UNICO: EFFICIENT UNIFIED HARDWARE-SOFTWARE CO-OPTIMIZATION FOR DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Specialized hardware has become an indispensable component to deep neural network acceleration. To keep up with the rapid evolution of neural networks, recently, holistic and automated solutions for jointly optimizing both hardware architectures and software mapping have been studied. In this paper, we propose UNICO, a Unified Co-Optimization framework for hardware-software co-design, aimed at addressing the efficiency issues of vast design space exploration and the issue of overfitting to specific input neural network workloads that are facing current approaches. UNICO employs multi-objective Bayesian optimization to sample hardware, and performs parallel and adaptive software mapping search for hardware samples with a customized successive halving algorithm. To reduce overfitting, UNICO incorporates quantitative robustness measures to guide the proposed search and evaluation procedure. Experiments performed for both open-source spatial accelerators and a real-world commercial environment show that UNICO significantly improves over its counterparts, finding not only superior but also more robust hardware configurations, yet at drastically lower search cost.

1 INTRODUCTION

Deep neural networks (DNNs) (LeCun et al., 2015) are pervasive nowadays, finding diverse applications in, e.g., computer vision (He et al., 2016), natural language processing (Dosovitskiy et al., 2021), and autonomous driving (Chen et al., 2015). 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 (Chen et al., 2016b), i.e. *neural network accelerators*, have been designed to speed up DNN execution. These accelerators (Chen et al., 2016b; Norrie et al., 2020; Liao et al., 2021) 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 only to execute a fixed sized GEMM tensor computation; given a DNN model, it is the responsibility of *software mapping* 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 fast execution promised by the hardware. To ease this difficulty, deep learning compiler frameworks (Li et al., 2020; Chen et al., 2018a) have been proposed, aiming at automatically synthesizing efficient mappings for different neural network models and AI accelerators.

While AI hardware and software stacks are conventionally updated separately, the solutions found may be suboptimal, resulting into compounded end-to-end performance loss when jointly deployed. Recently, holistic approaches (Zhang et al., 2022; Xiao et al., 2021; Kao et al., 2022) 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 (Tan & Le, 2019) by joint optimization requires an exploration of a large search space of $\mathcal{O}(10^{2300})$ (Zhang et al., 2022). To counter this issue, different solutions have been proposed to reduce the size of the search space, mainly focusing on 1) design space pruning, e.g., HASCO (Xiao et al., 2021), or 2) design space approximation, e.g., FAST (Zhang et al., 2022). Despite these efforts, the hardware and software design choices are still explored in isolation from an algorithmic perspective. Moreover, new models, neural network architectures are constantly emerging, which may void the hardware/software co-design optimized for specific applications/workloads.

In this paper, we propose a Unified Co-Optimization (UNICO) framework for AI accelerator co-design, which can dramatically speed up hardware/software co-optimization for DNN workloads as compared to the state-of-the-art methods. This is achieved by solving bi-level exploration in a symbiotic way such that we focus on performing software exploration only for promising hardware candidates while discarding unfavourable ones early. In the meantime, UNICO also finds robust accelerator hardware that generalizes better to new applications unseen in the co-optimization. We show that by taking additional quantitative measures in software exploration, UNICO can lessen the effect of "overfitting" hardware to input workloads as in prior approaches. Specificially, our contributions can be summarized as follows:

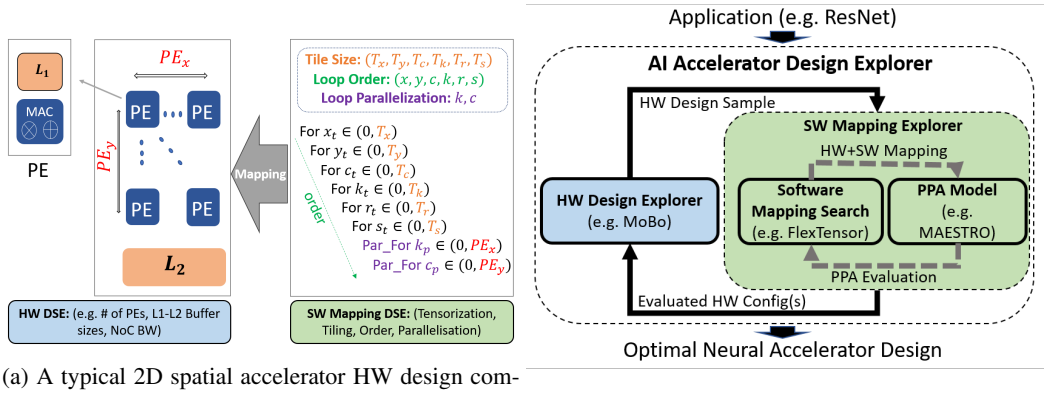
- We propose a batched hardware sampling strategy, to enable parallel hardware evaluation guided by multi-objective Bayesian optimization (MOBO) with a surrogate model that is refined with high-fidelity data samples selected by an adaptive data-driven approach;
- We propose concurrent software mapping exploration with successive halving for sampled hardware configurations, using a customized and effective candidate promoting criterion;
- We further propose a method to enhance the generalization of hardware to unseen workloads, by introducing an additional quantitative robustness measure into co-optimization.

We conduct extensive experiments on spatial accelerator co-design for a wide range of DNN workloads under the edge and cloud devices' power constraints based on open-source accelerator micro-architectural model, MAESTRO (Kwon et al., 2020). We also perform hardware-software co-search on a cycle-accurate simulator for Ascend-like NPU Architecture (Liao et al., 2021), where each evaluation of hardware-software 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 state-of-the-art methods with 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 dimension) DNNs unseen in optimization.

2 Background on Hardware/Software Co-Design

A number of studies have been conducted for designing customized hardware accelerators for providing real-time processing of deep neural networks (Jia et al., 2021; Prabhakar et al., 2017; Jouppi et al., 2017). 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 accesses, thus preserves high energy-efficiency. Figure 1a illustrates a general design template of a typical 2D spatial accelerator where the key design choices are number of processing element (PE) in X and Y axis (PE_x, PE_y), private scratch pad 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 into large variations in efficiency (Huang et al., 2021). 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 difficulty to maintain as it demands timely update whenever there is a change in the HW configuration.



(a) A typical 2D spatial accelerator HW design components (e.g. (PE_x, PE_y), L_1 and L_2 buffer sizes)

(b) Schematic illustration of HW/SW co-search.

Figure 1: Spatial Accelerator and HW/SW Co-Design.

To ease this difficulty, there have been auto-scheduling frameworks (Chen et al., 2018a; Ragan-Kelley et al., 2013) 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, tiling*. As demonstrated in Figure 1a, 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 unite (e.g. inner-most loop) can be mapped directly to certain HW resources *spatially* or *temporally* (Zheng et al., 2020b; Kwon et al., 2020). Apparently, which is the best SW mapping depends on what HW topology and parameters are selected.

