

III. Evolutionary Architecture Search

To enable effective training for compact pNCs, it is imperative to explicitly incorporate the circuit area into the training objective. In this work, the circuit area A is estimated through its components by $A = f(N_i, A_i)$, where $A_i \in \mathbb{R}^+$ denote the area of the individual devices, e.g., A_R denotes the area of a <u>r</u>esistor, and $N_i \in \mathbb{N}$ denote their counts. Here, $f(\cdot)$ describes the area relationship between individual components and the overall circuit. Since N_i is an integer variable related to circuit topology, considering the area in the training objective necessitates training algorithms that are capable for discrete optimization. To this end, we propose an EA-based method for the simultaneous training of both crossbar conductances (weights) and circuit topologies (neural architecture).

The core parts of the proposed approach for training pNCs is shown in Fig. 5, involving *genes* that encode the circuit parameters and *genomes* that compose of multiple structured genes to represent the structure of pNCs. They are optimized through the crossover and mutation during their evolution.

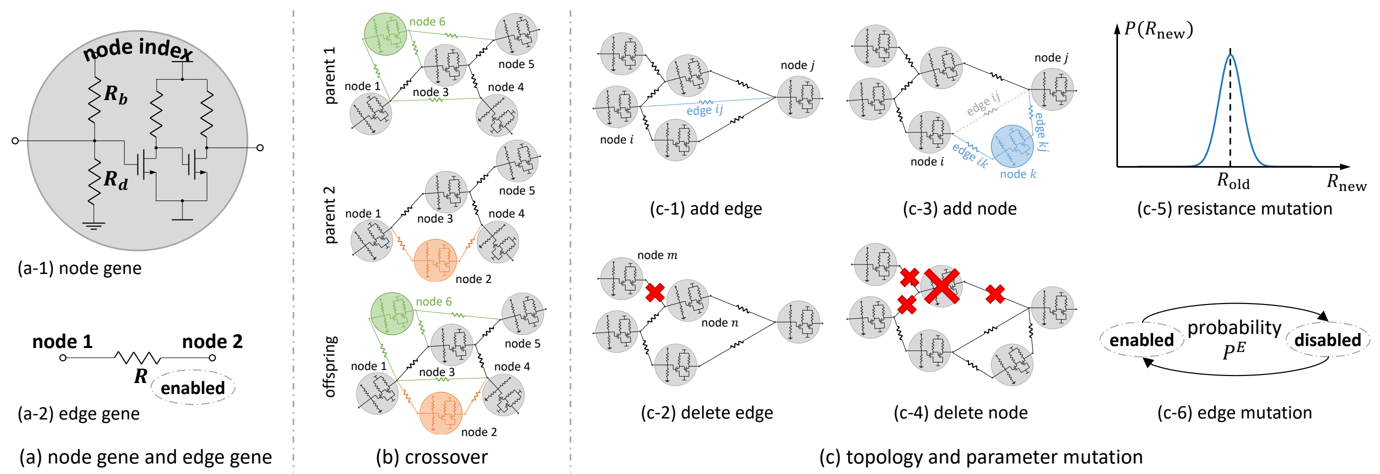


Fig. 5. Overview of the EA-based training of printed neuromorphic circuits (pNCs). (a) Genes that encode nodes and edges. (b) Crossover from the parent genomes to offsprings. (c) Mutation of the topology and learnable parameters.

A. *Encoding*

The algorithm involves two gene types, namely **node genes** indicating neurons and **edge genes** denoting connections between neurons. As shown in Fig. 5(a-1), every node gene holds a unique, fixed, and global index for identification. Moreover, each **node gene** includes an R_{b} and an R_{d} as learnable parameters (for weights and biases) followed by a fixed ptanh circuit as shown in Fig. 4. These learnable parameters will mutate from generation to generation. Meanwhile, there are **edge genes** identified by the indices of the connected nodes, as shown in Fig. 5(a-2), which are directional. Each edge gene contains a learnable resistance R, indicating the crossbar resistance for weights, and a learnable boolean parameter that indicates the state (enabled/disabled) for circuit connectivity (architecture). Similarly, these learnable parameters mutate during evolution.

Several node genes and their connections, i.e., edge genes, form a structured network that represents a pNC. In this work, such a set of genes that represents a pNC is referred to as a **genome**. Afterwards, a group of genomes can further form a **population**. We use 🧬 to denote the set of genes of all genomes involved in a population $\mathcal{P}$.

B. *Evolution*

In a population, the genomes are segregated into multiple **species** based on their similarities during the evolution. Each species undergoes origination, reproduction, and sometimes extinction. Here, in reproduction, well-performed genomes will crossover to produce offsprings and their genes will mutate. In this way, the fitness of the genomes will be gradually optimized, until a certain stop criterion is reached. The overview of this process is illustrated in Algorithm 1, the detailed python implementation is available at our GitHub repository (given in Sec. VI).

a) Speciation: To protect novel genomes from immediate extinction without being evolved, the genomes within the population $\mathcal{P}$ are grouped into multiple species $\mathcal{S}$ based on their similarity. This speciation allows each species s to develop without impact from other species. Here, the similarity is determined by both common structures (exemplified as the gray part in Fig. 5(b) in parent 1 and 2) and distinctive structures (exemplified as the green and orange parts in Fig. 5(b) in parent 1 and 2). The distance of the former is quantified by the absolute difference in their learnable parameters, whereas the distance of the latter is measured by the absolute value of their parameters. For boolean variables, the distance between $True$ and $False$ is set to 1.

b) Extinction and Selection: After speciation, the set of fitness $\mathcal{F}$ of each genome is evaluated by the objective $\mathcal{O}(\cdot)$. Meanwhile, the average fitness within a species is used to represent the fitness of each species F_s and is summarized in $\mathcal{F}_S$ for all species. If the fitness F_s of a certain species does not show improvement over $\mathcal{K}_1$ generations, it will be extinct unless it is one of the best $\mathcal{K}_2$ species. Subsequently, the top $\mathcal{K}$ genomes with respect to their fitness in each species are chosen as parents $P_s = \{p_1, \cdots, p_{\mathcal{K}}\}$ for crossover.

c) Crossover: Crossover refers to producing offsprings o that randomly inherits the genes from its parental genomes. This is a crucial process in producing evolved offspring while preserving well-performed structures. In this work, each offspring is produced from the crossover between two parents randomly selected from the parent candidates P_s. For common architectures in both parents (illustrated in Fig. 5(b) as the gray part), the offspring inherits these structures directly, and the parameters for these structures are chosen from one of the parent based on a probability proportional to their fitness ratio. As for the distinct structures (depicted in Fig. 5(b) as the green and orange parts), they are directly passed on to the offspring.

d) Mutation: Mutation is another primary method for introducing new circuit architectures and serves as the essential source for circuit parameter evolution. As shown in Fig. 5(c), in this work, mutation is a two-stage process consisting of genome-level mutation (targeting on neural architecture) and gene-level mutation (primarily for network parameters).

At the **genome-level**, mutation can either add or delete an edge between two existing nodes. Analogously, nodes can be added or deleted at an existing edge. Importantly, to prevent the extinction of newly mutated genomes, the structural changes introduced by mutation should not substantially affect the genome fitness. Therefore, to maintain the unchanged circuit output (thus fitness), when adding an edge in between two nodes, as shown in Fig. 5(c-1), the conductance of the new edge should be initialized to zero and be optimized during

Algorithm 1: Evolutionary Algorithm

```
Data: $\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$
Input: $N$: population size,
       $\mathcal{O}(\cdot)$: objective function,
       $\mathrm{selection}(\cdot)$: parents selection function,
       $\mathrm{adjust}(\cdot)$: species population size calculator,
       $\mathcal{K}$: number of candidate parents for crossover,
       $\mathcal{K}_1$: patience for species improvement,
       $\mathcal{K}_2$: number of protected species,
       $\mathcal{K}_3$: patience for evolution
Init  : population $\mathcal{P} \leftarrow N$ genomes,
       $\mathrm{stop} \leftarrow False$,
       set of species $\mathcal{S} \leftarrow \varnothing$,
       set of species population size $\mathcal{N}_S \leftarrow \varnothing$,
       set of genome fitness $\mathcal{F} \leftarrow \varnothing$,
       set of species fitness $\mathcal{F}_S \leftarrow \varnothing$
while not stop do
    $\mathcal{S} \leftarrow \mathrm{speciation}(\mathcal{P})$
    $\mathcal{F}, \mathcal{F}_S \leftarrow \mathcal{O}(\mathcal{S}, \mathcal{D})$
    $\mathcal{N}_S \leftarrow \mathrm{adjust}(\mathcal{F}_s)$
    for $s$ in $\mathcal{S}$ do
        if $F_s$ not improved for $\mathcal{K}_1$ generations then
            if $s$ not the best $\mathcal{K}_2$ species then
                $s$ extinct
            end
        else
            for $n$ in $\{1, 2, \cdots, N_s\}$ do
                $p_1, p_2 \leftarrow \mathrm{selection}(s, \mathcal{K})$
                $o_n \leftarrow \mathrm{crossover}(p_1, p_2)$
                $o_n \leftarrow \mathrm{mutation}(o_n)$
            end
        end
        $s \leftarrow \{o_1, o_2, \cdots, o_{N_s}\}$
    end
    if $F$ not improved for $\mathcal{K}_3$ generations then
        $\mathrm{stop} \leftarrow True$
    end
end
```

evolution.

