# UNICO: Unified Hardware Software Co-Optimization for Robust Neural Network Acceleration

Bahador Rashidi*
Chao Gao*
bahador.rashidi@huawei.com
chao.gao4@huawei.com
Huawei Canda Research Center
Edmonton, AB, Canada

Shan Lu
Huawei Canda Research Center
Edmonton, AB, Canada
shan.828346@huawei.com

Zhisheng Wang
Huawei Hisilicon
Shanghai, China
wangzhisheng1@hisilicon.com

Chunhua Zhou
Huawei Canada Research Center
Edmonton, AB, Canada
zhouchunhua@huawei.com

Di Niu
University of Alberta
Edmonton, AB, Canada
dniu@ualberta.ca

Fengyu Sun
Huawei Hisilicon
Shanghai, China
sunfengyu@hisilicon.com

## ABSTRACT

Specialized hardware has become an indispensable component to deep neural network (DNN) acceleration. To keep up with the rapid evolution of neural networks, holistic and automated solutions for jointly optimizing both hardware (HW) architectures and software (SW) mapping have been studied. These studies face two major challenges. First, the combined HW-SW design space is vast, which hinders the finding of optimal or near-optimal designs. This issue is exacerbated for industrial cases when cycle accurate models are used for design evaluation in the joint optimization. Second, HW design is prone to overfitting to the input DNNs used in the HW-SW co-optimization. To address these issues, in this paper, we propose UNICO, an efficient **Uni**fied **Co**-Optimization framework with a novel Robustness metric for better HW generalization. Guided by a high-fidelity surrogate model, UNICO employs multi-objective Bayesian optimization to effectively explore the HW design space, and conducts adaptive, parallel and scalable software mapping search based on successive halving. To reduce HW overfitting, we propose a HW robustness metric by relating a HW configuration's quality to its sensitivity in software mapping search, and quantitatively incorporate this metric to search for more robust HW design(s). We implement UNICO in open source accelerator platform, and compare it with the state-of-the-art solution HASCO. Experiments show that UNICO significantly outperforms HASCO; it finds design(s) with similar quality to HASCO up to 4× faster, and eventually converges to better and more robust designs. Finally, we deploy UNICO for optimizing an industrial accelerator, and show that it generates enhanced HW design(s) for key real-world DNNs.

*Equal Contribution

## CCS CONCEPTS

• **Hardware → Emerging tools and methodologies**.

## KEYWORDS

HW-SW Co-Design, Neural Network Accelerator, HW Robustness, Multi-Level Optimization

**ACM Reference Format:**
Bahador Rashidi, Chao Gao, Shan Lu, Zhisheng Wang, Chunhua Zhou, Di Niu, and Fengyu Sun. 2023. UNICO: Unified Hardware Software Co-Optimization for Robust Neural Network Acceleration. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3613424.3614282

## 1 INTRODUCTION

Deep neural networks (DNNs) [37] are pervasive nowadays, finding diverse applications in, e.g., computer vision [24], natural language processing [17], and autonomous driving[5]. DNNs are based on tensor computations, where tensors are data represented and processed in the form of multi-dimensional arrays. Typically, a deep neural network consists of multiple layers of tensor operators, where each operator performs multiple basic tensor computations, e.g., general matrix multiply (GEMM), general matrix-vector multiplication (GEMV), etc. Tensor computations are expensive. Thus, specialized hardware [8], i.e. *neural network accelerators*, have been designed to speed up DNN execution. These accelerators [8, 42, 48] deliver fast execution by taking advantage of parallel computation while preserving high energy efficiency. Theoretically speaking, there is an optimal accelerator architecture that best suits every specific deep neural network workload. In practice, however, considering the cost of chip design and corresponding tool-chain development, moderately general-purpose hardware is preferred. Consequently, the success of end-to-end AI acceleration hinges not only on hardware design but also on the effectiveness of software mapping compilation for the specific input DNN. For example, the hardware could be designed to execute a fixed-sized GEMM tensor computation; given a DNN model, it is the responsibility of *software mapping optimizer* to decide how to split the workload into sub-tasks and invoke corresponding GEMM intrinsics on hardware for best end-to-end efficiency. Therefore, the quality of software mapping optimization becomes another crucial factor for achieving the fast execution promised by the hardware. To ease this difficulty, deep learning compiler frameworks [6, 40] have been proposed, aiming at automatically synthesizing efficient mappings for different neural network models and AI accelerators.

While AI hardware and software stacks conventionally evolve separately, the solutions found may be suboptimal when jointly deployed, resulting in compounded end-to-end performance loss.

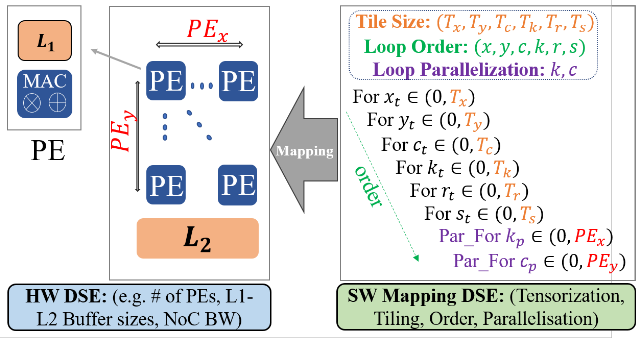
Recently, holistic approaches [33, 56, 64, 66] aspiring to jointly optimize both the hardware architecture and software mapping have been investigated. While this is a more appealing paradigm, the major challenge is that the space for hardware-software co-optimization can be gigantic. For example, it is estimated that addressing the bottlenecks of EfficientNet [58] by joint optimization would require an exploration of a large search space of $O(10^{2300})$ [66]. To counter this challenge, different solutions have been proposed to reduce the size of the search space, mainly focusing on 1) design space pruning or 2) design space approximation. For example, HASCO [64] uses a notion of unified IR to prune the search space; FAST [66] introduces several approximation techniques such that the design space exploration only needs to be done in a smaller and approximated space. Despite these efforts, the hardware and software design choices are still explored in isolation from an algorithmic perspective.

Moreover, new neural network architectures are emerging constantly, which may negate the effectiveness of hardware/software co-design obtained for specific prior DNN workloads.

In this paper, we propose a Unified Co-Optimization framework for AI accelerator co-design. UNICO approaches this bi-level large design space exploration in a symbiotic way such that it focuses on performing software exploration more for promising hardware candidates while discarding unfavorable hardware configurations progressively. In the meantime, UNICO is designed to find robust hardware configurations that can better generalize to new workloads unseen in the co-optimization. We show that by taking additional quantitative measures in software exploration, UNICO can alleviate the effect of *overfitting* hardware to input workloads as in prior approaches. Specifically, our contributions can be summarized as follows:

- We propose a batched hardware sampling strategy, to enable parallel hardware evaluation, yet guided by multi-objective Bayesian optimization (MOBO) with a surrogate model that is refined with high-fidelity data samples selected adaptively by a data-driven approach.
- We propose software mapping exploration with successive halving for sampled hardware configurations, with effective candidate promoting criterion, to speed up the HW-SW co-search process.
- We devise a method to enhance the generalization of hardware configurations to unseen workloads, by introducing an additional quantitative robustness measure into the co-optimization process.

We conduct extensive experiments on spatial accelerator co-design for a wide range of DNN workloads under edge and cloud device power constraints based on the open-source micro-architectural



$L_1$
MAC
PE
$PE_x$
$PE_y$
PE
PE
PE
PE
$L_2$
Mapping
Tile Size: $(T_x, T_y, T_c, T_k, T_r, T_s)$
Loop Order: $(x, y, c, k, r, s)$
Loop Parallelization: $k, c$
For $x_t \in (0, T_x)$
For $y_t \in (0, T_y)$
For $c_t \in (0, T_c)$
For $k_t \in (0, T_k)$
For $r_t \in (0, T_r)$
For $s_t \in (0, T_s)$
Par_For $k_p \in (0, PE_x)$
Par_For $c_p \in (0, PE_y)$
order
HW DSE: (e.g. # of PEs, L1-L2 Buffer sizes, NoC BW)
SW Mapping DSE: (Tensorization, Tiling, Order, Parallelisation)

**Figure 1: A typical 2D spatial accelerator HW design components (e.g. $(PE_x, PE_y)$, $L_1$ and $L_2$ buffer sizes)**

accelerator model MAESTRO [35]. We also perform HW-SW co-search on a cycle-accurate simulator for Ascend-like Architecture [42], where each evaluation of HW-SW co-design is costly. Experiments show that UNICO achieves consistently better performance on the identified Pareto front in terms of power, latency and area compared to the state-of-the-art methods, at a significantly smaller search cost. Moreover, the hardware discovered by UNICO by searching on multiple input workloads achieves better performance on a range of newer (in age and operators' dimensions) DNNs that are not involved in HW-SW co-optimization.

The rest of the paper is organized as follows. Section 2 presents the background and HW-SW co-optimization challenges. Section 3.2 explains details on how UNICO performs hardware design exploration using high-fidelity multi-objective Bayesian optimization (MOBO). Section 3.3 presents using early stopping rule for SW mapping exploration. Then, section 3.4 introduces the concept of hardware design generalizability. Moreover, section 3.5 explains parallel and scalable UNICO implementation details. Section 4 shows a variety of experimental studies to highlight the algorithmic implication and industrial competitiveness of UNICO. Finally, section 6 concludes the paper.