The HW-SW co-design paradigm is a bi-level optimization, since 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 shapes a constraint for SW mapping parameters search space. This sequential dependency naturally implies a bi-level optimization scheme (Xiao et al., 2021; Zhang et al., 2022; Kao & Krishna, 2020). Figure 1b shows the flow diagram of a bi-level HW-SW co-optimization. In the outer-level, a HW design configuration is sampled and is passed to the inner-level for SW mapping exploration. When a specific HW config and its corresponding SW mapping is determined, a power-performance-area (PPA) estimator is needed to evaluate the quality of given HW-SW candidate. In practice, HW-SW co-design still faces the following challenges:

Large co-optimization space. The combined HW/SW space for optimization can be huge. Suppose HW design space is of $\mathcal{O}_{HW}(.)$, and SW space is of $\mathcal{O}_{SW}(.)$. Clearly, the joint space would be $\approx \mathcal{O}_{HW}(.) \times \mathcal{O}_{SW}(.)$. For a given convolution workload expressed as a 7D loop, the corresponding unconstrained SW mapping space is of $\mathcal{O}_{SW}(2^{60})$. For the spatial accelerator template as in Figure 1a, there are $\mathcal{O}_{HW}(2^{22})$ 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 (Zhang et al., 2022), it is $\mathcal{O}_{HW}(2^{44})$.

Generalizing to unseen DNN workloads. The other challenge that has not been discussed 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 designs that achieve the best PPA for that time being may not handle new DNN models at the time these HW are released to the market. This is particularly pressing for automatic HW design, since, in essence, the HW configuration being explored are invariably aimed to best fit the input DNNs. It is important to consider generality in HW-SW co-optimization from co-search aspect, ensuring that the PPA optimization is not over-fitting to a narrow set of DNNs.

3 UNICO: A UNIFIED CO-OPTIMIZATION FRAMEWORK

In this section, we present the design of UNICO to address the aforementioned challenges. Fig. 2 illustrates the overall workflow of UNICO. How these components are used in UNICO is described

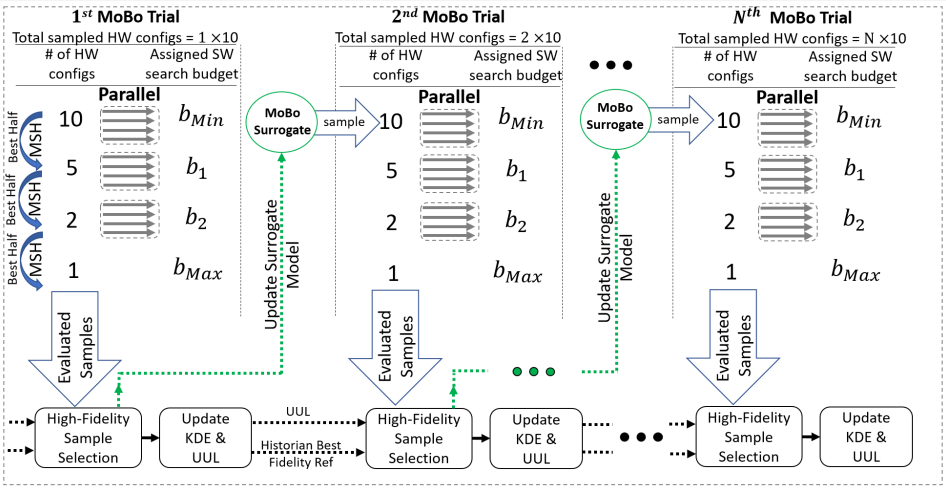


Figure 2: Implementation of UNICO with multi-objective Bayesian optimization and successive halving.

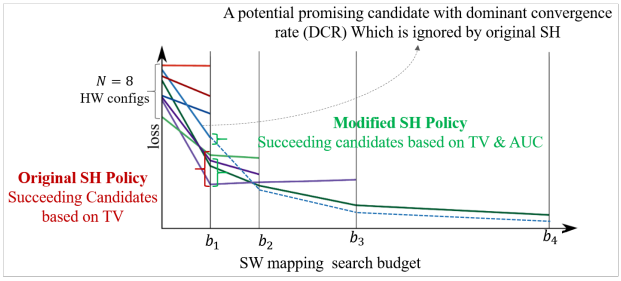
Algorithm 1: UNICO

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

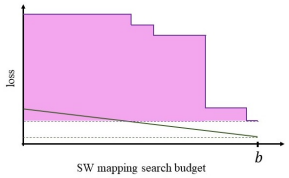
1 Randomly initialize MOBO's `Surrogate` model
2 $b = b_{\max}\eta^{-\lfloor \log_\eta b_{\max} \rfloor}$
3 **for** $i \in \{1, \ldots, \text{MaxIter}\}$ **do**
4 $H_0 \leftarrow$ Sample a batch of N HW configurations using `Surrogate` model
5 **for** $j \in \{1, ..., \lceil \log_2 N \rceil\}$ **do**
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 D by assessing $s(h, b)$, $\forall h \in H_0, \forall b \in [1, b_{\max}]$
11 Update `Surrogate` model using D
12 Update HW Pareto Front $\mathcal{X}$
13 **return** HW Pareto Front $\mathcal{X}$

in Algorithm 1. UNICO is an iterative algorithm such that in each iteration, it samples a batch of N hardware candidates. As shown in Fig. 2, UNICO adopts a bi-level HW-SW co-search strategy such that the MOBO guides the HW design space exploration (DSE). Following conventional MOBO (Nardi et al., 2019), for the co-search problem, the surrogate model inputs are HW design configurations and its outputs are co-search objectives that need to be minimized. In our implementation, we use Gaussian process (GP) as the surrogate model for MOBO. First the surrogate model is initialized randomly in line 1. Then acquisition function (e.g. MesMo) will sample a batch of N HW configs to likely minimize all objectives in line 4. Then at the end of each MOBO iteration, the surrogate model is then refined with data points collected from that iteration. In the following subsections we describe how batch of N HW candidates are evaluated and how high-fidelity configurations are selected to update the surrogate model.

To facilitate exposition, we introduce the following notations. First, HW-SW co-search can be viewed as a hyperparameter optimization task such that the choice of HW are the hyper-parameters and SW mapping exploration is deemed as a training/evaluation task. The input is a (DNN) workload w for conducting HW/SW optimization. Let $h \in H$ be a hardware sample, and



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



(b) Area under the curve (AUC) measures the area that is trapped between the curve and the horizontal line corresponding to the end loss value of the curve (i.e. dashed line).

Figure 3: A modified successive halving (MSH) that uses both AUC and terminal value to select candidates.

