SoC-Tuner: An Importance-guided Exploration Framework for DNN-targeting SoC Design

Shixin Chen Su Zheng Chen Bai Wenqian Zhao Shuo Yin Yang Bai Bei Yu
The Chinese University of Hong Kong

***Abstract*—Designing a system-on-chip (SoC) for deep neural network (DNN) acceleration requires balancing multiple metrics such as latency, power, and area. However, most existing methods ignore the interactions among different SoC components and rely on inaccurate and error-prone evaluation tools, leading to inferior SoC design. In this paper, we present SoC-Tuner, a DNN-targeting exploration framework to find the Pareto optimal set of SoC configurations efficiently. Our framework constructs a thorough SoC design space of all components and divides the exploration into three phases. We propose an importance-based analysis to prune the design space, a sampling algorithm to select the most representative initialization points, and an information-guided multi-objective optimization method to balance multiple design metrics of SoC design. We validate our framework with the actual very-large-scale-integration (VLSI) flow on various DNN benchmarks and show that it outperforms previous methods. To the best of our knowledge, this is the first work to construct an exploration framework of SoCs for DNN acceleration.**

I. Introduction

Designing system-on-chip (SoC) for deep neural network (DNN) acceleration is getting increasingly critical and challenging. To keep up with the rapid evolution of the DNN algorithm, the demand for the optimization of DNN accelerators increases as well. Although many approaches have been proposed over the past few decades to optimize accelerator design, they may suffer from the rapidly growing scale and complexity and can not perform as effectively or efficiently on the advanced SoC design.

The first challenge is accurate performance evaluation. For example, some existing works targeting accelerators [1]–[5] ignored discussions on the interaction between the host processor and the accelerator in SoCs, ignoring the costs of communication and control. Therefore, the overall inference latency of DNN given by these tools may be inaccurate, which hinders the design of an optimal accelerator. Meanwhile, some analytical tools [6]–[8] are proposed to evaluate SoC design swiftly. However, these tools only consider very limited parameters of SoC architecture and rigidly report the calculation of single-layer, which brings huge gaps to the reporting values and the actual metrics. We tackle this problem by implementing the complete very-large-scale-integration (VLSI) flow for authentic and detailed evaluation.

The following challenge is the exploration difficulty, which is rather critical as the SoC design gets more complicated. It usually requires many rounds of improvement iterations [9] with domain expertise to get optimal design. Time costs will be enormous if we avoid error-prone simple analytical models, as mentioned above. Apart from that, The exploration process is heavily dependent on the personal experience of architects, which may bring personal bias in design optimization and result in inferior SoC design.

To bridge the bottlenecks mentioned above, we propose SoC-Tuner, an importance-guided exploration framework to find the optimal SoC design for DNN acceleration. SoC-Tuner aims to find the optimal SoC design, which balances multiple metrics including inference latency, power consumption, and chip area for various DNNs. Our contributions can be concluded as follows:

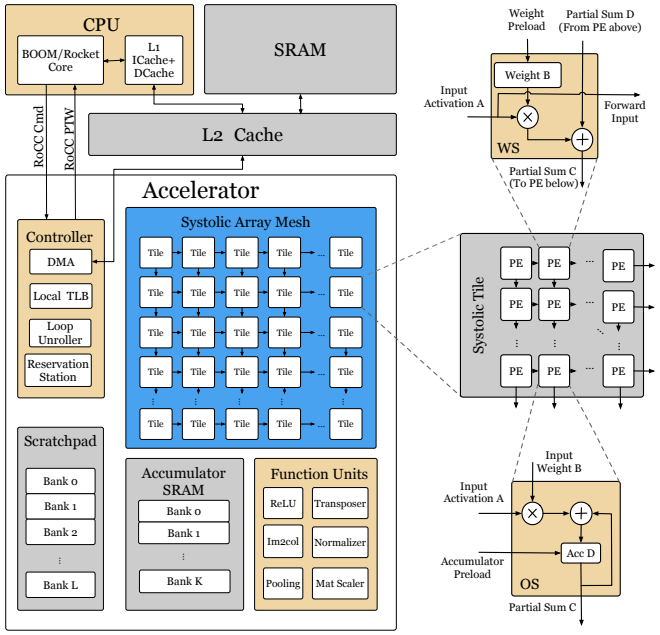


Fig. 1 Architecture of a SoC with a systolic-based accelerator.