## 2 BACKGROUND

A number of studies have been conducted for designing customized hardware accelerators for providing real-time processing of deep neural networks [30, 31, 50]. These accelerators are mostly spatial in nature. They use an array of interconnected processing elements (PEs) for parallelism. The internal dataflow between the PEs is optimized via network-on-chips(NoCs) for efficient data reuse (e.g. input activations, weights, or output activations). Such a design reduces memory access and thus preserves high energy efficiency. Figure 1 illustrates a general design template of a typical 2D spatial accelerator where the key design choices are the number of processing elements (PE) in X and Y axis $(PE_x, PE_y)$, private scratchpad size $(L_1)$, global memory size $(L_2)$ and network-on-chip bandwidth (NoCBW).

Once HW design is fixed, for a given input DNN workload, the remaining task is optimizing SW mapping choices. DNN accelerators invariably expose a number of runtime parameters where the programmers have to explicitly manage how computation is scheduled both *spatially* and *temporally*. Indeed, it is known that different scheduling choices result in large variations in efficiency [28]. Traditionally, tensor SW mapping generation largely relies on manually optimized, high-performance tensor kernel libraries, such as cuDNN. However, these manual operator-level libraries development is not only laborious but also difficult to maintain as it demands timely updates whenever there is a change in the HW configuration.

To ease this difficulty, there have been auto-scheduling frameworks [6, 51] aiming to automatically synthesize efficient software mapping for various hardware targets. These frameworks assume DNNs as programs of domain-specific languages (DSLs), then introduce a set of optimization *primitives* where the compiler can translate the high-level DNN DSL into low-level code; the process is thus named as *scheduling*. For example, commonly used primitives for loop transformation include loop *split, reorder, fuse, and tiling*. As demonstrated in Figure 1, SW mapping space is composed by a particular set of scheduling primitives that can be applied in a specific order to the original loop representation such that the smallest computation unit (e.g. inner-most loop) can be mapped directly to certain HW resources *spatially* or *temporally* [35, 68]. Apparently, the best SW mapping choice depends on what HW topology and parameters are selected.

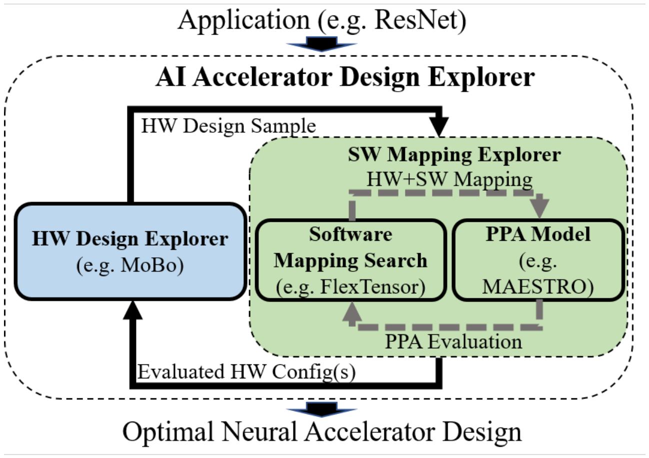


**Figure 2: Schematic illustration of HW/SW co-search.**

## 2.1 Formulation of HW-SW Co-optimization

We can intuitively formalize HW-SW co-design as a co-optimization such that the choice of HW parameters serves as the hyper-parameters for SW mapping exploration. Thus, the HW-SW co-design paradigm is a bi-level optimization, as the SW mapping choices are affected by selected HW config (e.g. #PEs, $L_1$ and $L_2$), and the latter must be sampled first such that it induces constraints for the SW mapping parameters search space. This sequential dependency naturally implies a bi-level optimization scheme shown in Fig. 2. In other words, for a given tensor workload $w$, a HW design configuration $h$ is sampled in the outer level and is passed to the inner level to find the best choice of SW mapping $s(w, h, b)$ given a search budget $b$. The inner-level SW mapping exploration shown in Fig. 2 can be done by adopting existing mature heuristic tools such that they aim to find a better SW Mapping $s(w, h, b)$ by minimizing an objective (e.g. latency and/or energy-delay-product (EDP)). An important characteristic of a mature SW mapping exploration tool is that given a search budget $b$, the objective loss is monotonically non-increasing during the course of the SW mapping search. For instance, FlexTensor[68] guides SW mapping choices by a Q-learning policy, and assigning a higher search budget $b$ for SW mapping exploration intrinsically improves the quality of selected mapping choice $s(w, h, b)$. Another choice of SW-mapping exploration tool is GAMMA [32] which utilizes an evolutionary genetic approach to iteratively find the best SW mapping choice.

## 2.2 Challenges to HW-SW Co-optimization

In practice, HW-SW co-design faces the following major challenges:

**Large co-optimization space:** The combined HW-SW space for optimization can be huge. Suppose HW design space is of $O_{HW}(.)$, and SW space is of $O_{SW}(.)$. Clearly, the joint space would be $\approx O_{HW}(.) \times O_{SW}(.)$. For a given convolution workload expressed as a 7D loop shown on the right side of Fig. 1, the corresponding unconstrained SW mapping space is of $O_{SW}(2^{60})$. For the spatial accelerator template as in Fig. 1, there are $O_{HW}(2^{22})$ hardware parameters. For commercial accelerators, assuming the overall architecture is fixed, the parameter choices of buffer capacities and PE array shapes can still be huge, e.g. for TPU [66], it is $O_{HW}(2^{44})$.

**Expensive HW-SW PPA evaluation:** During the course of SW mapping exploration shown in the inner level of Fig. 2, a power-performance-area (PPA) estimator is required to evaluate the quality of a given HW-SW candidate. Analytical cost models such as MAESTRO [35] and TimeLoop[49] are cheap to run and output PPA in order of milliseconds. However, when it comes to commercial custom-designed accelerators, cycle-accurate models (CAModels) (e.g. [42]) are typically utilized for PPA estimation that is highly time-costly (i.e. order of minutes) due to its simulation complexity. As a result, the combination of expensive PPA evaluation and the aforementioned large co-optimization space dictates a crucial need for an efficient unification of HW-SW co-optimization.

**Generalizing to unseen DNN workloads:** The other challenge that has not been well investigated in previous research is HW design generalization ability to unseen DNN workloads. We argue that this is a crucial issue since HW accelerators are typically designed w.r.t specific DNN workloads, such that the Pareto-optimal HW configurations achieve the best PPA for that time being for those specific workloads. However, the same HW design may not achieve top performance for future DNN models that are different in architecture and/or operator type, weights, and activation dimensions. This is particularly pressing for automatic HW design, since, in essence, the HW configurations being explored are invariably aimed to best fit the input DNNs. It is important to consider generality in HW-SW co-optimization from the co-search aspect, ensuring that the PPA optimization is not overfitting to a narrow set of DNNs.

## 3 UNICO: A UNIFIED AND ROBUST CO-OPTIMIZATION FRAMEWORK

In this section, we present the design of UNICO to address the aforementioned challenges. Fig. 3 illustrates the overall workflow of UNICO, which formalizes HW-SW co-search as a bi-level co-optimization task such that the choices of HW are the hyper-parameters and SW mapping exploration is assumed as an iterative evaluation procedure. How UNICO components interact with each other is described in Algorithm 1.

### 3.1 Notations

To facilitate exposition, we introduce the following notations. The input is a (DNN) workload $w$ for conducting HW-SW optimization. Let $h \in H$ be a hardware sample, and $s(w, h, b) \in S$ is the best SW mapping generated with search budget $b$ for workload $w$ and hardware $h$ using a SW mapping tool. The SW mapping search is driven by minimizing some cost function with an objective, e.g., latency: $l(s)\ \forall s \in S$, such that it is clear $\forall b_1 \leq b_2,\ l(s(w, h, b_1)) \geq l(s(w, h, b_2))$. That is, the software mapping search objective is **monotonic**. To ease presentation, in the remaining text, we drop $w$ when the context is clear.

### 3.2 HW Design Exploration

As shown in Algorithm 1, UNICO adopts a bi-level co-optimization strategy such that the MOBO guides the HW design space exploration (DSE) in outer loop. Following conventional MOBO [47], for the HW-SW co-optimization problem, the surrogate model inputs are HW design configurations and its outputs are co-optimization objectives that need to be minimized. We use the Gaussian Process (GP) as the surrogate model for MOBO. Each $y_j \in Y$ refers to a co-optimization objective in `(latency, power, area, sensitivity)`. In Section 3.4, we elaborate on how the `sensitivity` value is defined and how it is used for more robust HW search.

As shown in Algorithm 1 (line 1), in UNICO, the surrogate model is initialized randomly. Then, as in line 4, we sample a batch of $N$ hardware candidates $H_0$. Each HW is sampled with an acquisition function that balances *exploration* and *exploitation*, i.e., it recommends a HW with decreased objective value predicted by the surrogate model while taking into account the uncertainty of this prediction. From lines 5-9, UNICO performs adaptive SW mapping search for each hardware configuration in $H_0$. We explain the details in Section 3.3. At the end of each MOBO iteration, as in line 11, the surrogate model is updated with high-fidelity data points collected from that iteration, such that in the next MOBO iteration a better batch of hardware configurations can be sampled.