$g_b^{w,h} : H \rightarrow S$ represents the SW mapping search function with parameter search budget b, where each $s(w,h,b) \in S$ is the best mapping generated with budget b for workload w and hardware h. The SW mapping search is driven by minimizing some cost function, 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 is **monotonic**. In other words, for a set of software mappings with different budgets $\{s(w,h,b_1), s(w,h,b_2), \ldots, s(w,h,b_n)\}$, then $s(w,h,b_j)$ is *monotonically non-increasing* with respect to $j \in \{1,2,\ldots,n\}$. Since the input workload w is fixed throughout the optimization, to ease presentation, in the remaining text we drop w when the context is clear.

3.1 SW Mapping Search with Modified Successive Halving (MSH)

After a batch of N hardware configurations are sampled, the next step is 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) (Jamieson & Talwalkar, 2016) for this aim. However, in the original 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 software mapping search, we observe that the HW configurations with relatively steep convergence rate are also likely to be promising. In other words, if those of steep convergence candidates were given a second chance to be evaluated with higher budget in next round (i.e. selected as succeeding candidate), they might outperform those with best TVs. Fig. 3 illustrates the difference between original SH and our modified successive halving (MSH) for software mapping search. Specifically. we quantify the convergence rate of each h by measuring the area under the curve (AUC) of its mapping history (see Fig. 3b) — hardware h with higher AUC tends to converge faster, resulting into better final outcome than those with relatively smaller AUC. After measuring both by TV and AUC, we select top k succeeding HW $H^{\texttt{top}\ k}$ as follows:

$$H^{\texttt{top}\ (k)} = H_{\texttt{TV}}^{\texttt{top}\ (k-p)} \cup H_{\texttt{AUC}}^{\texttt{top}\ (p)} \quad \text{s.t. } H_{\texttt{TV}}^{\texttt{top}\ (k-p)} \cap H_{\texttt{AUC}}^{\texttt{top}\ (p)} = \emptyset \tag{1}$$

where $H_{\texttt{TV}}^{\texttt{top}\ (k-p)}$ is top $(k-p)$ HW configurations according to their TV sorted in ascending order and $H_{\texttt{AUC}}^{\texttt{top}\ (p)}$ is top (p) HW configs according to their AUC sorted in descending order. Essentially, Eq. (1) represents a generalization of the original SH, and would degenerate to the original SH if $k = \lfloor 0.5N \rfloor$ 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.

3.2 Hardware Robustness and High-Fidelity Surrogate Update

After performing parallel SW mapping search for the batch of N hardware with successive halving, for 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 HW config h. Then, there are two major questions: 1) how should we measure the quality of each HW-SW pair $(h, s(h,b))$, given that $s(h,b)$ is the best mapping for h with budget b; 2) how should we select a set of data samples to retrain MOBO's surrogate model. For 1), the straightforward approach is to call the PPA estimator function Y for each hardware h and its best mapping

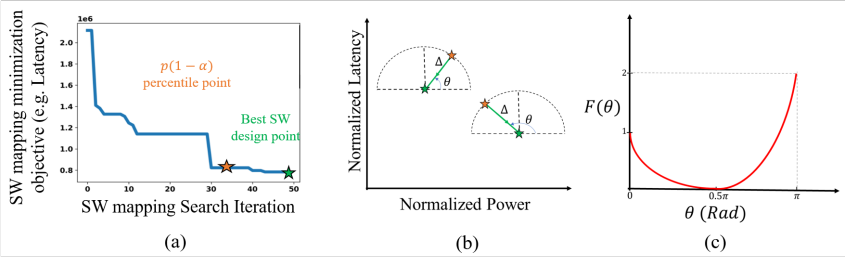


Figure 4: (a) SW Mapping objective convergence during search. (b) Pictorial illustration of HW config robustness metric calculation. (c) Proposed analytical function $F(\theta)$ to quantify the power variation behaviour with respect to latency decrease in $R = \Delta(1 + F(\theta))$.

$s(h, s(h, b_{\max}))$. That is, $Y(h, s(h, b_{\max}))$ provides a three-dimensional measurement of (`power`, `performance`, `area`). 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 mapping search. For 2), a popular approach is to collapse the multi-objectives into a single objective as in (Knowles, 2006), then select top candidates based on that single objective. The issue is, however, that for $(h, s(h, b))$, the `power` and `performance` measurements are often correlated; therefore, it is better to consider such correlation when they are combined together.

In UNICO, we propose yet another `robustness` metric R for both 1) and 2), effectively realizing a high-fidelity measurement for assessing hardware configurations. Specifically, we say a hardware h is deemed robust when both `latency` (i.e. `performance`) and `power` have negligible variations with SW mappings at a range of different budgets. Fig. 4(a) show SW mapping minimization loss for a given HW config during SW mapping search. We consider the variation of promising SW mapping choices with latency values fall in $(1 - \alpha)$ right-tail percentile of the distribution. To tackle the correlation between `latency` and `power`, we propose a geometric approach to quantify robustness as $R = \Delta(1 + F(\theta))$ where Δ is the 2-norm distance of the two selected candidates as in Fig. 4(a). θ is the linear approaching angle of the α^{th}-percentile mapping choice to the best found SW mapping. By design, $\Delta = 0$ implies ideal robustness and as shown in Fig. 4(b); $0 \leq \theta \leq \pi/2$ means that both `power` and `latency` decrease from α^{th} percentile candidate to the best one, while $\pi/2 \leq \theta \leq \pi$ means power increases which is less favourable. To incorporate this intuitive design, an analytical function $F(\theta) = (6/\pi^2)\theta^2 - (5/\pi)\theta + 1$, as shown in Fig. 4(c), is derived to incorporate the involvement of $0 \leq F(\theta) \leq 2$ in robustness metric calculation. See Appendix A.1 for detailed explanation of R.

After introducing robustness, a single scalar $V^{\texttt{Best}}_{\texttt{ParEGO}}$ of high-fidelity metric $V^{W}_{\texttt{ParEGO}}$ is determined as follows:

$$V_{\texttt{ParEGO}}(\hat{Y}, W) = \max_{j \in 1,2,3,4} (w_j y_j) + \rho(\hat{Y}^T W) \quad s.t. \quad \sum_j w_j = 1 \tag{2}$$

where W is importance weight vector for each optimization objective y_j. $\hat{Y}(h, s(h, b))$ gives a four dimensional measurement of (`performance`, `power`, `area`, `robustness`), and y_j is the measurement at dimension j. ρ (Default=0.2) adds a weighted sum of objectives to ensure partial participation of all objectives in fidelity metric calculation. Then, upper update limit (UUL) is defined as $\alpha(\text{default} = 0.95)$ percentile value corresponding to a kernel density estimator (KDE) that is fitted to a historian list of distances $d = ||V_{\texttt{ParEGO}} - V^{\texttt{Best}}_{\texttt{ParEGO}}||_2$ obtained from high-fidelity HW configurations. Finally, for every evaluated HW at the end of one MOBO iteration, only those with $d \leq UUL$ are deemed as high-fidelity and used to update MOBO surrogate model (i.e. they are used to compose distribution D as in Algorithm 1 Line 11). As in Fig. 2, new high-fidelity HW configs are used to update the KDE model and recalibrate the UUL for the next trial. By intuition, the empirical KDE learns to select higher fidelity HW configurations when MOBO surrogate model becomes more accurate as the advancing of more iteration trials. For detailed descriptions of proposed empirical MOBO surrogate update mechanism, see Appendix A.1.