- We thoroughly consider various SoC components that influence DNN computations and construct a huge design space to avoid insufficient evaluation of overall DNN inference.
- We employ actual very-large-scale-integration (VLSI) flow to evaluate multiple metrics, which achieves more accurate modeling of SoC than simplified analytical tools.
- We propose an importance-based analysis to prune the design space, a sampling algorithm to select the most representative initialization points, and an information-guided multi-objective optimization method to balance multiple design metrics of SoC design.
- Experimental results demonstrated the efficiency and effectiveness of our framework on various benchmarks compared to some state-of-the-art methods.

II. Preliminaries

A. SoC with DNN Accelerator

A typical DNN-targeting SoC containing an accelerator is shown in Fig. 1, where the systolic array [10] [11] is one of the most widely used architectures for DNN accelerators. In Fig. 1, the CPU allocates instructions to the accelerator using Rocket [12] co-processor command (RoCC) instructions including `Load`, `Store`, and `Execute`. The SRAM stores the DNN models to be computed by the accelerator. The CPU and accelerator share L2 cache.

To facilitate designing SoC rapidly, we employ the agile hardware development method. It aims to construct highly modeled and parameterized hardware components in Chisel [13] language, which

can be easily initiated with various architecture parameters. Chipyard [14] is integration with a variety of hardware components like CPU cores (*e.g.*, in-order Rocket [12] and out-of-order CPU core BOOM [15]) and co-processors (*e.g.*, vector-thread processor Hwacha [16] and DNN accelerator Gemmini [17]). It provides us an opportunity to easily design new SoC architectures given existing components.

Designing an optimal SoC design with given components is time-consuming and complicated. Previous work [9] explores the design space of microarchitecture of processor core with learning methods, and [18] utilizes a ranking-based approach to explore the optimal design of CPU core. However, these works solely focus on a single processor and bring challenges to more complicated systems like SoC. In SoC design, the control logic and computations are rather complicated due to the interactions between various components shown in Fig. 1. Moreover, the design space of SoC is more huge than a single CPU. Therefore, an exploration framework for characteristics of SoC design is necessary.

B. Problem Formulation

Definition 1 (SoC Architecture Design)**.** *A combination of the features listed in TABLE I is denoted as a design point $\boldsymbol{x}$ of SoC, and all design points make up the entire design space $\mathcal{X}$. An SoC architecture can be determined by a design point $\boldsymbol{x}$. We define the SoC design problem as finding an $\boldsymbol{x} \in \mathcal{X}$ to design an SoC that balances the latency, power, and area for various DNNs.*

Definition 2 (Multi-objectives Optimization of SoC)**.** *Multi-objective optimization of SoC is defined to find $\boldsymbol{x} \in \mathcal{X}$ to trade-off m objective functions $\{f_1(\boldsymbol{x}), \cdots, f_m(\boldsymbol{x})\}$. We define $\boldsymbol{y} = \mathbb{F}(\boldsymbol{x}) = (y_1, \cdots, y_m)$, where different $y_i = f_i(\boldsymbol{x})$ is obtained by various evaluation tools based on $\boldsymbol{x}$. All $\boldsymbol{y}$ formulates metrics space $\mathcal{Y} = \{\boldsymbol{y} | \boldsymbol{y} = \mathbb{F}(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}\}$.*