After one MOBO iteration, a major question is which HW-SW design samples among the batch of $N$ samples should be used for updating MOBO surrogate model, as we have to make a trade-off between the *amount* of selected samples and the *quality* of these samples. To help the selection, in UNICO, we collapse the multiple objectives $y_j \in Y$ into a single objective as in [34], and then choose top candidates based on a single fidelity objective, which is defined in below.

$$v_{\mathtt{ParEGO}} = \max_{j \in 1,2,3,4} (w_j y_j) + \rho Y^T W, \tag{1}$$

Here, each $w_j \in W$ is an importance weight for objective $y_j$, and $\sum_j w_j = 1$. Also, $\rho Y^T W$ (where $\rho = 0.2$ by default) is the weighted sum of all four objectives. We add this term to ensure the inclusion of all objectives in the scalar computation. Using this scalarization, as in line 10 of Algorithm 1, we empirically form a high-fidelity dataset $\mathcal{D}$ by selecting top hardware configurations among a batch of $N$ candidates, as illustrated by the bottom of Fig. 3. The following steps explain the details of our proposed **High Fidelity Update Rule**:

- **Step 1:** Given an objective vector $Y$ for each of the $N$ hardware configurations in every MOBO iteration, use Eq. (1) to calculate the fidelity scalar $v_{\mathtt{ParEGO}}$.
- **Step 2:** Let $v^{\mathtt{Best}}_{\mathtt{ParEGO}}$ be the smallest fidelity scalar value that has been seen thus far. Measure the $L2$-norm distance $d = ||v_{\mathtt{ParEGO}} - v^{\mathtt{Best}}_{\mathtt{ParEGO}}||_2$ for each hardware configuration.
- **Step 3:** Update surrogate model with high-fidelity hardware configurations that satisfy $d \leq UUL$ and add these $d$ to the set $\mathcal{D}$.
- **Step 4:** Recompute the Upper Update Limit ($UUL$) by the 95% percentile value of $\mathcal{D}$.

In particular, the upper update limit parameter $UUL$ gets updated at the end of each MOBO trial, which is then used as a new high-fidelity sample selection threshold for the next MOBO trial. In practice, $UUL$ tends to decrease over time, leading to a more strict selection criterion. This is desirable for MOBO, since we want more exploitation of high-quality samples as MOBO progresses.

### 3.3 Adaptive SW Mapping Search with Successive Halving

After a batch of $N$ hardware configurations are sampled, we need to call the software mapping search for each hardware $h$. Ideally, for hardware $h_1$ and $h_2$, if $h_1$ is superior to $h_2$, we would hope more search budget can be given to $h_1$ than $h_2$. In UNICO, as shown in Algorithm 1 (Line 2–9), we use *successive halving* (SH) [29] for this goal. However, in default SH, the succeeding candidates selection criterion is only based on terminal value (TV) at the end of the current round of budget $b_j$ such that only the best half candidates are selected for further exploration. For the software mapping search, we observe that the hardware configurations with relatively steep convergence rates are also likely to be promising, i.e., those steep-converging candidates should be given a second chance to be evaluated with higher budgets in the next round. Fig. 4 illustrates the difference between default SH and our modified successive halving (MSH) for SW mapping search. Specifically, We quantify the convergence rate of each hardware configuration $h$ by measuring the area under the curve (AUC) of its mapping history (see Fig. 4b for the definition of the AUC). In other words, hardware sample $h$ with higher AUC tends to converge faster, resulting in better final outcomes than those with relatively smaller AUC.

As in Fig. 4, given a batch of $N$ hardware configurations, SW mapping exploration is conducted in multiple rounds such that each time only a portion of candidates can survive to the next round. To identify the succeeding candidates at each round, both TV and AUC are used. Let $H_{\mathtt{TV}}$ represent the list of hardware configurations according to TV sorted in ascending order. Let $H_{\mathtt{AUC}}$ represent the list of hardware configurations according to AUC sorted in descending

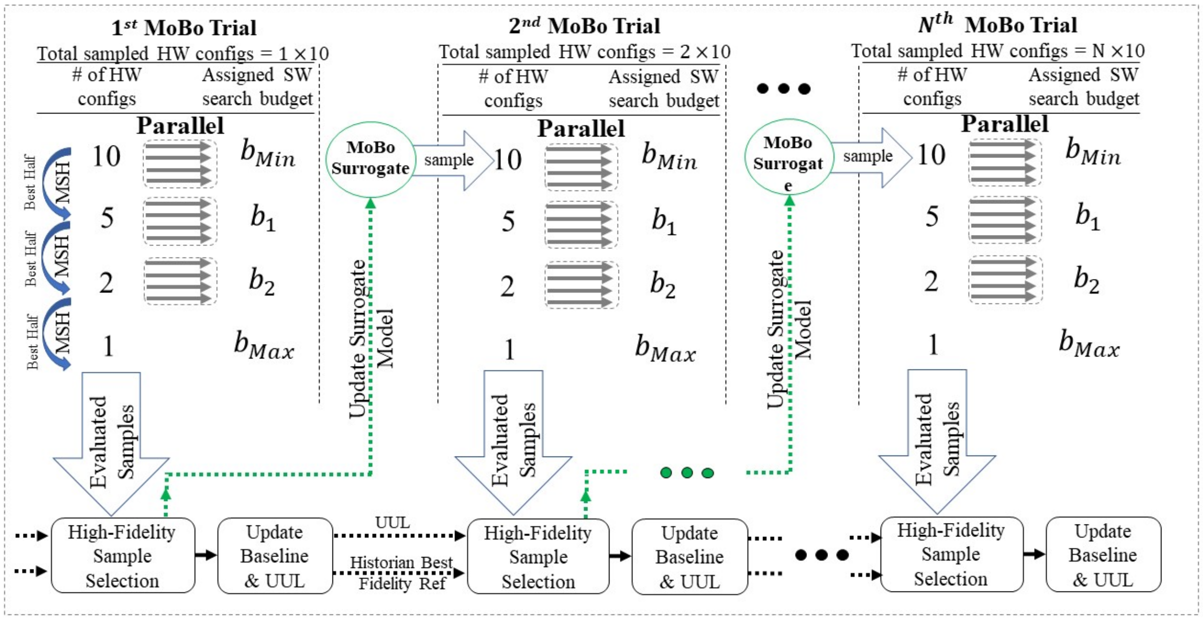

1st MoBo Trial
Total sampled HW configs = 1 ×10
# of HW configs
Assigned SW search budget
Parallel
2nd MoBo Trial
Total sampled HW configs = 2 ×10
Nth MoBo Trial
Total sampled HW configs = N ×10
MoBo Surrogate
sample
Best Half
MSH
Update Surrogate Model
Evaluated Samples
High-Fidelity Sample Selection
Update Baseline & UUL
UUL
Historian Best Fidelity Ref

**Figure 3: Implementation of UNICO with multi-objective Bayesian optimization and successive halving.**

**Algorithm 1:** UNICO

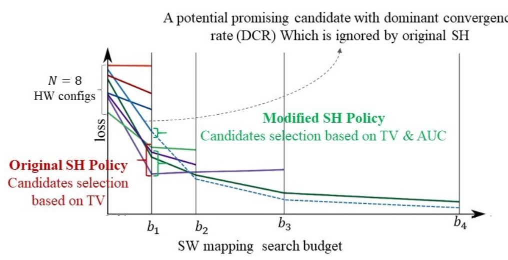
```
Input: $w$: workload
Output: Pareto-Front set of HW configurations $\mathcal{X}$
Parameters: $N$: HW batch size; $MaxIter$: maximum MOBO iterations; $b_{max}$: Maximum SW mapping search budget
1  Randomly initialize MOBO's Surrogate model
2  $b = b_{\max}\eta^{-\lfloor \log_\eta b_{\max} \rfloor}$
3  for $i \in \{1, \ldots, \text{MaxIter}\}$ do
4      $H_0 \leftarrow$ Sample a batch of $N$ HW configurations using Surrogate model
5      for $j \in \{1, ..., \lceil \log_2 N \rceil\}$ do
6          $b_j \leftarrow \lfloor b\eta^{-j} \rfloor$
7          for $h \in H_{j-1}$ parallel do
8              $\{s(h, b_1), ..., s(h, b_j)\} \leftarrow$ Software_Mapping_Search$(h, b_j)$
9          $H_j \leftarrow$ top $k$ HW samples from $H_{j-1}$ by assessing $\cup_{b=1,\ldots,b_j} s(h, b), \forall h \in H_{j-1}$
10     Form high-fidelity HW sample set $\mathcal{D}$ by assessing $s(h, b), \forall h \in H_0, \forall b \in [1, b_{\max}]$
11     Update Surrogate model using $\mathcal{D}$
12     Update HW Pareto Front $\mathcal{X}$
13 return HW Pareto Front $\mathcal{X}$
```

order. Then, we select top-$k$ succeeding hardware configurations as $H^k = H_{\text{TV}}^{(k-p)} \cup H_{\text{AUC}}^{(p)}$ subject to $H_{\text{TV}}^{(k-p)} \cap H_{\text{AUC}}^{(p)} = \emptyset$. According to this constraint, when we select top-$p$ candidates to form $H_{\text{AUC}}^{(p)}$, we ensure the same hardware candidate be not selected if it was already in $H_{\text{TV}}^{(k-p)}$. In this way, $H_{\text{AUC}}^{(p)}$ guarantees to always promote at least $p/N$ portion of hardware candidates to the next round with respect to convergence rate criterion.