4 Experiments

To show the effectiveness of our approach, we deploy UNICO to both open source accelerator (Xiao et al., 2021) and a commercial architecture (Liao et al., 2021). Specifically, we compare UNICO's

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)	*Min. Dist.*	*Cost(h)*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*	*Cost(h)*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*	*Cost(h)*
Bert	0.0032, 99.7, 1.1	0.043	35.5	0.00064, 317.9, 6.5	0.16	35.5	0.018, 18.0, 0.3	**0.0007**	**7.48**
MobileNet	3.33, 160.5, 1.9	0.07	155.5	40.77, 45.5, 1.1	0.025	85.75	36.31, 38.9, 0.2	**0.021**	**31.4**
ResNet	33.28, 244.3, 1.7	0.12	105.5	81.13, 63.4, 0.58	0.05	76.45	40.91, 57.1, 1.1	**0.029**	**21.43**
SRGAN	281.71, 103.9, 1.6	0.15	145.5	278.29, 317.9, 6.5	0.21	44	109.45, 155.2, 3.7	**0.09**	**29.4**
UNet	220.06, 303.3, 4.6	0.19	100.5	88.03, 277.3, 4.2	0.14	52.17	85.05, 148.6, 2.8	**0.08**	**20.43**
VIT	171.28, 317.9, 6.5	0.18	35.5	206.85, 239.7, 5.3	0.15	35.5	199.37, 176.0, 3.7	**0.13**	**7.48**
Xception	35.80, 244.3, 1.7	0.12	130.5	11.79, 317.9, 6.5	0.16	80.67	22.33, 68.7, 1.3	**0.03**	**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)	*Min. Dist.*	*Cost(h)*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*	*Cost(h)*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*	*Cost(h)*
Bert	0.00037, 142.4, 2.4	0.02	35.5	0.00064, 317.9, 6.5	0.04	35.5	0.0062, 29.0, 0.4	**0.002**	**7.48**
MobileNet	4.69, 57.1, 1.2	0.006	155.5	5.59, 185.4, 3.8	0.024	107.45	6.35, 43.1, 0.7	**0.0048**	**39.63**
ResNet	47.57, 136.8, 3.1	0.028	105.5	8.96, 317.9, 6.5	0.043	91.5	15.89, 114.6, 2.7	**0.016**	**28.41**
SRGAN	106.6, 263.1, 4.9	0.061	145.5	84.04, 543.2, 10.4	0.084	45.45	51.02, 310.3, 4.7	**0.048**	**29.4**
UNet	95.3, 173.2, 2.4	0.05	100.5	118.59, 293.1, 3.9	0.068	52.5	50.64, 215.7, 4.8	**0.037**	**24.42**
VIT	72.65, 591.8, 8.4	0.088	35.5	171.28, 317.9, 6.5	0.091	35.5	155.61, 287.0, 2.9	**0.082**	**13.29**
Xception	14.05, 132.6, 1.2	0.018	130.5	11.97, 317.9, 6.5	0.043	88.03	16.48, 82.9, 1.3	**0.012**	**26.41**

performance against the state-of-the-art co-search solution from the following aspects 1) efficiency improvement due to the use of batch sampling and parallel SW mapping search with successive halving, (2) generalizability capacity to unseen applications because of the newly introduce robustness metric, and (3) overall performance enhancement for industrial-class HW design upon an Ascend-like (Liao et al., 2021) architecture.

For open source accelerator experiments, GEMMCore intrinsic is selected such that the hardware search space includes PE array shape, $L1$ and $L2$ buffer sizes, NoC bandwidth and dataflow style; see (Xiao et al., 2021) for details about experiment search space. FlexTensor (Zheng et al., 2020b) is used to perform SW mapping search and MAESTRO (Kwon et al., 2020) is used to estimate PPAs. For experiment on industrial Ascend-like architecture, an expensive cycle accurate model (similar to (Tang et al., 2021; Samajdar et al., 2020; Xi et al., 2019; Muñoz-Martínez et al., 2020)) is used as the PPA estimator.

4.1 Performance of UNICO on Open Source Accelerator

We compare the performance of UNICO with the state-of-the-art open source co-design framework HASCO (Xiao et al., 2021) and NSGAII(Deb et al., 2002) on individual networks (BERT (Devlin et al., 2019), MobileNet (Howard et al., 2017), ResNet (He et al., 2016), SRGAN (Ledig et al., 2017), UNET (Ronneberger et al., 2015), VIT (Dosovitskiy et al., 2021), Xception (Chollet, 2017)) under edge (power $\leq 2W$) and cloud (power $\leq 20W$) constraints. To demonstrate the end-to-end performance of each co-search approach, the best HW config is selected according to the achieved *min-Euclidean-distance* on PPA Pareto-Front set. As can be seen in Tables 1 and 2, on average among all DNN workloads, UNICO outperforms both HASCO and NSGAII by $2.29\times$ and $2.35\times$ for edge device, and by $1.34\times$ and $1.94\times$ for cloud device, respectively. Furthermore, due to UNICO's concurrent and adaptive mapping search using modified successive halving, UNICO found these higher quality HW configs with approximately $4.5\times$ and $2.7\times$ less search cost in comparison to HASCO and NSGAII, respectively. For details regarding the selected HW configs of listed methods and models, please refer to Tables 4 and 5 in Appendix.

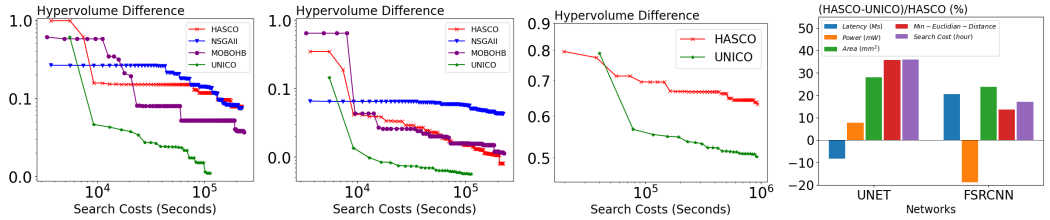
In addition to HASCO and NSGAII, we also implement a multi-objective version of BOHB (Falkner et al., 2018b). Figures 5a and 5b show the hypervolume difference across the tested networks on edge and cloud devices, respectively. Clearly, UNICO shows faster search convergence due to its advantageous batch HW 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 adaptive SW mapping search. For additional ablation studies on different UNICO's components, please refer to Appendix B.1.