Additionally, when adding a node to an edge, the existing edge will be disabled (not deleted) and the new node is introduced with two connections to replace said edge, as shown in Fig. 5(c-3). To preserve the output, the conductance on edge (k, j) should be initialized by that of the edge (i, j), whereas the output of the node k should be the same as that of node i, i.e.,

$$\mathrm{ptanh}\left(\frac{g^{(i,k)}}{g^{(i,k)} + g_{\mathrm{b}}^k + g_{\mathrm{d}}^k} V_{\mathrm{a}}^i + \frac{g_{\mathrm{b}}^k}{g^{(i,k)} + g_{\mathrm{b}}^k + g_{\mathrm{d}}^k} V_{\mathrm{b}}\right) = V_{\mathrm{a}}^i,$$

with the superscript denoting the gene index. For simplicity, we always initialize $g_{\mathrm{b}} = 0$ and $g_{\mathrm{d}} = 1$ for new nodes. Hence,

$$\eta_1^{\mathrm{A}} + \eta_2^{\mathrm{A}} \cdot \tanh\left(\left(\frac{g^{(i,k)}}{g^{(i,k)} + 1} V_{\mathrm{a}}^i - \eta_3^{\mathrm{A}}\right) \cdot \eta_4^{\mathrm{A}}\right) = V_{\mathrm{a}}^i.$$

Finally, the conductance on edge (i, k) is initialized to

$$g^{(i,k)} = \frac{M}{1 - M},$$

with

$$M = \frac{1}{\eta_4^{\mathrm{A}} V_{\mathrm{a}}^i} \tanh^{-1}\left(\frac{V_{\mathrm{a}}^i - \eta_1^{\mathrm{A}}}{\eta_2^{\mathrm{A}}}\right) + \frac{\eta_3^{\mathrm{A}}}{V_{\mathrm{a}}^i}.$$

Note that, according to Eq. (2), M is always real-valued. Furthermore, the algorithm is only negligible affected even if $M \approx 1$ or $V_{\mathrm{a}}^i \approx 0$, since g will not approach ∞ but is bounded by the range of printable conductances.

In contrast, the **gene-level** mutation is targeting to mutate circuit parameters (i.e., crossbar resistances) and is realized by a perturbation of the old values. This is implemented by adding a scaled sample from a standard normal distribution to the current resistance value, as shown in Fig. 5(c-5). Additionally, the state parameter of the edge (i.e., enabled/disabled) is mutated through a variable drawn from the Bernoulli distribution, as shown in Fig. 5(c-6).

e) Objective: The training objective, i.e., the fitness function, considers both circuit area and classification accuracy. The former is assessed by counting the total number of nodes and edges and multiplied by their respective area. The latter is designed as a combination of the classification accuracy (ACC) and the cross-entropy (CE) loss function, which is a smooth and convex surrogate function for classification accuracy [42]. Although EA enables to directly employ accuracy (i.e., the actual classification metric) as the training target, integrating cross-entropy can offer smoother guidance during evolution and provides fine-grained feedback on improvements. This becomes particularly valuable when assessing minor perturbations in resistance values in gene mutations. Consequently, the combined metric for classification accuracy is

$$\mathcal{O}(\boldsymbol{x}, \boldsymbol{y}, \text{🧬}) = \mathrm{CE}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x}, \text{🧬})\right) - \mathrm{ACC}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x}, \text{🧬})\right), \tag{4}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ are training examples provided by the target dataset, while $\hat{\boldsymbol{y}}(\boldsymbol{x}, \text{🧬})$ denotes the output of the genome. Then, the overall training objective, i.e., the fitness function, is given by

$$\underset{\text{🧬}}{\mathrm{minimize}}\ (1 - \gamma)\mathcal{O}(\boldsymbol{x}, \boldsymbol{y}, \text{🧬}) + \gamma \frac{A(\text{🧬})}{A'}, \tag{5}$$

where $\gamma \in \mathbb{R}^+$ expresses the balance between accuracy and area, and A' is a constant multiplier to calibrate the of area term to the similar magnitude of accuracy term $\mathcal{O}(\boldsymbol{x}, \boldsymbol{y}, \text{🧬})$.

At the beginning of evolution, the EA starts with only output nodes. From there, it progressively increases the number of neurons and their connectivity. Over successive generational iterations, genome fitness improves progressively. Upon reaching the stop criterion (with which the genomes are sufficiently optimized), the associated topological structures and parameters can be mapped to the respective hardware primitives and fabricated.

IV. Baseline Methods

This section introduces the baseline we employed in our experiment. Different from EA that starts from a minimal architecture and gradually grows the network size, as the counterpart of EA, modern machine learning models are

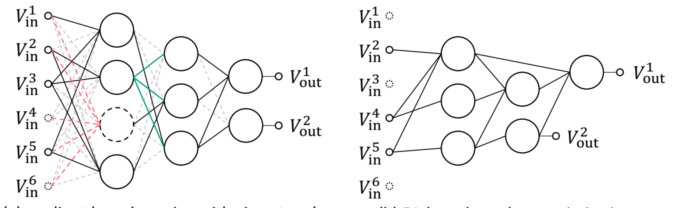


Fig. 6. Comparison of the circuit architecture from (a) gradient and (b) evolutionary approaches. In (a), the dashed edges and nodes are pruned. The red color refers to pruning a neuron when all its input weights are pruned. The green color represents the case of pruning a negation circuit when all the weights associated to a voltage are positive.

typically trained through gradient-based techniques. Gradient-based training usually provides much higher efficiency through backpropagation, but are generally less effective in topological optimization. To explore the effectiveness of the gradient approach in pNC architecture and to provide a strong benchmark for evaluating the effectiveness of the proposed EA, we utilize a state-of-the-art pruning method. In addition, we introduce a task-specific *area-aware training* approach to especially encourage pNCs with lower area footprint. These approaches start with a large and predefined circuit architecture, progressively removes circuit components, and finally results in an area-efficient circuit architecture.

A. Cutting Edge Approaches

Gradient-based methods are more efficient than EAs in terms of training time, however, they are often inadequate to optimize network architectures because the architectures are usually discrete variables that do not produce an informative gradient to guide the training process. State-of-the-art gradient-based strategies for optimizing network architectures are *NAS* [43] and *network pruning* [28]. Unfortunately, NAS approaches are mainly designed for DNNs with block structures [44]–[47] such as residual blocks with different kernel sizes or long-short-term-memory (LSTM) blocks, and they require hand-crafted architectures. For instance, in *differentiable architecture search* (DARTS) [29], several convolutional kernels with different sizes are pre-designed as candidates, and finally, the optimal one is chosen by learning the *importance factor* for each block. However, NAS essentially degrades to network pruning in the context of MLPs, because the *importance factors* for blocks in DNNs can be interpreted directly as the *weights* in MLPs.