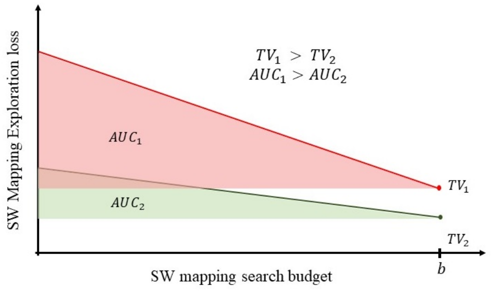
Essentially, our modified SH is a generalized version of the default SH, and would degenerate to the SH by setting $k = \lfloor 0.5N \rfloor$ and $p = \lfloor 0 \rfloor$. To balance the contribution of TV and AUC, for UNICO, we use $k = \lfloor 0.5N \rfloor$ and $p = \lfloor 0.15N \rfloor$ for all experiments.

## 3.4 Hardware Robust to SW Search and Unseen DNNs

To reduce HW over-fitting to specific workloads used during co-optimization, one has to consider two questions: **(1)** How to measure



(a) Illustration of candidate promoting criteria and comparison between SH and proposed MSH.



(b) AUC measures the area that is trapped between the curve and the horizontal line corresponding to the end loss value of the curve. For this toy example.

**Figure 4: A modified successive halving (MSH) that uses both AUC and terminal value to select candidates.**

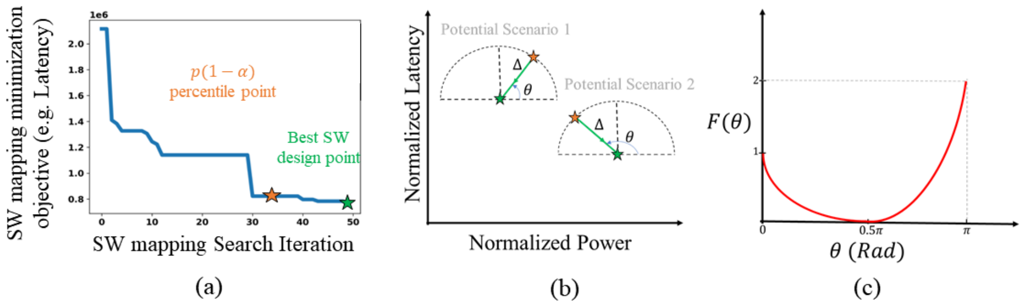


**Figure 5: (a) SW mapping objective convergence during search and indication of two promising SW mapping choices. The orange point is thought of as "sub-optimal", while the green one is "optimal" point found by SW search. (b) Illustration of latency and power for two hypothetical scenarios. (c) Analytical function $F(\theta)$ to quantify the power variation behaviour with respect to latency change.**

the robustness of hardware configurations which reflects its generalization to different SW mapping choices. **(2)** How to integrate such measurement into HW-SW co-optimization.

For question **(1)**, we design metric $R$ to quantify HW's robustness to SW search. Specifically, after performing parallel SW mapping search for the batch of $N$ hardware configurations using successive halving, for evaluation of each hardware $h$, what we obtain is a list of best mapping history at different budgets, i.e $\{s(h, b_1), s(h, b_2), \ldots, s(h, b_{\max})\}$ where $b_{\max}$ is the maximum budget eventually spent on a hardware configuration $h$. As an assessment of the quality of hardware samples, $Y(h, s(h, b_{\max}))$ provides a three-dimensional measurement of `(power, latency, area)` for a given hardware configuration $h$ with the best-found software mapping $s(h, b_{\max})$. However, this assessment on $h$ is arguably fragile since it evaluates $h$ solely by the best-seen software mapping and omits how the mapping optimization landscape looks during the mapping search. In other words, $Y$ contains no information on the correlation between performance and different promising SW mapping choices. We thus introduce a new metric for quantifying such a correlation. Specifically, we might say a hardware $h$ is robust to SW search when the performances (e.g., latency) have negligible variation with SW mappings at a range of different budgets.

Fig. 5(a) shows SW mapping optimization losses for a given hardware configuration during SW mapping search, where we call a SW mapping choice "optimal" if it is the final converged SW, and "sub-optimal" if the mapping objective value is the $(1-\alpha)$ (e.g. 95%) right-tail percentile of the whole loss history. In practice, SW search affects both latency and power and they do not always decrease simultaneously, thus our metric has to model the sensitivity of `latency change` and `power change` jointly. Therefore, we refer to a geometric formula for defining the sensitivity:

$$R = \Delta(1 + F(\theta)), \tag{2}$$

where $\Delta$ is the 2-norm distance of the two selected candidates, as in Fig. 5(a), and $\theta$ is the angle of the two mapping choices in the space of `latency` and `power` with reference to the horizontal line (as shown in Fig. 5(b)). It is clear that $\Delta = 0$ implies ideal robustness, i.e., $R = 0$, since this means there is no variation between *optimal* and *sub-optimal* SW choices for both `latency` and `power`.

If $\Delta > 0$, we use the second term $(1 + F(\theta))$ to penalize $\Delta$ by considering the relation of *latency change* and *power change*. To have more emphasis on `power change`, we design $F(\theta)$ as follows (shown in Fig. 5(c)).

$$F(\theta) = \frac{6}{\pi^2}\theta^2 - \frac{5}{\pi}\theta + 1.$$

Therefore, we see that when $\theta = \pi/2$, $(1 + F(\theta)) = 1$, thus $R = \Delta$. Consider two cases: (i) $\theta \in [0, \pi/2]$ and (ii) $\theta \in [\pi/2, \pi]$. For (i), $(1 + F(\theta))$ decreases from $2\Delta$ to $\Delta$ as $\theta$ increases. For (ii), $(1 + F(\theta))$ increases from $\Delta$ to $3\Delta$ as $\theta$ increases. The asymmetry of Fig. 5(c) implies that our design prefers $0 \leq \theta \leq \pi/2$ more than $\pi/2 \leq \theta \leq \pi$. The intuition is that, for the latter case, from point "orange" to "green", power is increased, which is less favorable than the other case (first circle quarter) when both latency and power decrease.

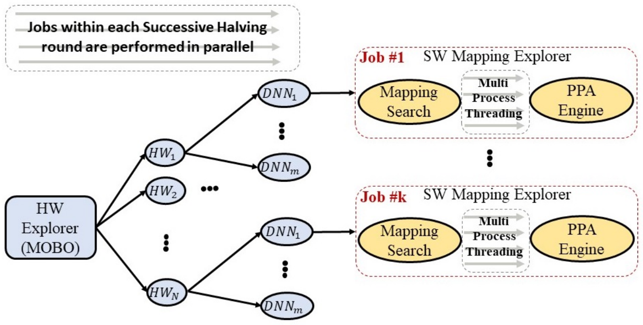
For question **(2)**, we compute `sensitivity` $R$ at the end of each MOBO trial for each hardware configuration in the sampled batch. Then, we include $R$ as the fourth dimension objective, i.e., $Y = (\texttt{latency}, \texttt{power}, \texttt{area}, \texttt{sensitivity})$. In this way, the impact of metric $R$ on HW-SW co-optimization is twofold: First, MOBO surrogate model considers $R$ as an optimization target and learns to sample hardware configurations with less $R$ value as MOBO iterations advance. Second, the MOBO surrogate update mechanism incorporates $R$ in Eq. (1) when generating scalar $v_{\texttt{ParEGO}}$. That is, we also refer to $R$ for selecting high-fidelity samples for updating the surrogate model. This further encourages the MOBO to learn to sample hardware configuration that is more robust to SW search. By reducing HW sensitivity, we speculate that the HW being generated would also yield better generalization on unseen workloads that are not used for co-optimization.

## 3.5 Scalable and Parallel Implementation

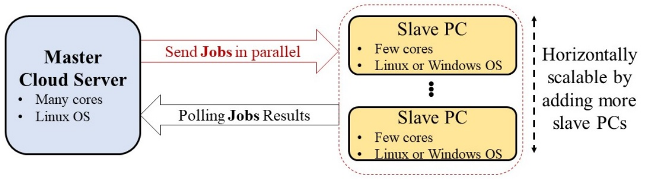
To enhance practical applicability, we describe how to implement UNICO using parallel computation. This is particularly important when the co-optimization space is vast and when `(power, latency, area)` estimation is time-consuming. Fig. 6a shows our modular implementation which consists of the following components:

- **HW Design Explorer:** A module responsible for hardware configuration sampling and multi-objective optimization, which calls the proposed high-fidelity MOBO algorithm as an internal API.
- **SW Mapping Explorer:** An internal program to call to perform a SW mapping search for a given batch of hardware configurations. FlexTensor [68] and GAMMA [32] are examples for this component in open-source spatial accelerator design environments.
- **PPA Estimation Engine:** A standalone REST API to call which requires hardware configuration, SW mapping configuration, and a tensor workload as inputs to estimate performance, power and area. This component can be an analytical model such as MAESTRO [35] or TimeLoop [49]. These models can be used in the early stage for prototyping. In later-stage production, simulation-based cycle-accurate models (CAModels) for specific industrial architecture (e.g., Ascend [42]) might be used. CAModels can be orders of magnitude slower than analytical models but are more accurate.