4.2 Generalization Ability to Unseen Neural Networks

To demonstrate UNICO's better generalization ability for unseen applications, we again compare UNICO with HASCO (Xiao et al., 2021) by conducting the co-optimation on a set of "training"

Table 3: Comparisons of HASCO, HASCO-Robustness and UNICO on Validation Networks.

Networks	HASCO		HASCO with Robustness		UNICO	
	L(Ms),P(mW),A(mm^2)	*Min. Dist.*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*	*L(Ms),P(mW),A(mm^2)*	*Min. Dist.*
UNET	379.26, 308.6, 5.2	0.238	46.72, 200.9, 3.1	0.0923	54.18, 194.7, 4.3	**0.0899**
VIT	159.43, 311.6, 5.3	0.170	132.03, 201.2, 3.1	**0.110**	158.35, 196.2, 4.3	0.116
Xception	12.78, 307.8, 5.2	0.150	3.86, 205.4, 3.1	0.0920	2.72, 194.0, 4.3	**0.0856**
MobileNetV3Large	1.16, 308.6, 5.2	0.150	0.87, 203.0, 3.1	0.0906	0.80, 191.5, 4.3	**0.0841**
MobileNetV3Small	0.49, 308.4, 5.2	0.150	0.40, 202.9, 3.1	0.0905	0.36, 190.8, 4.3	**0.0837**
ConvNeXtBase	67.59, 308.6, 5.2	0.154	23.70, 201.5, 3.1	0.0905	11.96, 190.3, 4.3	**0.0837**
EfficientNetV2	4.06, 308.6, 5.2	0.150	1.10, 203.7, 3.1	0.0910	0.76, 191.9, 4.3	**0.0844**
NASNetMobile	1.76, 308.2, 5.2	0.150	1.50, 203.9, 3.1	0.0911	1.22, 193.1, 4.3	**0.0850**



(a) Edge Device (b) Cloud Device (c) Ascend Architecture. (d) Ascend Architecture.

Figure 5: Comparisons of UNICO and listed methods in terms of Hypervolume Difference.

workloads, and then directly apply the best found HW configurations to new and unseen applications. See Appendix B.3 for more information about how they are selected. To solely show the usefulness of the robustness metric, we implement a "HASCO with Robustness" version by directly injecting UNICO's partial robustness metric $R = \Delta$ as forth HASCO's MOBO optimization objective; thus the only difference between HASCO and "HASCO with Robustness" is that the latter contains our partial robustness metric.

Specifically, we perform the co-search on a set of training networks consisting of four networks (i.e. MobileNetV2, ResNet, SRGAN and VGG) to obtain the best HW config w.r.t the *min-Euclidean-distance*. Then, selected HW config for each approach is directly applied to a validation set consisting of seven new networks: UNET, VIT, Xception, MobileNetV3 (Howard et al., 2019), NASNet-Mobile (Zoph et al., 2018), EfficientNetV2 (Tan & Le, 2021), ConvNeXt (Liu et al., 2022). The best achieved PPAs from software mapping are reported in Table 3. As we can see, on average for all validation networks, UNICO improves the *min-Euclidean-distance* of HASCO and "HASCO with Robustness" by 44% and 7%, respectively. Considering that, for UNICO, the robustness metric R is not only an additional MOBO optimization objective but also being used in selecting high-fidelity HW configurations for MOBO surrogate model update and learning — this feature further enables UNCIO to harness the full strength of the robustness measurement and improves its generalization ability on unseen DNN workloads.

4.3 Performance of UNICO on Ascend Architectures

We compare the performance of our proposed UNICO with HASCO (Xiao et al., 2021) on Ascend-like architecture (Liao et al., 2021) on individual networks UNET and FSRCNN (Dong et al., 2016). For all experiments, PPA estimation is done by cycle accurate model (CAmodel) with benchmarked simulation accuracy of $8 \pm 3\%$. The CAmodel takes the workload App according to a compiled SW mapping choice along with a particular HW configuration and return profiled PPAs. In comparison to MAESTRO that takes fraction of seconds to output PPAs, Ascend like CAmodel wall-clock time is highly expensive due to it topological complexity. Hence, the effect of co-search wall-clock time efficiency and its fast convergence is even more crucial on total experiment time.

Relative improvement results of UNICO over HASCO is illustrated in Figure 5d. On average among all applications, UNICO achieved a HW-SW design point with 17.2% less *min-Euclidean-distance*, and the co-optimization using UNICO consumed 18.4% less search cost. Furthermore, comparing UNICO and HASCO convergence rate by hypervolume difference in Figure 5c, we can see that UNICO's high-fidelity MOBO surrogate update mechanism and its concurrent exploration strategy has led to the discovery of better HW configurations faster. In addition, we compare the PPA *min-Euclidean-distance* of best HW configurations for HASCO or UNICO and the one used in Ascend

DaVinci Lite as the default HW configuration (Liao et al., 2021). For both applications, UNICO outperforms HASCO and Ascend DaVinci Lite by 7.2% and 20.8% on average, respectively.

5 Related Work

Spatial Accelerators. Many spatial accelerators have been developed. Some accelerators follow a rigid architecture (e.g. Eyeriss (Chen et al., 2016a)) that supports only a fixed type of computation pattern. Some are more flexible and supports a large range of tensor operations (e.g. Eyeriss v2 (Chen et al., 2019), MAERI (Kwon et al., 2018)). Some aggregates multiple chiplet as one accelerator hardware (e.g. SIMBA (Shao et al., 2019)). The commercial accelerators typically have additional and more complex memory and processing element (PE) hierarchy. For example, TPU (Norrie et al., 2020) allocates dedicated buffers to weights and output with sophisticated synchronization. Huawei's *DaVinci* core (Liao et al., 2021) 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 (Parashar et al., 2019; Yang et al., 2020; Genç et al., 2019; Kao & Krishna, 2020; Zheng et al., 2020b; Chatarasi et al., 2021; Dave et al., 2019) typically use an analytical model to estimate and compare different schedule candidates. Together with manual pruning, the schedule exploration algorithms try to identify a best candidate judged by the static cost-model. The feedback-based search (Chen et al., 2018b; Ragan-Kelley et al., 2013; Zheng et al., 2020a; Gao et al., 2021) 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 analytical model by human experts, relying on feedback from the real hardware for cost model learning and search.

PPA Estimation. Various models have been proposed to estimate PPA from internal metrics such as data-reuse, # of FLOPS, minimum required buffers, etc. MAESTRO (Kwon et al., 2020) models spatial accelerator architecture from a data-centric perspective. TimeLoop (Parashar et al., 2019) 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 (Tang et al., 2021; Samajdar et al., 2020; Xi et al., 2019; Muñoz-Martínez et al., 2020).