Network pruning refers to remove parts of the network parameters to reduce the network size. Here, a regularizer (penalty function) is often employed to encourage higher sparsity of network parameters. Depending on the forms of regularization, pruning can be divided into *unstructured pruning* (targeting individual parameters) [48] and *structured pruning* (targeting groups of parameters) [49]. The former typically incorporates the ℓ_p norms of parameters into the regularizer independently, e.g., through

$$\|g_1\|_1 + \|g_2\|_1 + \cdots + \|g_i\|_1 + \cdots \tag{6}$$

to promote increased number of zero-valued parameters In contrast, the latter applies penalties to the ℓ_p norms of grouped parameters, e.g., all weights associated with a neuron

$$\|[g_1,\, g_2,\, \cdots,\, g_i,\, \cdots]\|_2 \tag{7}$$

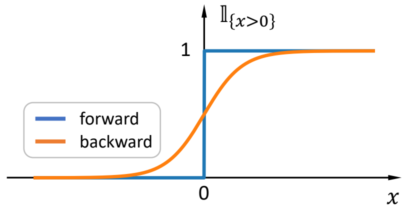


Fig. 7. Forward and backward pass of the soft count. The blue curve counts the number of x by 1 when $x > 0$, otherwise 0. In backward, the orange function is employed to derive the gradient information.

to foster the elimination of complete neurons. Therefore, the latter is also named grouped pruning [50, 51].

In ML, structured pruning is more favored as it streamlines the network architecture both from the algorithm and the hardware perspectives by simplifying the matrix multiplications. Unstructured pruning, on the other hand, does not provide obvious improvement due to the lack of conclusive tools to support sparse matrix multiplication. Conversely, in the context of pNCs, both pruning approaches bring significant benefits. Because owing to the highly flexible and agile manufacturing process of PE, removing any component can contribute to the compactness of the circuits: Unstructured pruning can remove crossbar resistors, whereas structured pruning enables to remove entire printed neurons. Notably, this is a unique advantage of the additive PE. In this regard, we employ combined unstructured pruning, Eq. (6) and structured pruning, Eq. (7) methods as a state-of-the-art baseline for the proposed evolutionary architecture algorithm.

B. Area-Aware Training with Pruning

In addition to existing network pruning methods, this work also enhances the existing pruning method to specifically encourage the compact design of pNCs, namely the *area-aware training*.

Despite the large amount of research on the functional forms [52]–[54] of regularization functions, these studies typically employ simple and differentiable functions, that are not directly applicable to assess the circuit area. To address this issue and explore more potential of gradient-based pruning in compact pNC design, we aim to incorporate circuit area of pNCs directly as the regularization term in the training objective. This approach promotes the explicit optimization of compact pNCs. Additionally, as the circuit area is significantly influenced by the circuit architecture, which is non-differentiable and fails to offer valid gradient information, we introduce gradient-relaxation methods to address these challenges and thus enable gradient-based training.

As illustrated by the dashed neuron in Fig. 6(a), the presence of a neuron can be expressed by the existence of its input weights embodied by the conductance g_i, i.e.,

$$\max_i \left\{ [\mathbb{1}_{\{g_1>0\}}, \mathbb{1}_{\{g_2>0\}}, \cdots, \mathbb{1}_{\{g_i>0\}}, \cdots] \right\}. \tag{8}$$

This method belongs to structured pruning, as it aims to eliminate the entire neuron. Different from traditional regularization like Eq. (7), Eq. (8) only suppresses the largest input conductance in the crossbar, which avoids the impact on other input conductances and thereby minimizing the effect on classification accuracy caused by the regularization. However, the indicator function $\mathbb{1}_{\{\cdot\}}$ is a piece-wise constant function, as shown by the blue function in Fig. 7, it has a gradient of zero almost everywhere. To address this issue and obtain gradient information that can guide the parameter update, we use the result of Eq. (8) in the forward pass, while calculating gradients using a smooth relaxation called soft counts [14], N^{soft}, in the backward pass. In this work, the $\text{sigmoid}(\cdot)$ function is used as the smoothing function, therefore, the function used for backpropagation of Eq. (8) is given by

$$\max_i \left\{[\text{sigmoid}(g_1), \text{sigmoid}(g_2), \cdots, \text{sigmoid}(g_i), \cdots]\right\},$$

as shown in by the orange function in Fig. 7.

Analogously, the presence of a negation circuit can be calculated through negative surrogate conductances, i.e.,

$$\max_j \left\{[\mathbb{1}_{\{\theta_1<0\}}, \mathbb{1}_{\{\theta_2<0\}}, \cdots, \mathbb{1}_{\{\theta_j<0\}}, \cdots]\right\}.$$

where θ_j refers to the succeeding conductances from a neuron, as shown by the green part in Fig. 6. If none of the corresponding weights is negative, the output voltage from the preceding neuron does not need to be negated [14]. Similarly, the gradient of this function is relaxed by

$$\max_j \left\{1 - [\text{sigmoid}(\theta_1), \text{sigmoid}(\theta_2), \cdots, \text{sigmoid}(\theta_i), \cdots]\right\}.$$

Regarding the count of the resistors, which aligns with unstructured pruning, we employ $\mathbb{1}_{g_i>0}$ to count each crossbar resistor, while its gradient is given by

$$\text{sigmoid}(g_i)$$

With these soft counts, the area estimator adapted for gradient-based is expressed as

$$A^{\text{soft}} = f(N_i^{\text{soft}}, A_i).$$

Finally, we use the cross-entropy function [42] to guide the training for higher accuracy. Comparable to Eq. (5), the training objective for the pruning is

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\ (1-\gamma)\,\text{CE}\left(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x}, \boldsymbol{\theta})\right) + \gamma\frac{A^{\text{soft}}(\boldsymbol{\theta})}{A'}, \qquad (9)$$

with $\boldsymbol{\theta}$ summarizing all learnable conductances in Eq. (3), and $\text{CE}(\cdot)$ refers to the cross-entropy loss function.

C. Fine-Tuning

It is worthy to highlight that pNCs trained with Eq. (9) may not reach optimal trade-off between area and accuracy. Because the area term, functioning as a penalty, suppresses the conductances through *soft count* for decreasing the device counts, and thus circuit footprint. However, if, e.g., a conductance cannot be suppressed to zero for a given γ, the resistor can not be removed. In this case, such suppression not only fails to reduce the circuit area, but also diminishes the classification accuracy by forcing parameters from the optimal values for the cross-entropy loss.

To mitigate this problem, we introduce the fine-tuning process. After the main training process introduced above, we generate masks m for each surrogate conductance θ and multiply them to indicate the parameters after pruning, i.e.,

$$\theta \leftarrow m \cdot \theta,$$

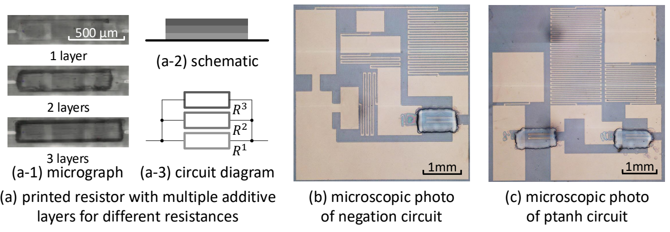

500 µm
1 layer
2 layers
3 layers
(a-1) micrograph
(a-2) schematic
R^3
R^2
R^1
(a-3) circuit diagram
1mm
1mm
(a) printed resistor with multiple additive layers for different resistances
(b) microscopic photo of negation circuit
(c) microscopic photo of ptanh circuit

Fig. 8. Schematics and photos of the primitives in printed neuromorphic circuits. (b) and (c) sourced from [10] with permission for reprinting.

where m is either 1 or 0, indicating whether the parameter is pruned. Regarding the pruning of neurons, if all input parameters of a neuron are pruned, the output voltage of this neuron will be multiplied with a mask value equaling 0. As for the negation circuits, we introduce

$$\theta \leftarrow \theta^+ \cdot (1 - m^{\text{N}}) + \theta \cdot m^{\text{N}}$$

to emulate the pruning of negation circuit. Here, $\theta^+ = \max\{0, \theta\}$, and m^{N} refers to the pruning of the negation circuit. $m^{\text{N}} = 0$ marks the negation circuit is pruned, therefore, surrogate conductance θ can only be positive.

Subsequently, we take the cross-entropy loss as the objective function to train the pruned network towards higher classification accuracy. With this mask-based pruning emulation, the classification accuracy can be further improved without any area increase. In other word, the impact of the noneffective area penalty on the classification accuracy can be recovered in the fine-tuning stage.