As in Fig. 3, for each MOBO trial, SW mapping search needs to be performed for a sampled hardware configuration sequentially in multiple successive halving rounds. Within each successive halving round, we run standalone `Jobs` via multi-processing in parallel, where each `job` handles the SW mapping search (in that successive halving round) for a selected hardware configuration on a specific DNN workload. To maximize the parallelization, depending on the choice of SW mapping search tool, another level of parallelization on software search itself might be used for fast exploration of SW mapping space.


Jobs within each Successive Halving round are performed in parallel
HW Explorer (MOBO)
$HW_1$
$HW_2$
$HW_N$
$DNN_1$
$DNN_m$
Job #1 SW Mapping Explorer
Job #k SW Mapping Explorer
Mapping Search
Multi Process Threading
PPA Engine

**(a) Parallel implementation of UNICO algorithm to support multi-workload HW-SW co-optimization.**



**(b) Computation distribution strategy for UNICO major components.**

**Figure 6: UNICO parallel and scalable implementation.**

Another important feature in the implementation of UNICO is its scalability in terms of computational power. We propose to utilize a master-slave architecture shown in Fig. 6b, which distributes different components of UNICO on different machines. In particular, high-fidelity MOBO algorithm (i.e., the HW explorer) and other miscellaneous computation tasks are placed on cloud machines with high computing power while SW exploration `Jobs` can be distributed to slave machines with relatively lower computing power. As a result, UNICO allows adding additional slave machines to be able to run more SW search `Jobs` in parallel to achieve scalability.

In summary, UNICO offers a co-optimization algorithm framework consisting of three major new features: (1) an enhanced MOBO with enhanced surrogate modeling learning for HW sampling; (2) a customized successive halving for more efficient software mapping search; (3) a novel sensitivity metric $R$ for reducing HW over-fitting. Therefore, UNICO can easily be applied for different platforms with specific *hardware design template*, *software mapping search tool*, and *PPA estimation engine*. In application, how scalable UNICO can be implemented is bounded by various pragmatic factors, such as the computation power or memory of the master machine, the communication latency between master and slave machines, the expensiveness of software mapping search and PPA evaluation. Practitioners should refer to specific characteristics of their platform and computation resources for the best deployment efficiency of UNICO.

## 4 EXPERIMENTS

To show the effectiveness of our approach, we deploy UNICO on both an open-source accelerator template [64] and an Ascend-like commercial architecture used in edge devices [42].

Specifically, we compare UNICO's performance against the state-of-the-art HW-SW co-optimization methods from the following aspects: **(1)** efficiency improvement due to the use of batch sampling, high-fidelity MOBO HW explorations, and parallel SW mapping search with proposed successive halving, which has been described in Section 4.2 and 4.5; **(2)** generalizing capability of hardware to unseen DNN workloads because of the newly introduced robustness measures, which has been presented in Sections 4.3 and 4.4; and **3)** how UNICO can be used to improve the performance of an existing industrial-class Ascend-like [42] architecture, which has been shown in Section 4.6.

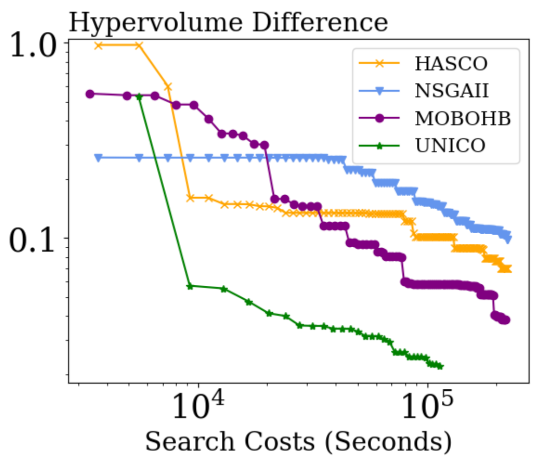
### 4.1 Experimental Setup

**Open-Source Platform:** For open source accelerator experiments, the hardware search space includes the *PE* array shape (the numbers of *PE* on the x and y axes ranging from 1 × 1 to 24 × 24), $L_1 \in \{2^i \times 3^j\}_{i,j=0,...,10}$ bytes, and $L_2 \in \{2^i \times 3^j\}_{i,j=0,...,10}$ KB buffer sizes and NoC bandwidth ∈ {64, 128}. As the hardware intrinsic is set to GEMMCore, we also search for the dataflow style which can be weight-stationary or output-stationary. As in HASCO [64], two scenarios, *edge* and *cloud*, are considered in our experimentation, they have HW design spaces $10^5$ and $10^9$, respectively. For the software mapping search technique, we employ out-of-box FlexTensor [68] to search for the optimal software mapping in the search space including split, reorder, unroll, inline, etc. The SW mapping space is around $10^6$ for one layer of a neural network. Given a DNN of $n$ layers, the joint HW-SW spaces are around $n \cdot 10^{11}$ and $n \cdot 10^{15}$ for *edge* and *cloud*, respectively. For the PPA estimation engine, we adopt MAESTRO [35], which is an open-source analytical framework to model the dataflows of neural networks and predict the latency and power consumption of the given hardware configuration. These settings are exactly the same as those used in HASCO [64].

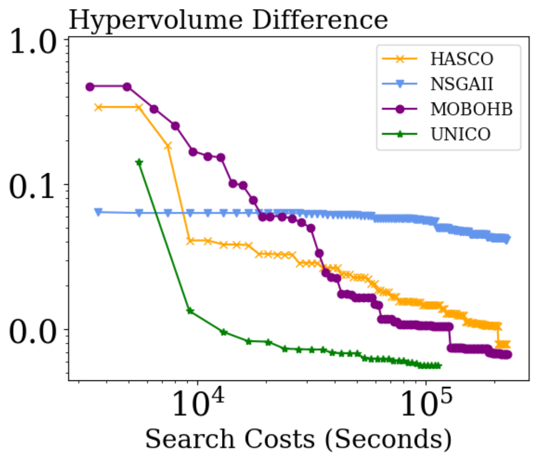
**Ascend-like Platform:** For Ascend-like platform, the hardware configuration search space includes the buffer sizes and bank groups for each of $L_0A, L_0B, L_0C, L1$, vector buffers, parameter buffer, the ICache size, and $MNK$ cube parameters. They compose a HW space of size $10^9$. For SW mapping exploration tool, we use a depth-first buffer fusion search technique similar to those in [23, 45, 55, 63] to search for SW mapping configurations with respect to a given search budget $b$. The SW mapping search is typically around $10^{10}$, resulting in a large joint HW-SW space around $10^{19}$. PPA estimation is done by a cycle-accurate model (CAmodel) with a benchmarked simulation error of 8 ± 3%. The CAmodel takes the workload DNN according to a compiled SW mapping choice along with a particular hardware configuration and returns profiled PPAs. In comparison to MAESTRO which takes seconds to output PPAs, Ascend-like CAmodel wall-clock time is highly expensive (e.g., ranging from 2-10 minutes) due to its topological complexity. Hence, the effect of co-search time efficiency and its fast convergence is even more crucial.

**HW/SW Design Feasibility:** UNICO is more of a co-optimization algorithm framework, rather than a brand-new co-design system for a specific HW template or architecture. When comparing with HASCO, we deploy UNICO to the same open source platform as HASCO, indicating that they share the same HW and SW joint space, and UNICO uses the same *space pruning* and *constraint* for addressing the feasibility issues in HW/SW co-design search. When deploying to industrial accelerators, we follow the same scheme: we build UNICO on expert-crafted HW design space and rely on an existing mature software mapping tool for software optimization.

**Computation Resources:** For experiments on open source accelerator, we run both UNICO and HASCO on the same Intel Xeon CPU server. For experiments on Ascend-like, we run the distributed version of UNICO with the same server as the *master* with 4 more *slave* workstations for conducting software mapping search and PPA evaluation.