Definition 3 (Pareto Optimal Set of SoC)**.** *For an optimization problem, an m-dimensional objective $\boldsymbol{y} = \mathbb{F}(\boldsymbol{x})$ is said to be dominated by $\boldsymbol{y}^* = \mathbb{F}(\boldsymbol{x}^*)$ if*

$$\begin{aligned} &\forall i \in [1, m], \mathbb{F}_i(\boldsymbol{x}) \leq \mathbb{F}_i(\boldsymbol{x}^*); \\ &\exists j \in [1, m], \mathbb{F}_j(\boldsymbol{x}) < \mathbb{F}_j(\boldsymbol{x}^*), \end{aligned} \tag{1}$$

where we denote $\boldsymbol{y}^* \succcurlyeq \boldsymbol{y}$ to represent this situation. In the entire design space, a set of design points not dominated by any other points form the Pareto optimal set. In the Pareto optimal set, a design point can not be optimized without sacrificing other objectives.

In the design space exploration of SoC, the chip area, inference latency, and power consumption are a group of negatively correlated metrics, so an SoC design cannot improve one metric without sacrificing another metric. Therefore, to design an optimal SoC is to find the Pareto optimal set of SoC designs, and then choose one design point that balances multiple objectives.

Problem 1 (Design Space Exploration of SoC)**.** *Given an SoC design space $\mathcal{X}$, and the metrics space $\mathcal{Y} = \{\boldsymbol{y} | \boldsymbol{y} = \mathbb{F}(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{X}\}$, we define the design space exploration of SoC as finding a subset $\mathcal{X}^* \in \mathcal{X}$, whose corresponding metrics $\mathcal{Y}^*$ form the Pareto optimal set. Hence,*

$$\begin{aligned} \mathcal{Y}^* &= \{\boldsymbol{y}^* | \boldsymbol{y}^* \not\succcurlyeq \boldsymbol{y}, \forall \boldsymbol{y} \in \mathcal{Y}\}, \\ \mathcal{X}^* &= \{\boldsymbol{x} | \mathbb{F}(\boldsymbol{x}) \in \mathcal{Y}^*, \forall \boldsymbol{x} \in \mathcal{X}\}. \end{aligned} \tag{2}$$

III. Methodology

A. Overview of SoC-Tuner

The overall flow of our framework is displayed in Fig. 2, which contains three parts, including SoC Design Space Construction, VLSI Flow, and Exploration Flow.

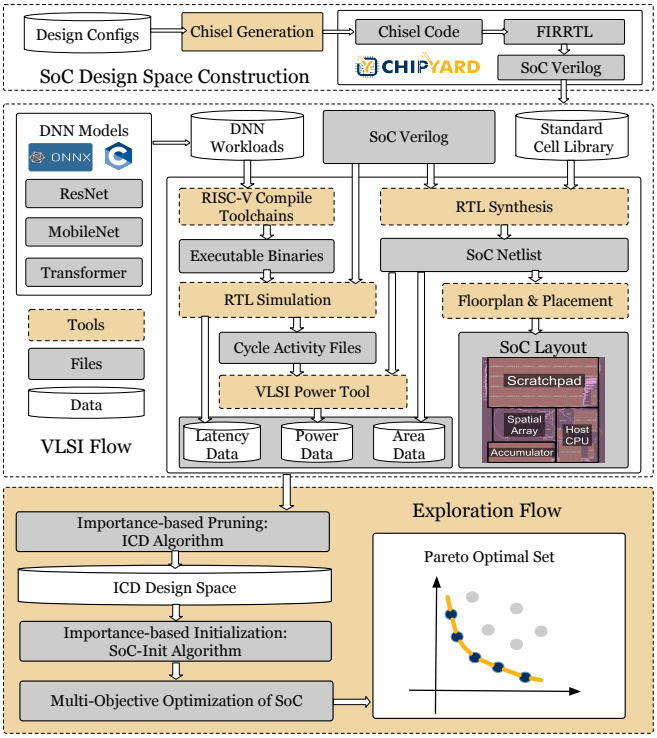


Fig. 2 The overall flow of the proposed SoC-Tuner.