V. Circuit Area Model

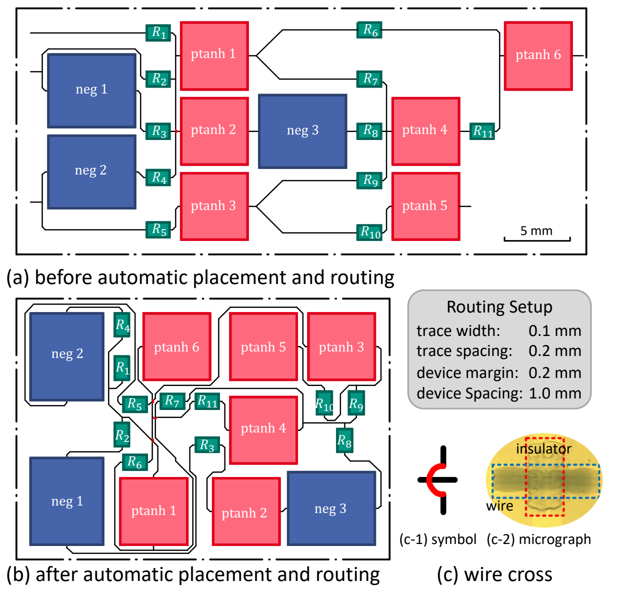
By establishing the training methodologies that allow optimizing circuit architectures, we can leverage highly flexible and bespoke manufacturing process of PE to reduce circuit footprint and enhance circuit compactness. However, achieving this objective requires also to establish a precise model for estimating the circuit area in the regularization term in the objective function.

A. Area of Circuit Primitive

The schematics of a printed resistor is depicted in Fig. 8(a). Due to the additive manufacturing, the resistance values are progressively modulated by sequentially depositing resistive material on top of the existing resistors. Therefore, differences in printed resistors with different values primarily arise in their thickness, whereas their areas remain unchanged. Moreover, Fig. 8(b) and (c) show the microscopic photos of the printed negation and ptanh circuits. As these circuits are predefined and fix during training, their respective areas remain also constant. Consequently, we read the area information directly from [10] as: area of the resistor (A_{R}) is $0.15\,\text{mm}^2$, area of negation circuit (A_{N}) is $30\,\text{mm}^2$ and area of activation (ptanh) circuit (A_{A}) is $22\,\text{mm}^2$, correspondingly.

B. Software for Circuit Area Calculation

To assess the area, we do **not** proceed to estimate the circuit area A directly through the summed area of each device. Because with the increasing number of devices, their connections will grow in a super-linear way, driving the


ptanh 1
ptanh 2
ptanh 3
ptanh 4
ptanh 5
ptanh 6
neg 1
neg 2
neg 3
R_1 R_2 R_3 R_4 R_5 R_6 R_7 R_8 R_9 R_{10} R_{11}
5 mm
(a) before automatic placement and routing
(b) after automatic placement and routing
insulator
wire
(c-1) symbol
(c-2) micrograph
(c) wire cross

Routing Setup
trace width: 0.1 mm
trace spacing: 0.2 mm
device margin: 0.2 mm
device Spacing: 1.0 mm

Fig. 9. Placement and routing of the printed analog neuromorphic circuits (pNCs). (a) A naive placement that mimics the form of neural networks described in Fig. 6(b), and (b) the solution of the automatic placement and routing from EasyEDA software. The gray box shows the major setups of the algorithm (with technology specification of PE). (c) illustrates the wire cross in PE: (c-1) is the symbol of cross that appears in (a) and (b), while (c-2) denotes the microscopic photo of a wire-cross with PEDOT:PSS as the conductive wire and dimethyl sulfoxide (DMSO) as the insulator, from [55].

routing problem more complicated and thus requiring more area than their linear relationship.

Rather, we utilize mature and well-developed commercial electronic design automation (EDA) tools to facilitate automatic placement and routing, serving as an estimator of circuit footprints. Although there are no specialized placement and routing tools developed for PE, the commonalities between PE and printed circuit boards (PCBs) suggests us to apply PCB-design tools for PE. Because both PE and PCBs are to place some predefined geometric components in a 2D surface, and their routing is featured by the connected pins on the given 2D space. In this context, we utilize EasyEDA[1] tool for the automatic placement and routing of pNCs.

Although PE is predominantly a 2D technology, thanks to the additive manufacturing, the issue of wire-crossing can be simply addressed. As illustrated in Fig. 9(c), we can print insulating materials (in our case, the dimethyl sulfoxide, DMSO) on top of the printed wires at the region that will be crossed. Subsequently, the second wire can be printed over the insulator. This approach is similar to multilayer PCBs with via holes, where the PCB substrates function as the insulator to avoid the intersection of wires, and the via holes provide connections across layers of substrates. Therefore, the automatic routing algorithm can natively support the tasks in PE. However, the additive PE offers significantly greater flexibility than PCBs in managing such issues.

C. Circuit Area Estimator

As it is hard to integrate the EasyEDA into the training approach of the pNCs, we have to develop a model that can estimate the circuit area during training and can be integrated into the algorithm. Given the sophisticated relationship

[1]The software is available at https://easyeda.com/.

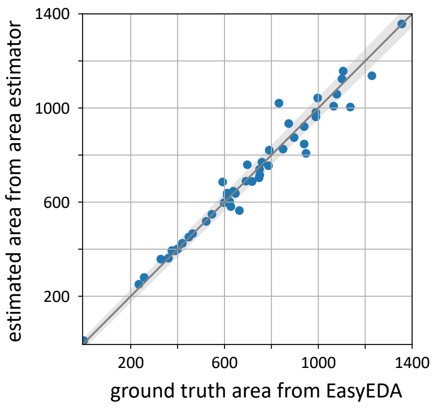


Fig. 10. Performance of the VAE-based area estimator on the test data. The x-axis is the ground truth area from the EasyEDA, while the y-axis denotes the estimated areas. Each blue point is a test data. The gray diagonal line refers to the ideal case where the estimation equals the ground truth. The gray area around the line represents the variation of the ground truth areas.

between the circuit area and its architecture, we adopt an ANN-based model as the area estimator, because it is proven to be a universal approximator. The establishment of the area estimator comprised three stages: data acquisition, model design, and model training.

a) Data acquisition: We first defined our customized library in EasyEDA based on the printed component geometry depicted in Fig. 8, which includes dimensions and pin configurations. Subsequently, we randomly generate 500 different pNCs architectures and convert them into netlists for importation into EasyEDA for automatic placement and routing. Key setups for placement and routing are reported in the gray box in Fig. 9. After this, we use the minimum bounding rectangles for each circuit to denote the area footprint of the pNCs. Since the algorithm provided by EasyEDA is not deterministic, i.e., each conduction may produce a different result, we repeated the algorithm 10 times per pNC, and record their bounding box areas.

b) Area estimator model: With the collected data, we constructed an ANN-based model that is capable of estimating the circuit areas A from their device counts N_i and device areas A_i, denoted by

$$A = \mathrm{AreaEstimator}(N_i, A_i).$$

As the area generated by EasyEDA is not a deterministic value but rather follows a certain distribution, we employ a Variational Autoencoder (VAE) as the area estimator. Because VAEs natively support probability distribution as model output [56] and widely used as generative AI models [57]. In our area estimator model, input features (circuit netlists) are encoded to multiple latent distributions. The decoder then draws samples from the latent distributions to estimate the circuit areas.

c) Area estimator training: To train the area estimator for precisely estimating circuit areas, the collected dataset is randomly divided into training (60%), validation (20%), and test (20%) sets. We used the training data to guide the training of the model, while employing the validation set to avoid overfitting through the early-stopping technique. Meanwhile, we utilized data normalization and hyperparameter tuning to enhance the precision of the area estimator. Afterwards, the final model with a 7-layer encoder and a 7-layer decoder is selected as the area estimator for pNCs. Fig. 10 illustrates the model performance on test data, which has not been used during training. The results suggest that, the area estimator can provide acceptable ($\leq 5\%$ error) area estimation, and does not overfit the training data.

Finally, the well-trained area estimator is used as the area terms in Eq. (5) and Eq. (9) to provide a precise circuit area.

VI. Evaluation

To evaluate the efficacy of the proposed methods, we implemented both evolutionary and pruning algorithms[2] with PyTorch and conducted experiments on 13 benchmark datasets, which are recommended by the neuromorphic computing survey [9]. They are also employed in other state-of-the-art studies on pNCs [10, 26] and aligned with the complexity of the application domains of PE. As the functionality of the printed neuromorphic hardware has been validated in [10, 58] and the contribution of this work, i.e., circuit area, can be completely verified at the algorithmic level, the experiment is conducted at simulation level based on the pPDK [59].