**(a) Edge Device**



**(b) Cloud Device**

**Figure 7: Comparisons of HASCO, NSGAII, MOBOHB and UNICO in terms of Hypervolume Difference. UNICO uses batch size $N = 30$ and $b_{max} = 300$. The $x$-axis is the search cost measured in wall-clock time on the same server machine.**

**Table 1: Comparisons of HASCO, NSGAII and UNICO on the Edge Device (Power < 2W).**

| Networks | HASCO | | NSGAII | | UNICO | |
|---|---|---|---|---|---|---|
| | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* |
| *Bert* | 0.001082, 270.2, 5.0 | 35.5 | 0.000644, 317.9, 6.5 | 35.5 | 0.000341, 149.7, 3.4 | **7.48** |
| *MobileNet* | 2.5, 317.9, 6.5 | 155.5 | 2.2, 182.4, 3.4 | 85.75 | 1.4, 161.4, 3.3 | **31.4** |
| *ResNet* | 33.3, 244.3, 1.7 | 105.5 | 9.6, 322.5, 3.1 | 76.45 | 8.1, 128.5, 2.1 | **21.43** |
| *SRGAN* | 134.9, 381.4, 6.6 | 145.5 | 278.3, 317.9, 6.5 | 44 | 109.4, 155.2, 3.7 | **29.4** |
| *UNet* | 270.0, 293.6, 3.4 | 100.5 | 275.5, 353.1, 3.4 | 52.17 | 85.0, 148.6, 2.8 | **20.43** |
| *VIT* | 409.2, 173.4, 2.3 | 35.5 | 1432.6, 187.1, 1.9 | 35.5 | 322.3, 132.7, 2.0 | **7.48** |
| *Xception* | 8.5, 429.0, 4.3 | 130.5 | 12.7, 422.8, 4.1 | 80.67 | 4.1, 154.3, 3.7 | **26.41** |

**Table 2: Comparisons of HASCO, NSGAII and UNICO on the Cloud Device (Power < 20W).**

| Networks | HASCO | | NSGAII | | UNICO | |
|---|---|---|---|---|---|---|
| | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* | *L(Ms),P(mW),A($mm^2$)* | *Cost(h)* |
| *Bert* | 0.000186, 785.5, 18.2 | 35.5 | 0.000134, 820.3, 17.7 | 35.5 | 0.000127, 618.5, 14.9 | **7.48** |
| *MobileNet* | 0.685, 659.0, 15.8 | 155.5 | 1.5, 1174.0, 15.0 | 107.45 | 0.531, 638.7, 14.4 | **39.63** |
| *ResNet* | 4.1, 711.9, 14.1 | 105.5 | 4.56, 958.8, 15.0 | 91.5 | 2.59, 658.1, 14.7 | **28.41** |
| *SRGAN* | 69.98, 797.5, 18.2 | 145.5 | 51.63, 658.7, 15.8 | 45.45 | 41.13, 639.4, 14.4 | **29.4** |
| *UNet* | 98.6, 534.3, 11.4 | 100.5 | 58.8, 533.1, 9.3 | 52.5 | 29.7, 359.6, 8.5 | **24.42** |
| *VIT* | 171.3, 317.9, 6.5 | 35.5 | 171.3, 317.9, 6.5 | 35.5 | 155.6, 287.0, 2.9 | **13.29** |
| *Xception* | 4.9, 444.1, 6.1 | 130.5 | 5.7, 509.8, 7.1 | 88.03 | 4.1, 383.8, 6.6 | **26.41** |

## 4.2 Performance on Open-Source Accelerator

We compare the performance of UNICO with the state-of-the-art open source co-design framework HASCO [64] and NSGAII[13] on individual networks (BERT [14], MobileNet [26], ResNet [24], SRGAN [38], UNET [52], VIT [17], Xception [11]) under edge (power $\leq 2W$) and cloud (power $\leq 20W$) constraints. To demonstrate the end-to-end performance of each co-search approach, we compare the PPA (`latency`, `power`, `area`) performance of the hardware configuration which achieves the *min-Euclidean-distance* to the origin on PPA Pareto-Front set.

For the edge device, as shown from Table 1, UNICO achieves strictly better PPA performance compared to HASCO and NSGAII on all listed networks except for ResNet and VIT. For ResNet, UNICO achieves strictly better PPA performance compared to NSGAII. Besides, compared to HASCO, UNICO sacrifices area by a small amount while achieving 75.7% and 47.4% improvements in latency and power, respectively. Moreover, for VIT, compared to NSGAII, UNICO sacrifices area by 5.3% while achieving 77.5% and 29.1% improvements in latency and power, respectively. In addition, UNICO achieves strictly better PPA performance compared to HASCO on VIT.

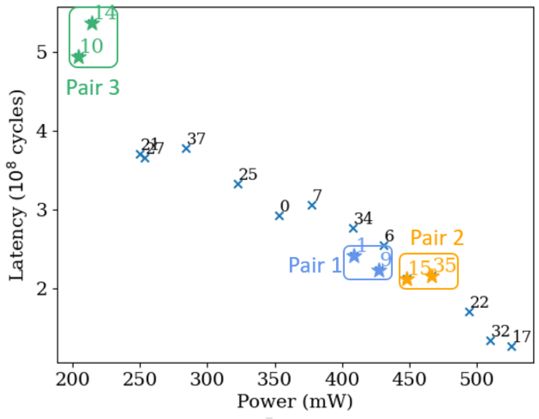
Similarly, for the cloud device, as shown from Table 2, UNICO achieves strictly better PPA performance compared to NSGAII on all listed networks. While compared to HASCO, UNICO achieves strictly better PPA performance on all listed networks except for ResNet and Xception. However, similar to the performance comparison on the edge device, UNICO sacrifices one PPA metric by a small amount, while achieving large gains on the other two PPA metrics. Specifically, for ResNet, UNICO sacrifices area by 4.2% while achieving 36.8% and 7.5% improvements in latency and power. Moreover, for Xception, UNICO sacrifices area by 8.2% while achieving 16.3% and 13.6% improvements in latency and power.

More importantly, the search costs of UNICO are noticeably smaller than HASCO and NSGAII across all listed networks. Therefore, UNICO can achieve better or comparable performance within a much smaller search time than HASCO and NSGAII. In addition to HASCO and NSGAII, we also implement a multi-objective version of BOHB [18]. Figures 7a and 7b show the hypervolume difference across the tested networks on edge and cloud devices, respectively. Clearly, UNICO shows faster search convergence due to its advantageous batch hardware sampling and high-fidelity MOBO surrogate update. In particular, compared with MOBOHB, which also uses successive halving, UNICO still shows superior convergence behavior, indicating that our customized successive halving is indeed a more suited algorithm for co-optimizing DNNs.

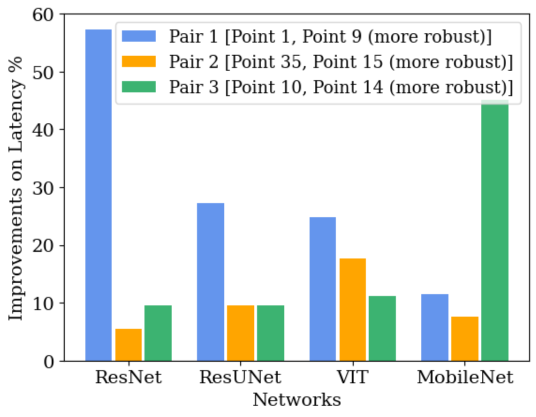
## 4.3 How Reliable is Metric $R$ as an Indicator for HW Generalization?

We empirically investigate whether metric $R$ can be a valid indicator for HW robustness. The experiment works as follows: (1) we run UNICO without `sensitivity` metric $R$ on a set of *training* DNNs ({UNET, SRGAN, BERT}); (2) we select pairs of Pareto fronts (HW) having similar PPAs; (3) we then compute the $R$ values for each HW point in a pair; (4) for each HW in a pair, we validate its PPAs on another set of *validation* DNNs ({ResNet [24], ResUNet [15], VIT [17], MobileNet [27]}). (5) for each pair HW, we check the correlation of $R$ values and their performance on the validation DNNs.

Fig. 8a illustrates the obtained Pareto-Front set of hardware configurations w.r.t power and latency. Then, among all design points, we select three different hardware configurations pairs (1, 9), (35, 15) and (10,14) such that each pair design points PPA have no larger than 10% difference collectively over the mentioned three training DNN workloads, and compute the sensitivity metric as



**(a) HW sample Pareto front, with three comparable pairs.**



**(b) Latency comparison on unseen DNNs for P1, P2 and P3.**

**Figure 8: Demonstration on how the selected more robust HW designs perform better than less robust HW samples on four unseen DNNs. The *x*-axis is the search cost measured in wall-clock time on the same server machine.**

in Eq. 2 for each design point within a pair and show the relative *robustness* (i.e., smaller $R$) in Fig. 8b (shown in legend).

For each of the three pairs, we perform individual SW mapping searches for each of the hardware configuration points on the validation workload set. As in Fig. 8b, even though the latency performance of Point 1 (more robust) on the training networks is 7.8% worse than that of Point 9 (less robust), Point 1 achieved 28.5% better latency performance on average across the validation networks. similarly for the other pairs (35, 15) and (10, 14) — the more robust design points, Point 35 and Point 14, resulted in lower average latency across all unseen workloads by 10% and 18% with respect to their less robust rivals, respectively. This shows our proposed robustness measurement can be a valid indicator of the generalization ability of hardware configurations.