Hyperparameter optimization. Both Bayesian optimization (BO) (Snoek et al., 2012) and successive halving (SH) (Jamieson & Talwalkar, 2016) have been used for hyperparameter optimization, but with different focuses. The former aims on adaptive configuration selection while the latter pays more attention on adaptive resource allocation and early stopping. HYPERBAND (Li et al., 2017) wraps over SH by essentially performing a grid search for addressing the "n versus B/n" question in SH. BOHB (Falkner et al., 2018a) combines HYPERBAND with BO, aspiring to achieve both strong anytime performance and optimal convergence. UNICO resembles BOHB in spirit but is specially designed to the application of HW/SW co-optimization.

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 which 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 cost, while generalizing well to new applications that are unseen in the co-optimization process.

REFERENCES

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

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

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. TVM: An automated End-to-End optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594, Carlsbad, CA, October 2018a. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/chen.

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

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

Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 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):292–308, 2019.

Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Diannao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM*, 59(11):105–112, 2016b.

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

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

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

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

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. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

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

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

Chao Gao, Tong Mo, Taylor Zowtuk, Tanvir Sajed, Laiyuan Gong, Hanxuan Chen, Shangling Jui, and Wei Lu. Bansor: Improving tensor program auto-scheduling with bandit based reinforcement learning. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 273–278. IEEE, 2021.

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. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *ArXiv*, abs/1911.09925, 2019.

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

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

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

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

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

Liancheng Jia, Zizhang Luo, Liqiang Lu, and Yun Liang. Tensorlib: A spatial accelerator generation framework for tensor algebra. *CoRR*, abs/2104.12339, 2021. URL `https://arxiv.org/abs/2104.12339`.

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. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, jun 2017.

Sheng-Chun Kao and Tushar Krishna. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, 2020.

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

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multi-objective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

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

Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro*, 40(3):20–29, 2020. doi: 10.1109/MM.2020.2985963.

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

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

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1): 6765–6816, jan 2017. ISSN 1532-4435.

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

Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. 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)*, pp. 789–801, 2021. doi: 10.1109/HPCA51647.2021.00071.

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

Francisco Muñoz-Martínez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna. Stonne: A detailed architectural simulator for flexible neural network accelerators, 2020. URL `https://arxiv.org/abs/2006.07137`.

Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun. Hypermapper: a practical design space exploration framework. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 425–426, 2019. doi: 10.1109/MASCOTS.2019.00053.

Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman P. Jouppi, and David Patterson. Google's training chips revealed: Tpuv2 and tpuv3. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1–70, 2020. doi: 10.1109/HCS49909.2020.9220735.

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. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 304–315, 2019.

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

Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *SIGPLAN Not.*, 48(6):519–530, jun 2013. ISSN 0362-1340. doi: 10.1145/2499370.2462176. URL https://doi.org/10.1145/2499370.2462176.

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

Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 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)*, pp. 58–68. IEEE, 2020.

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

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

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

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

Tianqi Tang, Sheng Li, Lifeng Nai, Norm Jouppi, and Yuan Xie. 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)*, pp. 841–853, 2021. doi: 10.1109/HPCA51647.2021.00075.

Sam Likun Xi, Yuan Yao, Kshitij Bhardwaj, Paul Whatmough, Gu-Yeon Wei, and David Brooks. Smaug: End-to-end full-stack simulation infrastructure for deep learning workloads, 2019. URL https://arxiv.org/abs/1912.04481.

Qingcheng Xiao, Size Zheng, Bingzhe Wu, Pengcheng Xu, Xuehai Qian, and Yun Liang. Hasco: Towards agile hardware and software co-design for tensor computation. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1055–1068, 2021. doi: 10.1109/ISCA52012.2021.00086.

Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 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*, pp. 369–383, 2020.

Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. *A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators*, pp. 27–42. Association for Computing Machinery, 2022. ISBN 9781450392051.

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. Ansor: Generating High-Performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 863–879. USENIX Association, November 2020a. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/zheng.

Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. *FlexTensor: An Automatic Schedule Exploration and Optimization Framework for Tensor Computation on Heterogeneous System*, pp. 859–873. Association for Computing Machinery, New York, NY, USA, 2020b. ISBN 9781450371025.

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

A UNICO'S IMPLEMENTATION

A.1 HW DESIGN ROBUSTNESS AND EMPIRICAL MOBO SURROGATE UPDATE PROCEDURE

In this subsection, we provide more details on proposed robustness metric R. Then, we elaborate on how UNICO selects high-fidelity HW configs D to update the MoBo surrogate model.

HW Config Robustness to Promising SW Mapping Choices:

The existing co-search solutions (Zhang et al., 2022)Xiao et al. (2021) do not have a mechanism in their search methodology to avoid Pareto-front HW configs overfit to only those apps present at the time of co-search. Hence, UNICO introduces the concept of HW design robustness as a regularizer to maintain a degree of generalizability. Such HW design robustness can be enforced either from a HW system design topology perspective or from algorithmic concept during search, which the latter is the case in UNICO.

As shown in Fig. 4(b), to intuitively quantify robustness metric $R = \Delta(1 + F(\theta))$, we use Δ to indicate the Euclidean distance between 'orange' (relatively worse among promising choice) and 'green' (best among promising choice) points in Fig. Fig. 4(a), meaning if the $\Delta = 0$, then $R = 0$. However, if $\Delta > 0$, we want to further consider the coupled behavior of latency and power; this is purposed by the second term $(1 + F(\Theta))$. According to Fig. 4(b), if $\Theta = \pi/2$, that means latency was decreased by Δ and power was not changed, which implies power robustness but latency non-robustness with value of Δ. To incorporate this power-latency interactions in our robustness metric calculation, we defined $F(\theta) = (6/\pi^2)\theta^2 - (5/\pi)\theta + 1$ such that $F(\pi/2) = 0$, hence $R = \Delta(1 + 0) = \Delta$.

For the scenario when the orange point is in first circle quarter, $0 \leq F(0 \leq \theta \leq \pi/2) \leq 1$, the robustness metric range is $\Delta \leq R \leq 2\Delta$. That means if $\theta = 0$, power variation value is Δ and $F(0) = 1$, hence $R = \Delta(1 + 1) = 2\Delta$. This specifically implies our design prefer $\theta = \pi/2$ more than $\theta = 0$. On the other hand, when the 'orange' point is in the second circle quarter, $0 \leq F(\pi/2 \leq \theta \leq \pi) \leq 2$, hence the robustness metric is further penalized to be within $\Delta \leq R \leq 3\Delta$. The intuition behind this design is that from point 'orange' to 'green', power was increased which is not favorable by our design in comparison to the other case (first circle quarter) when both latency and power decreases. That is, if $\theta = \pi$, then $F(\pi) = 2$, hence $R = \Delta(1 + 2) = 3\Delta$.