TABLE I
Device Parameters in Nonlinear Circuits in pPDK
(VDD=2V, Threshold Voltage $V_{\mathrm{th\_EGT}}$=0.24V)

negation circuit	R_1^{N} (kΩ)	R_2^{N} (kΩ)	R_3^{N} (kΩ)	L_1^{N} (µm)	W_1^{N} (µm)	L_2^{N} (µm)	W_2^{N} (µm)	L_3^{N} (µm)	W_3^{N} (µm)
value	250	8	500	100	80	500	30	80	150
ptanh circuit	R_1^{A} (kΩ)	R_2^{A} (kΩ)	–	L_1^{A} (µm)	W_1^{A} (µm)	L_2^{A} (µm)	W_2^{A} (µm)	–	–
value	2050	7	–	80	80	480	40	–	–

A. *Circuit Design Setup*

In the following, we provide a detailed description of the negation circuit and ptanh activation circuit based on our feasible circuit design space in the pPDK [59]. It is worth note that, the printed EGTs are characterized by the length L and the width W of the channel size [60], i.e., the overlap between the top gate (PEDOT:PSS) and the semiconductor ($\mathrm{In_2O_3}$) in Fig. 3(a).

The device parameters used in the pPDK for the negation circuits and ptanh activation circuits are reported in Tab. I. Regarding printed resistors, as they need to have different values to reflect specific weights, they are a range in pPDK rather than a fixed value, specifically, from $100\,\mathrm{k\Omega}$ to $10\,\mathrm{M\Omega}$.

B. *Experiment Setup*

We train pNCs with both EA and gradient approaches. For gradient-based training, we employ the existing (area-unaware) pruning as the baseline of this work. Meanwhile, we run the proposed area-aware pruning as another benchmark for compact pNC design.

a) Initialization: By referring to other works on EA and through a few preliminary trials, the population size is initialized to $1\,000$. Moreover, to preserve a minimum size of the circuits, the topologies for all datasets are initialized as unconnected networks, i.e., featuring only $\#\mathrm{output}$ nodes.

[2]The code for the training framework and the experiment will be available upon acceptance at https://github.com/Neuromophic/Area-Aware-Training.

Regarding network pruning via gradient-based approach, as pruning can only remove circuit components, it is critical to start with a larger architecture to ensure the competitive sub-architectures are included in the search space. Therefore, initialize the $\#\mathrm{input}$-4-3-$\#\mathrm{output}$ topology as a basis structure for pruning. This initial size is slightly more expansive than those typically employed in other works such as [26].

In addition, through few initial trials, the calibrating factor A' in Eq. (5) is empirically set to $600\,\mathrm{mm}^2$. Since this term simply aims to balance the loss term and the area term into a similar order of magnitudes, A' is not required to be a precise value, and will not impact the training results.

b) Training: During evolution, the top 10 best genomes in each species are chosen to be the candidate parents for crossover. The patience for species improvement, i.e., $\mathcal{K}_1$, is 20, while the number of species being protected, i.e., $\mathcal{K}_2$, is 2. In mutation, the probability of both adding node and adding edge in the mutation phase is 0.7, while that of deleting node and deleting edge is 0.3. In this way, the networks tend to have increasing size. Meanwhile, the edge state has 0.1 possibility to switch from enabled to disabled and vice versa. We employ full-batch training with the stop criteria being a 100-generation patience, i.e., $\mathcal{K}_3$, referring to consistently no improvement on the validation set. Moreover, to investigate the trade-off between accuracy and area, we run 50 evolutions with 50 different values in $\gamma \in [0, 1]$. The whole process is repeated 10 times (with random seed varying from 1 to 10) for each value of γ to make sure to achieve a sufficiently good solution.

Regarding gradient-based pruning approach, we employ the Adam [61] optimizer in default parameterization to update parameters. To prevent overfitting, we use early-stopping [62], namely, we initialize the learning rate with 0.1 and halve it after a patience (updates without improvement on objective function) of 100-epochs on the validation set. The training process is stopped, when the learning rate was halved 10 times. Other setups are kept the same as its EA counterpart. Similarly, fine-tuning is conducted to further improve the classification accuracy of the pruned networks.

C. *Result*

After training, results are calculated on the corresponding test sets. It is evident that, with increasing γ, the training objective gradually transitions from prioritizing classification accuracy to minimizing circuit area. Conceptually, $\gamma = 0$ yields the highest accuracy and the largest area. As the objective in this case only focuses on the accuracy and ignores the circuit area, the network trained through gradient approach in this case is therefore referred to as the *reference*. We report the classification accuracy, circuit area, and training time in this case in Tab. II.

Meanwhile, as γ increased, the both area and classification accuracy decreased. In this process, since the reduction ratio of accuracy and area holds more significance than their specific values, subsequent data will be normalized by the results of the *reference* values of the baseline, i.e., the existing pruning. The change of accuracy and area versus γ is described in Fig. 11.

TABLE II
SIMULATION RESULT AND RUNTIME OF THREE APPROACHES ON 13 BENCHMARK DATASETS WITH $\gamma = 0$

Dataset	Existing (area-unaware) Pruning			Area-Aware Pruning			Proposed EA Approach		
	Accuracy	**Area** (mm^2)	**Runtime** (min)	**Accuracy**	**Area** (mm^2)	**Runtime** (min)	**Accuracy**	**Area** (mm^2)	**Runtime** (min)
Acute Inflammation	1.000 ± 0.000	688 ± 47	8.3 ± 1.1	1.000 ± 0.000	743 ± 74	7.9 ± 1.5	1.000 ± 0.000	165 ± 17	32.8 ± 10.4
Balance Scale	0.966 ± 0.065	890 ± 40	11.5 ± 2.4	0.930 ± 0.030	894 ± 28	11.5 ± 2.5	0.929 ± 0.019	269 ± 5	101.1 ± 31.1
Breast Cancer Wisconsin	0.972 ± 0.003	827 ± 62	10.4 ± 2.2	0.970 ± 0.004	769 ± 136	11.3 ± 1.1	0.962 ± 0.013	176 ± 18	83.2 ± 13.7
Cardiotocography	0.876 ± 0.006	1378 ± 97	12.4 ± 1.9	0.867 ± 0.010	1299 ± 94	11.4 ± 2.6	0.865 ± 0.032	238 ± 16	118.9 ± 26.0
Energy Efficiency (y_1)	0.969 ± 0.006	959 ± 29	15.6 ± 3.8	0.972 ± 0.017	970 ± 32	13.6 ± 4.8	1.000 ± 0.000	228 ± 32	88.9 ± 36.4
Energy Efficiency (y_2)	0.921 ± 0.011	965 ± 62	11.2 ± 0.8	0.917 ± 0.022	988 ± 82	10.4 ± 1.1	0.903 ± 0.015	188 ± 3	82.9 ± 26.3
Iris	0.975 ± 0.012	832 ± 71	8.6 ± 0.7	0.962 ± 0.006	859 ± 59	7.5 ± 0.8	0.935 ± 0.000	227 ± 6	32.6 ± 12.2
Mammographic Mass	0.857 ± 0.008	876 ± 40	9.8 ± 0.8	0.860 ± 0.012	827 ± 33	9.1 ± 1.2	0.865 ± 0.019	213 ± 2	71.3 ± 21.6
Pendigits	0.322 ± 0.013	1121 ± 30	21.6 ± 3.8	0.338 ± 0.070	1076 ± 177	22.2 ± 2.5	0.485 ± 0.102	551 ± 13	196.3 ± 66.5
Seeds	0.919 ± 0.023	957 ± 34	8.7 ± 0.8	0.936 ± 0.045	915 ± 62	8.9 ± 0.8	0.884 ± 0.023	262 ± 19	47.0 ± 19.5
Tic-Tac-Toe Endgame	0.998 ± 0.004	883 ± 90	9.7 ± 0.5	0.974 ± 0.024	982 ± 36	9.4 ± 0.5	0.951 ± 0.015	199 ± 5	75.3 ± 26.3
Vertebral Column (2 cl.)	0.828 ± 0.006	811 ± 12	7.3 ± 0.5	0.825 ± 0.007	816 ± 2	7.9 ± 0.5	0.815 ± 0.009	238 ± 3	46.5 ± 20.3
Vertebral Column (3 cl.)	0.817 ± 0.017	978 ± 12	8.8 ± 1.2	0.810 ± 0.025	939 ± 60	8.6 ± 1.3	0.838 ± 0.009	238 ± 7	55.2 ± 16.8
Average	**0.878 ± 0.013**	**936 ± 48**	**11.2 ± 1.6**	**0.874 ± 0.021**	**929 ± 67**	**10.7 ± 1.6**	**0.879 ± 0.020**	**246 ± 11**	**79.4 ± 25.2**

a) Existing pruning vs. area-aware training: By comparing the state-of-the-art pruning methods with the modified compactness-specific area-aware pruning, we conclude that, although they yield similar accuracies, the circuit area from existing pruning is consistently larger than the area-aware pruning. We speculate that, this is because of the unawareness of the circuit area of the existing pruning. For instance, the device counts of the negation circuits is not included in the regularization term. Consequently, the training will not encourage more positive weights to reduce the number of negation circuits.