In SoC Design Space Construction, we implement a Chisel generation tool to take in design points from design configurations and generate Verilog-based SoC design in Chipyard. Then we utilize the VLSI Flow that consists of several tools colored in yellow and intermediate files colored in gray. Its outputs are important metrics like latency, power, and chip area of SoC. To evaluate the performance on DNN workloads, popular DNNs like ResNet, MobileNet, Transformer, *etc.* are provided in open neural network exchange (ONNX) format or C code. In the Exploration Flow, the metrics data from the VLSI Flow are fed in to optimize the SoC design. We propose an inter-cluster distance (ICD) algorithm for design space pruning, and an SoC-Init algorithm for exploration initialization, which improve the efficiency of exploration efficiency of huge design space. Finally, we employ correlated multi-objective Bayesian optimization to find the Pareto optimal set of SoC designs that balance multiple metrics. The details of SoC-Tuner will be elaborated as follows.

B. SoC Design Space Construction

In Fig. 1, the blue part shows the detailed structure of a mesh of tiles in the systolic array. A tile is an array consisting of a grid of processing elements (PEs) that can perform parallel multiplication-accumulation (`MAC`): $C = A \times B + D$, where A, B, D represent the activation matrix, the weight matrix, and the result of the prior `MAC`, respectively. The right part of Fig. 1 illustrates the details of these two modes of the systolic array, *i.e.*, weight-stationary (WS), and output-stationary (OS). In WS mode, the weight of DNNs is pre-stored in the PEs, while in OS mode, the partial sum of computations is pre-stored in the systolic array. We can choose either mode or both of them in an SoC design depending on different DNNs.

By thoroughly considering all components that influence the metrics of SoC, we build the TABLE I that lists the design parameters of the whole SoC in Fig. 1. All combinations of features in TABLE I form a huge design space of all possible SoC designs. The SoC parameters are classified into several groups *i.e.*, CPU core & L2 cache, systolic array,

TABLE I Selected parameters from the SoC containing a systolic-based accelerator

Components	Descriptions	Candidate Values
HostCore	Various Host CPU core	c1, c2, c3
L2Bank	Entries of L2 cache banks	1, 2, 4
L2Way	Entries of L2 cache way	4, 8, 16
L2Capa	Capacity of L2 cache bank	128, 256, 512
Tilerow/col	Dimension of the tile	1, 2, 4, 8
Meshrow/col	Dimension of the mesh	8, 16, 32, 64
Dataflow	Dataflow mode of systolic array	WS, OS, BOTH
InputType	Bit width of input datatype	8, 16, 32
AccType	Bit width of accumulator datatype	8, 16, 32
OutType	Bit width of output datatype	8, 20, 32
SpBank	Banks of scratchpad memory	4, 8, 16,32
SpCapa	Entries of scratchpad bank	64, 128, 256, 512
AccBank	Banks of accumulator memory	1, 2 ,4 ,8
AccCapa	Entries of accumulator bank	64, 128, 256, 512
LdQueue	Entries of the `Lord` queue	2, 4, 8, 16
StQueue	Entries of the `Store` queue	2, 4, 8, 16
ExQueue	Entries of the `Execute` queue	2, 4, 8, 16
LdRes	Entries of the `Lord` in ROB	2, 4, 8, 16
StRes	Entries of the `Store` in ROB	2, 4, 8, 16
ExRes	Entries of the `Execute` in ROB	2, 4, 8, 16
MemReq	memory requests in-flight	16, 32, 64
DMABus	Width of DMA bus	32, 64, 128
DMABytes	Number of bytes in DMA bus	32, 64, 128
TLBSize	Size of TLB page	4, 8, 16

accelerator memory, accelerator controller, and RoCC communication shown from the top to the bottom of TABLE I. Three representative CPU cores, *i.e.*, c1 (`LargeBoom`), c2 (`LargeRocket`), and c3 (`MedRocket`) are chosen as the candidates of the host core. For the

C. Importance-based Pruning and Initialization

Importance-based SoC Design Space Pruning. Designing an SoC with high performance is complicated and time-consuming. Developers should choose the most representative design points to evaluate the SoC design and get adequate information to guide the design. Due to the time-consuming VLSI flow, only a limited number of designs will be synthesized to obtain evaluation metrics. However, randomly sampling the design parameters like [19] may ignore some domain-specific knowledge in SoC design.