Empirical MoBo Surrogate Update

According to PPA and robustness metric R, UNICO adopts a data-driven approach to fit an empirical KDE model, which can distinguish low-fidelity HW configs from high-fidelity ones and use the later to update the MoBo surrogate model. In each MoBo trial, the full SW mapping history result during concurrent successive halving implementation is collected to be used for calculating the robustness metric using $R = \Delta(1 + F(\theta))$. After obtaining the $Y = (\texttt{latency}, \texttt{Power}, \texttt{Area}, R)$ for every HW configs, we compute the ParEGO metric $V_{\texttt{ParEGO}}$ shown in Eq. 2. In the next step, we compute the euclidean distance of $V_{\texttt{ParEGO}}$ for each HW config of the batch in this MoBo trial from the historian best (i.e. minimum) $V^{\texttt{Best}}_{\texttt{ParEGO}}$. It should be noted that $V^{\texttt{Best}}_{\texttt{ParEGO}}$ keeps updated at the end of each MoBo trial according to the new HW samples fidelity values. Let $V_{\texttt{Ref List}}$ is the historian list of all high-fidelity $V_{\texttt{ParEGO}}$ values form previous MoBo trials. Also, let $\mathcal{D}$ is the updated list of euclidean distance of set $V_{\texttt{RefList}}$ from current $V^{\texttt{Best}}_{\texttt{ParEGO}}$. Then UNICO proposes to fit a kernel density estimator (KDE) to $\mathcal{D}$ at the end of each MoBo trial, and re-calibrate UUL as $\alpha(\texttt{Default} = 0.95)$ percentile value of the newly fitted KDE. Finally, at the end of each MoBo trial, the HW configs whose their $V_{\texttt{ParEGO}}$ Euclidean distances from historian $V^{\texttt{Best}}_{\texttt{ParEGO}}$ are not greater than UUL is selected as high-fidelity samples and are used to update MoBo surrogate in Algorithm 1 line 11.

B ADDITIONAL EXPERIMENTAL EVALUATION & RESULTS

B.1 ABLATION STUDY ON DIFFERENT UNICO'S COMPONENTS

In this section, we investigate implication of different variations of UNICO's components. With regards to early stopping rule, we compare original successive halving (SH) with UNICO's customized SH. Also, with regards to MoBo surrogate update component, we investigate implication of adopting three settings including (a) using only champion HW config that was assigned $b_{\texttt{max}}$

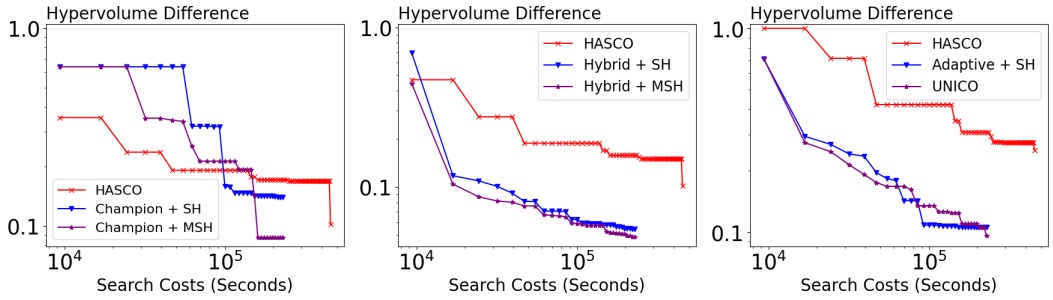
Table 4: Comparisons of HASCO, NSGAII and UNICO on the Edge Device (Power $<$ 2W).

Networks	HASCO		NSGAII		UNICO	
	PE_{xy},L1(B),L2(KB),NocBW,DF	Min. Dist.	PE_{xy},L1(B),L2(KB),NocBW,DF	Min. Dist.	PE_{xy},L1(B),L2(KB),NocBW,DF	Min. Dist.
Bert	(1,8), 80, 1, 48, WS	0.043	(1,24), 768, 2, 128, WS	0.16	(1,1), 192, 1, 64, OS	**0.0007**
MobileNet	(13,1), 256, 64, 48, WS	0.07	(1,3), 128, 96, 128, WS	0.025	(3,1), 32, 192, 16, OS	**0.021**
ResNet	(7,3), 56, 8, 16, OS	0.12	(1,5), 154, 64, 32, WS	0.05	(2,2), 72, 1024, 96, OS	**0.029**
SRGAN	(8,1), 640, 16, 64, OS	0.15	(1,24), 768, 16, 128, WS	0.21	(11,1), 72, 640, 160, OS	**0.09**
UNet	(1,24), 768, 1, 80, WS	0.19	(11,2), 84, 32, 80, OS	0.14	(11,1), 256, 64, 112, OS	**0.08**
VIT	(1,24), 768, 2, 128, WS	0.18	(1,18), 768, 16, 128, OS	0.15	(13,1), 154, 64, 128, OS	**0.13**
Xception	(7,3), 56, 8, 16, OS	0.12	(1,24), 768, 16, 128, WS	0.16	(5,1), 72, 24, 96, OS	**0.03**

mapping search budget, (b) using a hybrid filtering rule that only selects those HW configs that are Pareto-front among the current batch of N, and (c) UNICO's proposed adaptive high-fidelity approach.

According to above choices, five different scenarios are defined to be implemented for comparison with proposed UNICO and HASCO. All co-search experiments are implemented in a multi-App setting including UNET, SRGAN and BERT and VIT. The experimentation configuration is same as those in section 4.

From Fig. 6a, Modified-SH improves the performance of HASCO by approximately 7%, while SH decreases the performance of HASCO by approximately 48%. This shows that Modified-SH balances the exploitation and the exploration better than the original HASCO MOBO method (using no-SH and Champion Point update method), thus being able to find better design point using smaller search costs. Besides, from Figure 6b, the performance of Modified-SH is 43% better than HASCO and is 7% better than SH. Moreover, from Figure 6c, the performance of Modified-SH is approximately 52% better than HASCO, and is approximately 5% better than SH. In summary, by isolating choice of successive having policy, our proposed customized SH method improves the performance and efficiency of the original SH. In addition, our proposed Adaptive high-fidelity MoBo surrogate update mechanism outperforms outperforms the other update setting choices.



(a) Champion Point Update (b) Hybrid Update (c) Adaptive Update

Figure 6: Comparisons of UNICO and listed methods in terms of Hypervolume Difference

B.2 UNICO Co-search Performance

As in Section 4.1, we show the performance of HASCO, NSGAII and UNICO on individual networks under the edge and cloud devices' power constraint, respectively, in terms of PPA performance, min-Euclidean-distance and search cost. In this subsection, we further provide the corresponding HW configs for listed search methods and networks as the supplementary material in Tables 4 and 5.