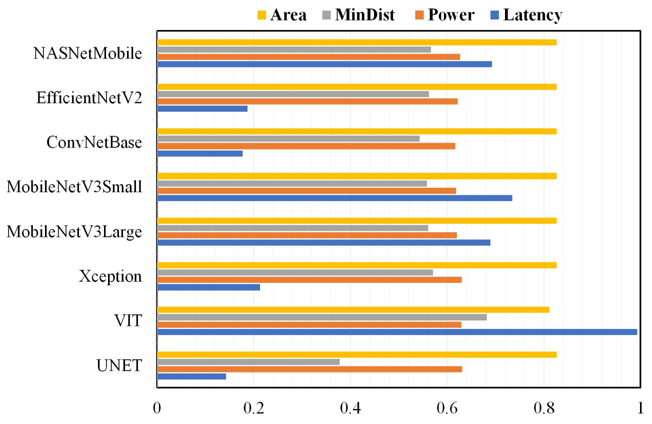


**Figure 9: UNICO vs HASCO comparison on generalization. Co-Optimization performed on 4 DNNs { MobileNetV2, SRGAN, ResNet, VGG}, obtained HW is tested on 8 new DNNs as in y-axis. The gain ratio achieved by UNICO w.r.t HASCO is shown on the x-axis.**

## 4.4 Comparison with HASCO on Generalization to Unseen DNNs

To see if UNICO can obtain better generalization for unseen applications with the `sensitivity` metric, we run UNICO and compare with HASCO [64] by conducting the co-optimization on a set of *training* workloads, and then directly apply the best-found hardware configurations to new and unseen applications. Specifically, the training networks consist of four networks (i.e. MobileNetV2, ResNet, SRGAN and VGG), and we obtain the best hardware configuration w.r.t the *min-Euclidean-distance*. Selected hardware configuration for each approach is directly adopted to perform individual SW mapping search on a validation set consisting of eight new networks: UNET, VIT, Xception, MobileNetV3 [25] (large and small), NASNetMobile [70], EfficientNetV2 [59], ConvNeXt [44].

The best achieved PPAs from software mapping are reported in Figure 9. On average for all validation networks, UNICO improves the *min-Euclidean-distance* of HASCO by 44%. Considering that, for UNICO, the metric $R$ is not only an additional MOBO optimization objective but also being used in selecting high-fidelity hardware configurations for MOBO surrogate model update and learning — this feature further enables UNICO to harness the full strength of the `sensitivity` measurement and improves its generalization ability on unseen DNN workloads.

## 4.5 Feature Contributions in UNICO

Recall that, UNICO employs two new ideas for improving the optimization: (1) enhanced surrogate model update rule by collecting high-fidelity samples; (2) customized successive halving for more effective adaptive search. To see how these two features contributed to the overall performance of UNICO, we create two new variants of UNICO: **SH + ChampionUpdate** — it uses unmodified successive halving with vanilla MOBO surrogate update rule, i.e., selecting only the best sample; **MSH + ChampionUpdate** — it uses our modified successive halving. UNICO is thus equivalent to **MSH**

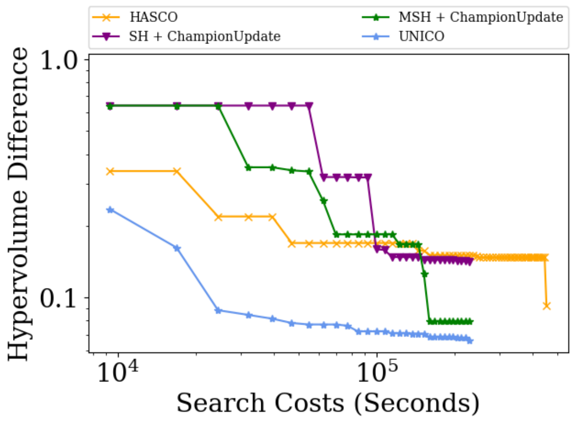


**Figure 10: Comparisons of HASCO, UNICO, and each of our proposed components in terms of Hypervolume Difference.**

+ **HighFidelityUpdate** yet with a robustness metric. We run co-search experiments on multiple DNNs, i.e., UNET, SRGAN, BERT and VIT. The experimentation configuration is the same as those in Section 4.2.

Figure 10 shows the performance comparison of HASCO, UNICO, and two other ablation scenarios in terms of the hypervolume — a commonly used metric for measuring the convergence behaviour of multi-objective optimization. HASCO can be viewed as *ChampionUpdate without SH* — this explains the slower convergence of HASCO. This result shows that the early stopping methods for batch software mapping search (i.e., due to SH or MSH) can improve the convergence speed of hardware-software co-optimization. However, the **SH + ChampionUpdate** obtained an overall performance worse than HASCO. This is because, in software mapping search, SH stops the worse-performing configurations too early, leading to too aggressive pruning on promising configurations for the HW. By using the MSH, we obtained improved performance approximately 13.7% better than HASCO. In comparison to **SH + ChampionUpdate**, replacing SH with MSH leads to an improvement of approximately 16%. Overall, the full implementation of UNICO with *MSH + HighFidelityUpdate* leads to a performance improvement by approximately 28% better than HASCO. These comparisons show the individual benefit of the two new algorithmic features in UNICO.

## 4.6 Deployment of UNICO for Ascend-like

We compare the performance of UNICO-found architecture with default Ascend-like architecture selected by domain experts on individual networks UNET [52], FSRCNN [16] with different resolutions, and Deep Learning image Enhancement and Upscaling technologies (DLEU) [1]. In particular, to UNICO HW-SW co-optimization, HW config batch size is $N = 8$ and MoBo iteration is $MaxIter = 30$. Also, the SW mapping exploration maximum budget is set to $b_{max} = 200$. The co-optimization goal here are reducing both latency and power consumption while not exceeding the edge-device chip design area constraint of 200 $mm^2$. Fig. 11 illustrates latency and power relative percentage reduction of the hardware design found by UNICO over Ascend-like default hardware configuration. As shown in this plot, on UNET and FSRCNN with 120x320 resolution, UNICO improves the latency by 12.1% and 26.4%, respectively.

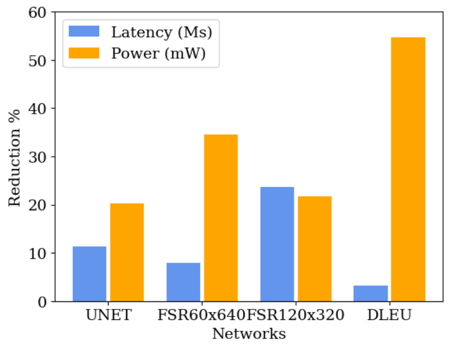


**Figure 11: Latency and power savings of UNICO architecture compared with Ascend-like architecture on latency and power, evaluated by CAModel simulation.**

More importantly, compared to Ascend-like architecture, UNICO-found architecture achieves better power performance by 32.3% on average across the listed networks. An interesting discovery revealed by UNICO is that it suggests reducing $L0B$ and $L0C$ and increasing $L0A$ w.r.t their default buffer sizes, respectively. While the default values of these are simply set by engineers by referring to cube parameters. From these results, we can see that UNICO's high-fidelity MOBO surrogate update mechanism and its concurrent exploration strategy have led to the discovery of hardware configurations with better speed and lower power consumption.

## 5 RELATED WORK

**Spatial Accelerators.** Many spatial accelerators have been developed and their hardware architecture is of great importance. Some accelerators follow a rigid architecture ( e.g. Eyeriss [9]) that supports only a fixed type of computation pattern. Some are more flexible and support a large range of tensor operations (e.g. Eyeriss v2 [10], MAERI [36]). Some aggregate multiple chiplet as one accelerator hardware (e.g. SIMBA [54]). The commercial accelerators typically have additional and more complex memory and processing element (PE) hierarchy. For example, TPU [48] allocates dedicated buffers to weights and output with sophisticated synchronization. Huawei's *DaVinci* core [42] has 3 different PE types (i.e. scalar unit, vector unit and 3D cube unit) and employs a more complex network on chip and memory hierarchy for data/computation orchestration.

**Software Mapping.** Given a fixed hardware, the scheduling space for a DNN layer can still have billions of valid candidates. Thus, many algorithms have been developed for the schedule optimization problem. The static cost model-based search [4, 12, 22, 32, 49, 65, 68] typically use an analytical model to estimate and compare different schedule candidates. Together with manual pruning, the schedule exploration algorithms try to identify the best candidate judged by the static cost model. The feedback-based search [7, 20, 21, 51, 67] approaches assume that the hardware is fixed and try to construct a learning-based cost model by collecting real hardware evaluation data through search. The search algorithm is then further guided with the refined cost model. This approach alleviates the burden of constructing an analytical model by human experts, relying on feedback from the real hardware for cost model learning and search.

**PPA Estimation.** The PPA estimation engine is a crucial component of UNICO. Various models have been proposed to estimate PPA from internal metrics such as data-reuse, # of FLOPS, minimum required buffers, etc. MAESTRO [35] models spatial accelerator architecture from a data-centric perspective. TimeLoop [49] estimates PPA from a loop-centric perspective by analyzing loop computation and data movements. For commercial accelerators, slower yet more accurate cycle-accurate models (CAModel) are often used [46, 53, 60, 62]. They usually provide more accurate estimations at the expense of being orders of magnitude slower in speed.