In fact, there exist important features that have a significant influence on the metrics of the SoC, which means that by slightly changing the feature value, the metrics will change heavily. To model this influence, we use a vector $\boldsymbol{v}$ to denote the importance of each parameter and propose Algorithm 1 to evaluate the parameter importance via a few VLSI flow trials. In Algorithm 1, line 1 represents a few VLSI trials, and line 4 clusters the metrics space $\mathcal{Y}'$ into several groups according to candidates of various design features. Line 5 $\sim$ 8 is to get the average metric vectors of each group. Line 9 illustrates the conclusion of inter-cluster distance (ICD), where $C_2^{|M|}$ is the number of two-combination in average vectors M. Finally, after normalization, a d_x-dimension vector $\boldsymbol{v}$ is given to represent the importance of each design feature.

Considering all the features listed in TABLE I, the whole SoC design space $\mathcal{X}$ is too huge to fully explore. To avoid unnecessary exploration brought by less important feature parameters, the original design space $\mathcal{X}$ will be pruned based on $\boldsymbol{v}$ given by the ICD algorithm. Supposing $\{\boldsymbol{x}_i^1, \cdots, \boldsymbol{x}_i^j\}$ indicates the j candidates of the i^{th} feature of design point $\boldsymbol{x}$, we can use ICD vector $\boldsymbol{v}$ to prune the design space, shown in line 1 of Algorithm 2, where v_{th} represents the importance threshold. Higher v_{th} will remove more design points, and medium(.) chooses the medium value.

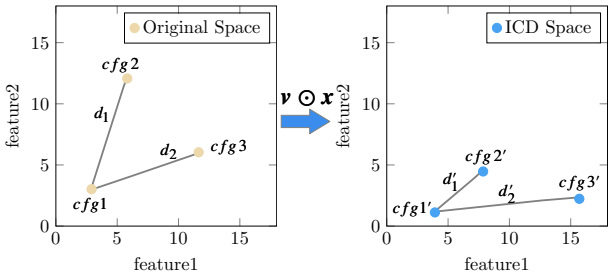


Fig. 3 A toy example with two features shows the transformation from the original space to the ICD space.

Algorithm 1 `ICD` $(\mathcal{X}, n)$

Input: $(\mathcal{X}, n)$, where $\mathcal{X}$ is the whole design space, n is the trial times of importance analysis. In a d_x-dimension design point $\boldsymbol{x} = \{x_1, x_2, ..., x_{d_x}\} \in \mathcal{X}$, each feature $x_i, i \in \{1, \cdots, d_x\}$ has t_i design candidates. And d_y is the dimension of the metrics $\boldsymbol{y} \in \mathcal{Y}$.
Output: The feature importance value factor $\boldsymbol{v}$.

1: $\mathcal{Y}' =$ **VLSIFlow**(`Sample`$(\mathcal{X}, n)$);
2: $\boldsymbol{v} = \emptyset$, $M = \emptyset$;
3: **for** $i \in \{1, 2, \cdots, d_x\}$ **do**
4: $\quad \{\mathcal{Y}'_1, \cdots, \mathcal{Y}'_{t_i}\} \leftarrow \mathcal{Y}'$; ▷ clusters based on t_i candidates of x_i.
5: $\quad$ **for** $j \in \{1, 2, \cdots, t_i\}$ **do**
6: $\quad\quad m_j =$ `mean`$(\mathcal{Y}'_j)$; ▷ average vector of $\mathcal{Y}'_j$.
7: $\quad\quad M = M \bigcup \{m_j\}$;
8: $\quad$ **end for**
9: $\quad v_i = \dfrac{\sum_{p,q} ||m_p - m_q||_2}{C_2^{|M|}}, p, q \in \{1, \cdots, t_i\}$;
10: $\quad \boldsymbol{v} = \boldsymbol{v} \bigcup \{v_i\}$;
11: **end for**
12: $\boldsymbol{v} =$ `normalize`$(\boldsymbol{v})$;
13: **return** $\boldsymbol{v}$;