B.3 Selection of Unseen DNN Workloads for UNICO Experimentation

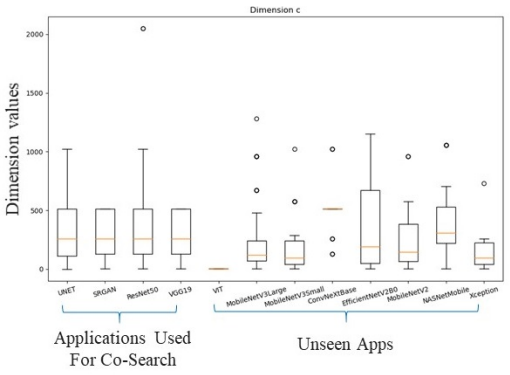
Section 4.2 shows how the robustness metric helps UNICO to achieve better generalization. In the experiments (section 4.2), four worklaods (MobileNetV2 Howard et al. (2017), ResNet He et al. (2016), SRGAN Ledig et al. (2017) and VGG19) are selected for co-design optimization. A list of other apps UNET, VIT, Xception, MobileNetV3 (Howard et al., 2019), NASNetMobile (Zoph et al., 2018), EfficientNetV2 (Tan & Le, 2021), ConvNeXt (Liu et al., 2022))3 are considered as

Table 5: Comparisons of HASCO, NSGAII and UNICO on the Cloud Device (Power $<$ 20W).

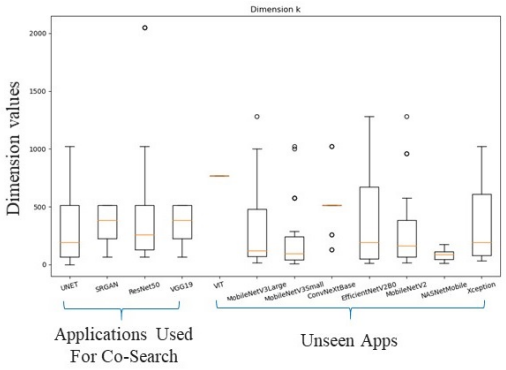
Networks	HASCO		NSGAII		UNICO	
	PE_{xy},L1(B),L2(KB),NocBW,DF	*Min. Dist.*	*PE_{xy},L1(B),L2(KB),NocBW,DF*	*Min. Dist.*	*PE_{xy},L1(B),L2(KB),NocBW,DF*	*Min. Dist.*
Bert	(1,11), 80, 128, 96, OS	0.02	(1,24), 768, 2, 128, WS	0.04	(2,1), 80, 12, 48, OS	**0.002**
MobileNet	(4,1), 42, 24, 112, OS	0.006	(1,14), 128, 16, 128, WS	0.024	(3,1), 84, 192, 80, OS	**0.0048**
ResNet	(5,2), 80, 4, 144, WS	0.028	(1,24), 768, 2, 128, WS	0.043	(2,4), 288, 512, 144, OS	**0.016**
SRGAN	(10,2), 112, 64, 112, OS	0.061	(3,14), 768, 16, 112, WS	0.084	(24,1), 192, 768, 80, OS	**0.048**
UNet	(12,1), 1024, 640, 64, OS	0.05	(12,2), 42, 32, 64, OS	0.068	(8,2), 1024, 64, 144, OS	**0.037**
VIT	(4,12), 192, 16, 64, WS	0.088	(1,24), 768, 2, 128, WS	0.091	(1,24), 154, 288, 32, OS	**0.082**
Xception	(11,1), 56, 32, 32, OS	0.018	(1,24), 768, 16, 128, WS	0.043	(2,3), 640, 96, 64, OS	**0.012**

unseen workloads. Our unseen apps are selected according to their year of release and differences in layers input dimensions. That is, co-search apps are mostly models published before 2018 and unseen apps are mostly after that. The intuition is that if we design a HW today according to the targeted end-to-end applications, the selected HW design configuration should remain superior for the years to come when new SOTA DNN models are introduced to the market.

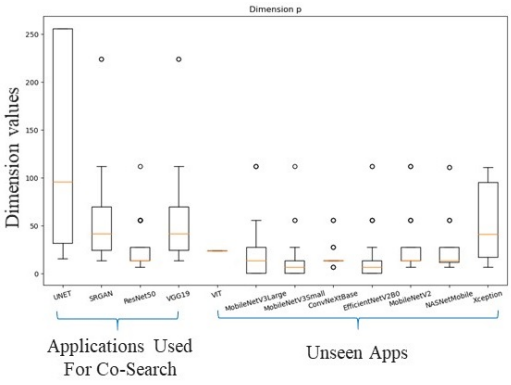
Either convolution(Conv2D, DWConv2D, etc.) operator or fully-connected operator (matrix multiplication, MatMul) can be representation as a seven nested loop with dimensions C, K, P, Q, R, S. For MatMul, some of these dimensions are 1. For all mentioned apps above, Fig. 7 compares operators' loops dimensions distributions to show our second reasoning behind selecting unseen apps. That is, the four apps utilized in co-search have average higher filter sizes P and Q. The output sizes P and Q for unseen apps are relatively larger.



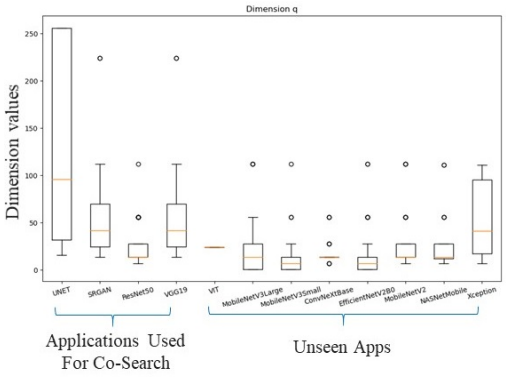
(a) Dimension C.



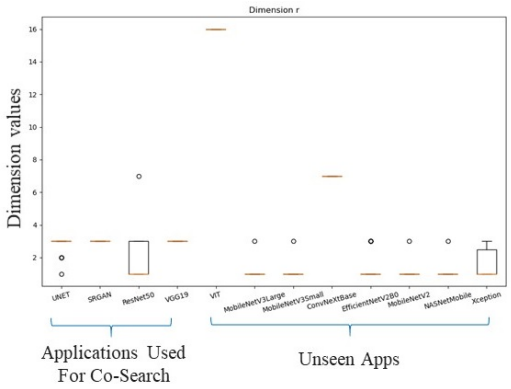
(b) Dimension K.



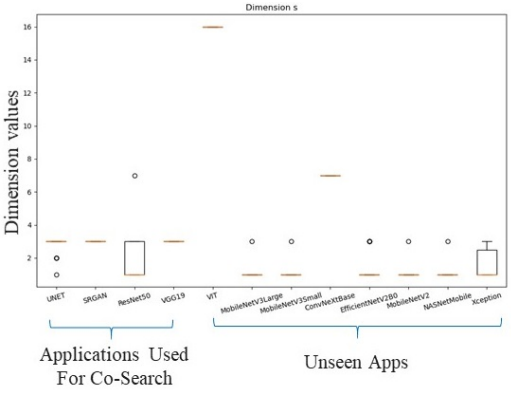
(c) Dimension P.



(d) Dimension Q.



(e) Dimension R.



(f) Dimension S.

Figure 7: Dimension statistics of seen and unseen apps w.r.t their seven nested loop representation of Conv2D and MatMul operators.