b) EA approach vs. area-aware training: It can be seen that even with $\gamma = 0$, EA can already achieve competitive classification accuracy and a substantially smaller circuit area, when compared to the gradient approach. We speculate that, this is because the EA method starts the search from the minimal architecture, and thus may converge near it. Subsequently, as the γ increases, although both methods decrease the accuracy and area of pNCs, the EA method produces a more modest degradation in accuracy. Because when $\gamma = 1$, the training objective focuses solely on minimizing area, disregarding classification accuracy. This should ideally produce the smallest circuit area, where only the output neurons are expected to be kept. In this case, EA enables input signals to be directly connected to output neurons to produce a reasonable classification result (even though the computing power of a single neuron is very weak). But the pruning-based approach can not even achieve this, because all neurons in the shallow layers are pruned, resulting in a forward propagation that was cut off from the input signal. Consequently, the output of the network behaves no better than a random guess.

TABLE III
COMPARISON OF ACCURACY-AREA TRADE-OFF

Normalized	**Area-Aware Pruning**		**EA (superiority over pruning)**	
Accuracy (%)	**Area (%)**	**Power** (mW)	**Area (%)**	**Power** (mW)
100	100	78	32 ($\downarrow$ 3.1×)	26 ($\downarrow$ 3.0×)
90	44	37	15 ($\downarrow$ 2.9×)	13 ($\downarrow$ 2.9×)
80	31	22	12 ($\downarrow$ 2.5×)	4 ($\downarrow$ 5.5×)

To obtain the Pareto optimal trade-offs between accuracy and area, we plot the entirety of normalized areas versus their respective normalized accuracies for all runs and all values of γ in Fig. 12 with the green (for existing pruning), red (for area-aware pruning), and black scatters (for EA). Based on the scatters, three Pareto fronts are drawn

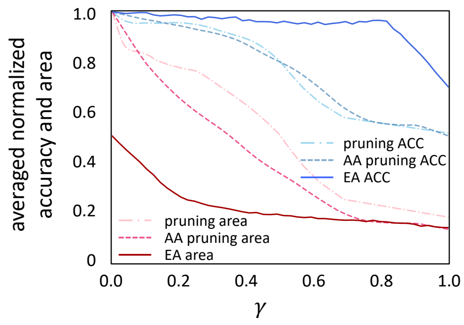


Fig. 11. Results of the experiment: averaged normalized accuracy (ACC) and area from three methods, namely the evolutionary algorithm (EA)-based training, the area-aware (AA) pruning, and the existing pruning method.

with their identical colors. Notably, the Pareto front of EA significantly outperforms that of the gradient-based training methods. Because EA natively support the optimization of circuit architectures. Moreover, the area-aware training yields better trade-offs compared to the area-unaware counterpart.

To provide more quantitative comparison, Tab. III displays several trade-off points from the Pareto fronts. It is evident that the EA approach offers $3.1\times$ area-savings without any accuracy degradation compared to pruning. In case a 10% reduction in normalized accuracy is permissible, only 15% of the reference area is needed, which is $2.9\times$ area reduction compared to area-aware pruning. Moreover, we also report the power consumption of the trained pNCs through an established power consumption model from [14]. Specifically, the power are calculated from a hybrid model consisting of an analytical formula for resistor crossbars and a ANN-based surrogate power model for the nonlinear circuits. We observe that, as a byproduct of lower device counts, the power can also be greatly reduced compared to pruning. Specifically, the EA surpasses the area-aware pruning method by $3.0\times$ and $2.9\times$ power reduction respectively while providing 100% and 90% normalized accuracies.

D. Multivariate Regression Analysis

To further assess the contribution of each circuit primitive in the circuit performance, we conducted multivariate regression analysis and reported the result in Tab. IV. It can be observed, each circuit primitive contributes to an increase in both area and power consumption, while also enhancing the classification accuracy.

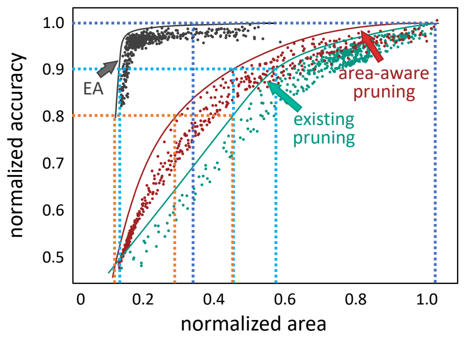


Fig. 12. Scatters and Pareto fronts of three methods, green for existing pruning that is unaware of circuit area, red for proposed area-aware pruning, and black for evolutionary algorithm (EA)-based area-aware training.

Notably, the final circuit areas are contributed through resistor, negation circuit, and activation circuit by $3.19\,\mathrm{mm}^2$, $42.2\,\mathrm{mm}^2$, and $27.9\,\mathrm{mm}^2$, respectively, which are slightly larger than the areas of the primitives themselves ($0.15\,\mathrm{mm}^2$, $30\,\mathrm{mm}^2$, and $22\,\mathrm{mm}^2$). Especially, while the nonlinear circuits contribute to the final area similarly to their individual area, the contribution of resistors is significantly larger than their own areas. This is because the resistors not only occupy their own area but also indicate the interconnection between neuron, which can lead to sophisticated placement and routing problems. This further justifies our work in establishing precise circuit area estimator based on EasyEDA and its placement & routing algorithm.

TABLE IV
COEFFICIENTS IN MULTIVARIATE REGRESSION ANALYSIS

Target\Factor	N_R	N_N	N_A
Area (mm^2)	3.1860	42.221	27.949
Accuracy	0.0158	0.0253	0.0790
Power (mW)	0.0339	0.3753	0.1553

E. Algorithm Complexity

Beyond the primary concerns, i.e., area and accuracy in this work, the complexity of the algorithms also draws our attention, because this is a notable distinction between gradient and evolutionary approaches. Thus, we summarize the training times for both methodologies in Tab. II.

In our experiments, EA demands significantly more time ($\approx\ 7\times$) than its gradient counterpart. However, we have the following comments on this issue: *(i)* The most time-consuming steps in EA are evaluating genomes and producing offsprings, which are strongly related (almost proportional to) the population size N. To fully explore the potential capabilities of EA, we have selected a large N in this work, nevertheless, it might be reduced in other scenarios. *(ii)* Genome evaluation and offspring production in EA are highly suitable to parallelization, through which the training can be accelerated. Although the parallelization is not included in this work, this can be done as a part of future work. *(iii)* Even though EA takes longer training time in our setup, the duration (ranging from 30 minutes to 3 hours) remains acceptable in the broader context of the product development cycle. This is not only because circuit optimization is only part of the non-recurring engineering (NRE), but also due to the target applications of PE that often require only small-scale circuits.

F. Case Study

To provide a clear and comprehensive overview of the effectiveness of our proposed method, we illustrate the algorithm performance with the *Energy Efficient* y_2 dataset as an example, as shown in Fig. 13.

From Fig. 13(a) we know that, without the precise area estimator model, i.e., without area-aware training, the circuit architecture often fails to be optimized for compactness, resulting in complicated circuits structure with high power consumption and area footprint.

In contrast, with the established area model and pruning technique, the algorithm can effectively remove redundant components with awareness of circuit area (Fig. 13b). This significantly simplifies the circuit architecture while maintaining unchanged classification accuracy. However, pruning still presents obvious limitations: 1) It is heavily dependent on the initial circuit architecture. Since pruning can only reduce circuit architecture from the original one, the initial design strongly restricts the upper bound of circuit scale, and thereby the circuit classification accuracy. 2) Meanwhile, the lower bound of pruning is also limited. For instance, pruning may not remove an entire layer, as it will interrupt the signal propagation, which could result in an output irrelevant to the input signal.

In opposite, our EA-based automatic architecture search addresses the limitations of pruning, and can therefore yield an optimal circuit architecture with significant smaller circuit area and power consumption, while preserving comparable classification accuracy.