**Hyperparameter optimization.** Both Bayesian optimization (BO) [57] and successive halving (SH) [29] have been used for hyperparameter optimization, but with different focuses. The former aims at adaptive configuration selection while the latter pays more attention to adaptive resource allocation and early stopping. HYPERBAND [39] wraps over SH by essentially performing a grid search for addressing the "n versus B/n" question in SH. BOHB [19] combines HYPERBAND with BO, aspiring to achieve both strong anytime performance and optimal convergence. UNICO resembles BOHB and MFES-HB [41] in spirit but is specially designed for the application of HW/SW co-optimization.

**HW/SW co-design with NAS.** This line of research [2, 61, 69] combines neural architecture search (NAS) with HW-SW co-search. For instance, NAAS [43] proposes to include an extra level of neural architecture search based on Once-For-All-NAS [3]; then, for the given DNN architectures, it performs HW-SW co-design using evolutionary search.

## 6 CONCLUSION

In this paper, we present UNICO, a HW-SW co-optimization framework for DNNs aimed at boosting the efficiency of exploring the vast HW-SW design space for neural accelerators. We propose a batched hardware sampling strategy guided by multi-objective Bayesian optimization (MOBO), and enable parallel hardware evaluation and software mapping search through a customized successive halving algorithm. To enhance the generalization of HW candidates, we have designed a robustness metric that is incorporated into multi-objective Bayesian optimization to guide HW sampling. We have applied our approach to both open-source and commercial architectures. Experimental results confirm that our algorithm can generate better solutions than existing methods with lower search costs while generalizing well to new applications that are unseen in the co-optimization process.

## REFERENCES

[1] [n. d.]. Nvidia GeForce, USA. NVIDIA DLSS 2.0 | A Big Leap in AI Rendering. (Mar. 23, 2020). Accessed: Sept. 20, 2020. [Online Video]. Available: https://youtube.com/watch?v=-X1RtXCvPFQ. https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/. Accessed: 2020-09-20.

[2] Mohamed S. Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2020. Best of Both Worlds: AutoML Codesign of a CNN and Its Hardware Accelerator. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 192, 6 pages.

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.

[4] Prasanth Chatarasi, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, Tushar Krishna, and Vivek Sarkar. 2021. Marvel: a data-centric approach for mapping deep learning operators on spatial accelerators. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 1 (2021), 1–26.

[5] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*. 2722–2730.

[6] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[7] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. Learning to Optimize Tensor Programs. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/8b5700012be65c9da25f49408d959ca0-Paper.pdf

[8] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao family: energy-efficient hardware accelerators for machine learning. *Commun. ACM* 59, 11 (2016), 105–112.

[9] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 367–379.

[10] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

[11] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[12] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. 2019. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 5s (2019), 1–27.

[13] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[15] Foivos I Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. 2020. ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (2020), 94–114.

[16] Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*. Springer, 391–407.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy

[18] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.

[19] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*. PMLR, 1437–1446.

[20] Chao Gao, Jingwei Chen, Tong Mo, Tanvir Sajed, Shangling Jui, Min Qin, Laiyuan Gong, and Wei Lu. 2022. A Memory-Bounded Best-First Beam Search and Its Application to Scheduling Halide Programs. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 15. 74–82.

[21] Chao Gao, Tong Mo, Taylor Zowtuk, Tanvir Sajed, Laiyuan Gong, Hanxuan Chen, Shangling Jui, and Wei Lu. 2021. Bansor: Improving Tensor Program Auto-Scheduling with Bandit Based Reinforcement Learning. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 273–278.

[22] Hasan Genç, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Charles Wright, Colin Schmidt, Jerry Zhao, Albert J. Ou, Max Banister,

Yakun Sophia Shao, Borivoje Nikolić, Ion Stoica, and Krste Asanović. 2019. Gemmini: An Agile Systolic Array Generator Enabling Systematic Evaluations of Deep-Learning Architectures. *ArXiv* abs/1911.09925 (2019).

[23] Koen Goetschalckx and Marian Verhelst. 2019. Breaking High-Resolution CNN Bandwidth Barriers With Enhanced Depth-First Execution. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 323–331. https://doi.org/10.1109/JETCAS.2019.2905361

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1314–1324.

[26] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[27] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[28] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. 2021. Cosa: Scheduling by constrained optimization for spatial accelerators. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 554–566.

[29] Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*. PMLR, 240–248.

[30] Liancheng Jia, Zizhang Luo, Liqiang Lu, and Yun Liang. 2021. TensorLib: A Spatial Accelerator Generation Framework for Tensor Algebra. *CoRR* abs/2104.12339 (2021). https://arxiv.org/abs/2104.12339

[31] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 1–12.

[32] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[33] Sheng-Chun Kao, Michael Pellauer, Angshuman Parashar, and Tushar Krishna. 2022. DiGamma: domain-aware genetic algorithm for HW-mapping co-optimization for DNN accelerators. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 232–237.

[34] J. Knowles. 2006. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation* 10, 1 (2006), 50–66.

[35] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. 2020. MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings. *IEEE Micro* 40, 3 (2020), 20–29. https://doi.org/10.1109/MM.2020.2985963

[36] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices* 53, 2 (2018), 461–475.

[37] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[38] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4681–4690.

[39] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 18, 1 (jan 2017), 6765–6816.

[40] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. 2020. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 708–727.

[41] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. 2021. MFES-HB: Efficient Hyperband with Multi-Fidelity Quality Measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8491–8500.

[42] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. 2021. Ascend: a Scalable and Unified Architecture for Ubiquitous Deep Neural Network Computing : Industry Track Paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 789–801. https://doi.org/10.1109/HPCA51647.2021.00071

[43] Yujun Lin, Mengtian Yang, and Song Han. 2021. NAAS: Neural Accelerator Architecture Search. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1051–1056. https://doi.org/10.1109/DAC18074.2021.9586250

[44] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11976–11986.

[45] Linyan Mei, Koen Goetschalckx, Arne Symons, and Marian Verhelst. 2022. DeFiNES: Enabling Fast Exploration of the Depth-first Scheduling Space for DNN Accelerators through Analytical Modeling. https://arxiv.org/abs/2212.05344

[46] Francisco Muñoz-Martínez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna. 2020. STONNE: A Detailed Architectural Simulator for Flexible Neural Network Accelerators. https://doi.org/10.48550/ARXIV.2006.07137

[47] Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun. 2019. HyperMapper: a Practical Design Space Exploration Framework. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 425–426. https://doi.org/10.1109/MASCOTS.2019.00053

[48] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman P. Jouppi, and David Patterson. 2020. Google's Training Chips Revealed: TPUv2 and TPUv3. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. 1–70. https://doi.org/10.1109/HCS49909.2020.9220735

[49] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315.

[50] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 389–402.

[51] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. *SIGPLAN Not.* 48, 6 (jun 2013), 519–530. https://doi.org/10.1145/2499370.2462176

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

[53] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 58–68.

[54] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 14–27.

[55] Man Shi, Pouya Houshmand, Linyan Mei, and Marian Verhelst. 2021. Hardware-Efficient Residual Neural Network Execution in Line-Buffer Depth-First Processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 4 (2021), 690–700. https://doi.org/10.1109/JETCAS.2021.3120103

[56] Zhan Shi, Chirag Sakhuja, Milad Hashemi, Kevin Swersky, and Calvin Lin. 2020. Learned Hardware/Software Co-Design of Neural Accelerators. https://doi.org/10.48550/ARXIV.2010.02075

[57] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* 25 (2012).

[58] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[59] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*. PMLR, 10096–10106.

[60] Tianqi Tang, Sheng Li, Lifeng Nai, Norm Jouppi, and Yuan Xie. 2021. NeuroMeter: An Integrated Power, Area, and Timing Modeling Framework for Machine Learning Accelerators Industry Track Paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 841–853. https://doi.org/10.1109/HPCA51647.2021.00075

[61] Jack Turner, Elliot J. Crowley, and Michael F. P. O'Boyle. 2021. Neural Architecture Search as Program Transformation Exploration *(ASPLOS '21)*. Association for Computing Machinery, 915–927. https://doi.org/10.1145/3445814.3446753

[62] Sam Likun Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2019. SMAUG: End-to-End Full-Stack Simulation Infrastructure for Deep Learning Workloads. https://doi.org/10.48550/ARXIV.1912.04481

[63] Qingcheng Xiao, Yun Liang, Liqiang Lu, Shengen Yan, and Yu-Wing Tai. 2017. Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062244

[64] Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. 2021. HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1055–1068. https://doi.org/10.1109/ISCA52012.2021.00086

[65] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 369–383.

[66] Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2022. *A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators*. Association for Computing Machinery, 27–42.

[67] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. 2020. Ansor: Generating High-Performance Tensor Programs for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 863–879. https://www.usenix.org/conference/osdi20/presentation/zheng

[68] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. 2020. *FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System*. Association for Computing Machinery, New York, NY, USA, 859–873.

[69] Yanqi Zhou, Xuanyi Dong, Tianjian Meng, Mingxing Tan, Berkin Akin, Daiyi Peng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2022. Towards the Co-design of Neural Networks and Accelerators. In *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu (Eds.), Vol. 4. 141–152. https://proceedings.mlsys.org/paper/2022/file/31fefc0e570cb3860f2a6d4b38c6490d-Paper.pdf

[70] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.