VII. CONCLUSION

Printed electronics, owing to its unique features, emerges as the enabler of future electronics. Within this realm, analog pNCs attracts increasing attention as they offer tailored computational functionalities and ultra-low cost in edge computing scenarios. However, inherent challenges, such as large feature sizes and limited device counts, restrict the broader application of pNCs in compact applications. To address this issue, we leveraged the capability of PE for flexibly printing any bespoke circuit architecture for specific target objectives. To explore this potential, we proposed methods for the compact design of pNCs by proposing an EA that enables the simultaneous training of both circuit architecture and parameters, while considering the resulting area of the circuit in the proposed area-aware training objective as shown in Fig. 14). As benchmark, we utilized a traditional pruning method and a modified compactness-oriented pruning method with area-aware training objective. Simulation results reveal that all three methods are capable to facilitate compact design of pNCs, however, EA presents a superiority over the gradient-based pruning benchmarks. Since the proposed EA methods open up a new search space of circuit architecture, they offer potential utility also for other objectives, such as enhancing robustness against printing variations. Moreover, future work may introduce additional genes in EA to enable the training of other circuit components, e.g., the geometric features of EGTs. Also, the acceleration of the EA can be considered.

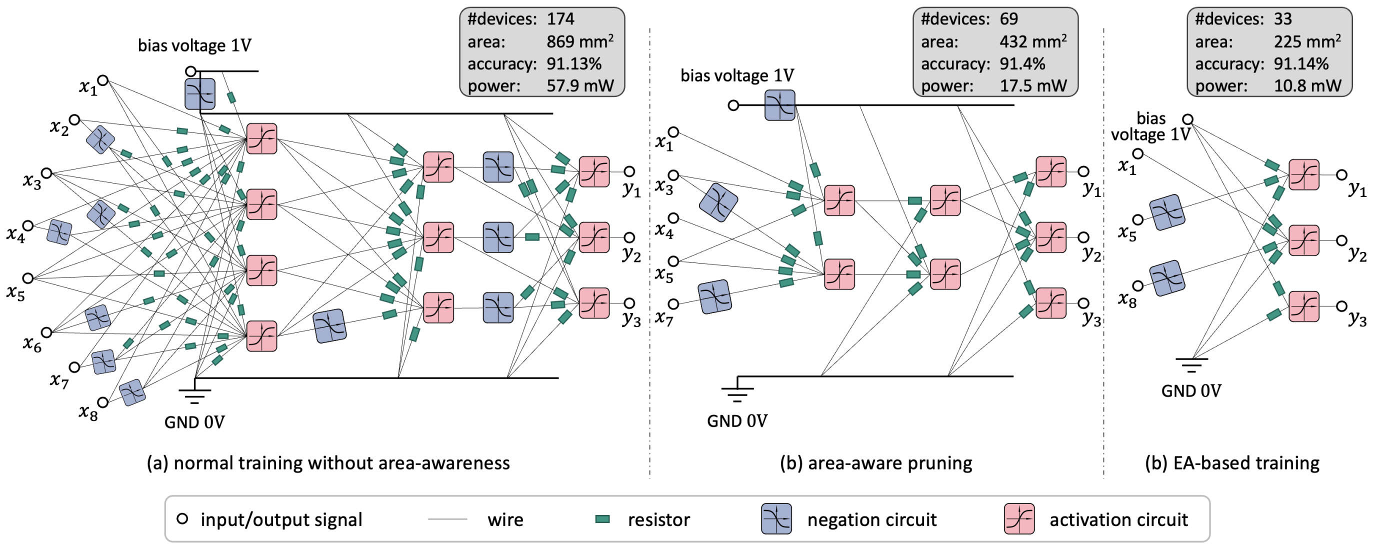


Fig. 13. Case study on the *Energy Efficiency* y_2 dataset with 8 input features and 3 output classes. Subfigures show the minimal area footprint to achieve $\approx 91\%$ classification accuracy. The circuits are training from (a) previous area-unaware training, (b) area-aware pruning, and (c) EA-based area-aware training with architecture search. To maintain clarity, we illustrate only the symbolic diagram at the primitive-level without automatic placement. Nevertheless, we report the related attributes in the gray boxes.

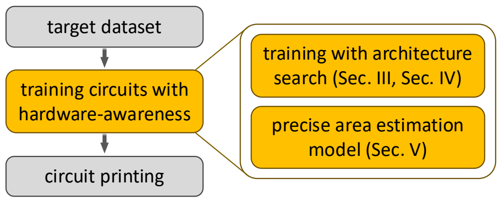

target dataset
training circuits with hardware-awareness
circuit printing
training with architecture search (Sec. III, Sec. IV)
precise area estimation model (Sec. V)

Fig. 14. Workflow of the entire process for obtaining a printed neuromorphic circuit. The yellow blocks highlight the major contribution in this work.

References

[1] E. Shirzaei Sani *et al.*, "A Stretchable Wireless Wearable Bioelectronic System for Multiplexed Monitoring and Combination Treatment of Infected Chronic Wounds," *Science Advances*, vol. 9, no. 12, p. 7388, 2023.

[2] R. Martins *et al.*, "Electronics with and on Paper," *physica status solidi (RRL)–Rapid Research Letters*, vol. 5, no. 9, pp. 332–335, 2011.

[3] Z. Cui, *Printed Electronics: Materials, Technologies and Applications*. John Wiley & Sons, 2016.

[4] I. I. Labiano *et al.*, "Flexible Inkjet-printed Graphene Antenna on Kapton," *Flexible and Printed Electronics*, vol. 6, no. 2, p. 025010, 2021.

[5] A. Kaidarova *et al.*, "Wearable Multifunctional Printed Graphene Sensors," *NPJ Flexible Electronics*, vol. 3, no. 1, pp. 1–10, 2019.

[6] J. Li *et al.*, "Micro and Nano Materials and Processing Techniques for Printed Biodegradable Electronics," *Materials Today Nano*, vol. 18, 2022.

[7] A. Nešić *et al.*, "Prospect of Polysaccharide-based Materials as Advanced Food Packaging," *Molecules*, vol. 25, no. 1, p. 135, 2019.

[8] C. Zhang *et al.*, "Time-Temperature Indicator for Perishable Products Based on Kinetically Programmable Ag Overgrowth on Au Nanorods," *ACS nano*, vol. 7, no. 5, pp. 4561–4568, 2013.

[9] C. D. Schuman *et al.*, "A Survey of Neuromorphic Computing and Neural Networks in Hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[10] D. D. Weller *et al.*, "Realization and Training of an Inverter-Based Printed Neuromorphic Computing System," *Scientific reports*, vol. 11, 2021.

[11] Q. Sun *et al.*, "Smart Band-Aid: Multifunctional and Wearable Electronic Device for Self-Powered Motion Monitoring and Human-Machine Interaction," *Nano Energy*, vol. 92, p. 106840, 2022.

[12] A. U. Alam *et al.*, "Fruit Quality Monitoring with Smart Packaging," *Sensors*, vol. 21, no. 4, p. 1509, 2021.

[13] K. O. Stanley *et al.*, "Efficient Evolution of Neural Network Topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, vol. 2. IEEE, 2002, pp. 1757–1762.

[14] H. Zhao *et al.*, "Power-Aware Training for Energy-Efficient Printed Neuromorphic Circuits," in *42nd IEEE/ACM International Conference on Computer-Aided Design*, 2023.

[15] S. Chung *et al.*, "All-Inkjet-Printed Organic Thin-Film Transistor Inverter on Flexible Plastic Substrate," *IEEE electron device letters*, vol. 32, no. 8, pp. 1134–1136, 2011.

[16] F. Shao and Q. Wan, "Recent Progress on Jet Printing of Oxide-Based Thin Film Transistors," *Journal of Physics D: Applied Physics*, vol. 52, no. 14, p. 143002, 2019.

[17] H. Zhao *et al.*, "Printed Electrodermal Activity Sensor with Optimized Filter for Stress Detection," in *International Symposium on Wearable Computers (ISWC'22), Atlanta, GA, September 11-15, 2022*.

[18] A. M. Turing, *Computing Machinery and Intelligence*. Springer, 1950.

[19] J. Von Neumann and R. Kurzweil, *The Computer and The Brain*. Yale university press, 1958.

[20] V. N. John, "First Draft of a Report on the EDVAC," *Pennsylvania: University of Pennsylvania*, 1945.

[21] D. Monroe, "Neuromorphic Computing Gets Ready for the (Really) Big Time," 2014.

[22] J. Schemmel *et al.*, "A Wafer-scale Neuromorphic Hardware System for Large-scale Neural Modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2010, pp. 1947–1950.

[23] L. A. Pastur-Romay *et al.*, "Deep Artificial Neural Networks and Neuromorphic Chips for Big Data Analysis: Pharmaceutical and Bioinformatics Applications," *International journal of molecular sciences*, vol. 17, no. 8, p. 1313, 2016.

[24] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[25] M. Ansari *et al.*, "PHAX: Physical Characteristics Aware Ex-Situ Training Framework for Inverter-Based Memristive Neuromorphic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, 2017.

[26] H. Zhao *et al.*, "Highly-Bespoke Robust Printed Neuromorphic Circuits," in *Design, Automation and Test in Europe (DATE)*. IEEE, 2023.

[27] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[28] T. Liang *et al.*, "Pruning and Quantization for Deep Neural Network Acceleration: A Survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[29] H. Liu *et al.*, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations (ICLR)*, 2018.

[30] T. Bäck *et al.*, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary computation*, vol. 1, pp. 1–23, 1993.

[31] H. Mubarik *et al.*, "Printed Machine Learning Classifiers," in *53rd Annual International Symposium on Microarchitecture (MICRO)*, 2020, pp. 73–87.

[32] E. H. Rijnbeek, "Approximate Computing - Reconsidering the Analog Computer," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:230730120

[33] G. Armeniakos *et al.*, "Cross-Layer Approximation For Printed Machine Learning Circuits," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 190–195.

[34] L. Deng *et al.*, “Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey,” *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[35] T. Elsken *et al.*, “Neural Architecture Search: A Survey,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[36] A. Gholami *et al.*, “A Survey of Quantization Methods for Efficient Neural Network Inference,” in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2021, pp. 291–326.

[37] F. Gomez *et al.*, “Accelerated Neural Evolution through Cooperatively Coevolved Synapses.” *Journal of Machine Learning Research*, vol. 9, no. 5, 2008.

[38] C. D. Schuman *et al.*, “Evolutionary Optimization for Neuromorphic Systems,” in *Proceedings of the 2020 Annual Neuro-Inspired Computational Elements Workshop*, 2020, pp. 1–9.

[39] N. Kasabov *et al.*, “Evolving Spiking Neural Networks for Personalised Modelling, Classification and Prediction of Spatio-Temporal Patterns with a Case Study on Stroke,” *Neurocomputing*, vol. 134, pp. 269–279, 2014.

[40] R. Miikkulainen *et al.*, “Evolving Deep Neural Networks,” in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2024, pp. 269–287.

[41] S. Cawley *et al.*, “Hardware Spiking Neural Network Prototyping and Application,” *Genetic Programming and Evolvable Machines*, vol. 12, pp. 257–280, 2011.

[42] A. Mao *et al.*, “Cross-Entropy Loss Functions: Theoretical Analysis and Applications,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 23 803–23 828.

[43] T. Elsken *et al.*, “Neural Architecture Search: A Survey,” *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[44] H. Cai *et al.*, “Efficient Architecture Search by Network Transformation,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[45] K. Greff *et al.*, “LSTM: A Search Space Odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[46] B. Zoph *et al.*, “Learning Transferable Architectures for Scalable Image Recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[47] Z. Zhong *et al.*, “Practical Block-wise Neural Network Architecture Generation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[48] S. Han *et al.*, “Learning Both Weights and Connections for Efficient Neural Network,” *Advances in neural information processing systems*, vol. 28, 2015.

[49] H. Li *et al.*, “Pruning Filters for Efficient Convnets,” *arXiv preprint arXiv:1608.08710*, 2016.

[50] J. Yoon and S. J. Hwang, “Combined Group and Exclusive Sparsity for Deep Neural Networks,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 3958–3966.

[51] Z. Liu *et al.*, “Learning Efficient Convolutional Networks through Network Slimming,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.

[52] R. Ma *et al.*, “Transformed ℓ_1 Regularization for Learning Sparse Deep Neural Networks,” *Neural Networks*, vol. 119, pp. 286–298, 2019.

[53] R. Mazumder *et al.*, “SparseNet: Coordinate Descent with Nonconvex Penalties,” *Journal of the American Statistical Association*, vol. 106, no. 495, pp. 1125–1138, 2011.

[54] Y. Zhou *et al.*, “Deep Neural Network Pruning with Progressive Regularizer,” in *2024 IEEE International Joint Conference on Neural Network (IJCNN 2024), Yokohama, 30th June - 05 July 2024*. Institute of Electrical and Electronics Engineers (IEEE), 2024.

[55] F. Rasheed *et al.*, “Crossover-aware Placement and Routing for Inkjet Printed Circuits,” *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–22, 2020.

[56] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[57] A. Mishra *et al.*, “A Generative Model for Zero Shot Learning Using Conditional Variational Autoencoders,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 2188–2196.

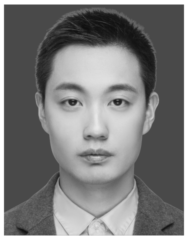
[58] S. A. Singaraju *et al.*, “Artificial Neurons on Flexible Substrates: A Fully Printed Approach for Neuromorphic Sensing,” *Sensors*, vol. 22, no. 11, p. 4000, 2022.

[59] F. Rasheed *et al.*, “Variability Modeling for Printed Inorganic Electrolyte-gated Transistors and Circuits,” *IEEE transactions on electron devices*, vol. 66, 2018.

[60] F. Rasheed, M. Rommel, G. C. Marques, W. Wenzel, M. B. Tahoori, and J. Aghassi-Hagmann, “Channel geometry scaling effect in printed inorganic electrolyte-gated transistors,” *IEEE Transactions on Electron Devices*, vol. 68, no. 4, pp. 1866–1871, 2021.

[61] D. P. Kingma *et al.*, “Adam: A Method for Stochastic Optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[62] L. Prechelt, “Automatic Early Stopping Using Cross Validation: Quantifying the Criteria,” *Neural networks*, vol. 11, no. 4, pp. 761–767, 1998.



Haibin Zhao received his B.Sc. in Mechanical Engineering from Chongqing University, China, in 2017. And he received his M.Sc. in mechatronics and information technology as well as Ph.D. in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2020 and 2024, respectively.

His research focuses on emerging neuromorphic computing paradigms, optimization theory, machine learning, AI-driven optimization and design automation for printed electronics.

Priyanjana Pal earned her B.Tech. in electronics and communication engineering from the National Institute of Technology, Agartala, in 2018, and her M.Tech. in electrical engineering from IIT Gandhinagar in 2020. Then, she worked as a senior ESD device design engineer at Global Foundries in Bangalore. In 2022, she joined CDNC at the Karlsruhe Institute of Technology, Germany.

Her research focuses on analog circuit design with CAD tools, reliability testing for printed circuits, and ML-trained design of printed neuromorphic circuits.

Michael Hefenbrock received the M.Sc. and Ph.D. degree in computer science from the Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2016 and 2022, respectively.

He is the founder and Head of artificial intelligence of RevoAI GmbH in Germany.

His research interests include machine learning theory, optimization, and design automation of emerging technologies.

Yuhong Wang received her B.Sc. from Hefei University of Technology, China in 2015, and her M.Sc. degree in mechatronics and information technology from the Karlsruhe Institute of Technology, Germany, in 2023, respectively.

Her research interests include mechatronics, robotics, and machine learning.

Michael Beigl received his Ph.D. from the University of Karlsruhe (now KIT) and is a Professor of Pervasive Computing Systems at KIT and Head of TECO Research Lab. Previously, he was a Professor at TU Braunschweig (2006-2010), a Visiting Associate Professor at Keio University, Japan (2005). Since 2014, he has led the SDIL, and SDSC-BW.

His research focuses on integrating human and computing systems, particularly in sensing and combining artificial and human intelligence.

Mehdi B. Tahoori (IEEE Fellow) received his B.Sc. in computer engineering in 2000. He received his M.Sc. and Ph.D. in electrical engineering from Stanford University in 2002 and 2003.

He is a Professor at KIT. Previously, he was a researcher at Fujitsu Laboratories, an Associate Professor at Northeastern University, and a Visiting Professor at the University of Tokyo.

He has received the NSF CAREER Award and multiple best paper awards, including at ICCAD, FPL, TODAES, and IEEE TVLSI.