Importance-based SoC Exploration Initialization. To take the importance value into initialization, line 2 in Algorithm 2 uses element-wise multiplication $\odot$ to transform the original design space to ICD space. In this way, design points with similar influences on metrics will move closer, and points with significant differences in metrics will move more separately.

Fig. 3 shows a toy example with 2 design features for transforming from the original space to the ICD space, where feature1 is important and feature2 is less important. After importance-based transformation with $\boldsymbol{v}$, $cfg2'$ will move closer to $cfg1'$, while $cfg3'$ will move further from $cfg1'$. In this way, the importance of parameters is introduced to ICD space when uniformly samples design points for initialization of space exploration. After the exploration, the Pareto optimal set will be transformed into the original space to use the feature parameters to design the optimal SoC.

To summarize the methods mentioned above, we proposed an importance-guided SoC-Init algorithm shown in Algorithm 2. The most significant inputs of the algorithm are the original design space, and the ICD values obtained from Algorithm 1. Given the ICD design space $\mathcal{X}'$ after pruning (line 1) and space transformation (line 2), the SoC-Init algorithm will sample a subset $\mathcal{Z} \in \mathcal{X}'$ for initialization of exploration. In line 3, $\boldsymbol{K} = \boldsymbol{K}_{\mathcal{X}'\mathcal{X}'} \in \mathbb{R}^{|\mathcal{X}'|\times|\mathcal{X}'|}$ is the distance matrix of all design points in $\mathcal{X}'$, and $\Phi(\boldsymbol{x}'_i, \boldsymbol{x}'_j) \in \boldsymbol{K}_{\mathcal{X}'\mathcal{X}'}, i, j \in \{1, 2, \cdots, |\mathcal{X}'|\}$ is computed as Euclidean distance, with $\boldsymbol{x}'_i, \boldsymbol{x}'_j \in \mathcal{X}'$. To make the initial configurations have higher diversity and scatter the whole design space, we use the TED [20] method to sample design points from the ICD space. The design points that contribute most to initialization will be sampled by Algorithm 2.

Algorithm 2 SoC-Init$(\mathcal{X}, u, b, \boldsymbol{v}, v_{th})$

Input: $(\mathcal{X}, u, b, \boldsymbol{v}, v_{th})$, where $\mathcal{X}$ is the un-sampled design space, μ is the normalization coefficient, b is the number of configurations we will sample, $\boldsymbol{v}$ is the ICD feature vector from Algorithm 1.
Output: $\mathcal{Z}$, the sampled set with $|\mathcal{Z}| = b$.

$\mathcal{Z} = \emptyset$; if $v_i < v_{th}$, then$\forall \boldsymbol{x} \in \mathcal{X}, \boldsymbol{x}_i = \text{medium}(\{\boldsymbol{x}_i^1, \cdots, \boldsymbol{x}_i^j\})$;
$\mathcal{X}' = \{\boldsymbol{v} \odot \boldsymbol{x}, \forall \boldsymbol{x} \in \mathcal{X}\}$;
$\boldsymbol{K} = \{\Phi(\boldsymbol{x}_i', \boldsymbol{x}_j') | \boldsymbol{x}_i', \boldsymbol{x}_j' \in \mathcal{X}'\}$; ▷ $\Phi(.)$ is Euclidean distance.
for $i \in \{1, 2, \cdots, b\}$ **do**
  $\boldsymbol{z} = \text{argmax}_{\boldsymbol{x}' \in \mathcal{X}'} = \frac{||\boldsymbol{K}_{\boldsymbol{x}'}||^2}{\Phi(\boldsymbol{x}', \boldsymbol{x}') + \mu}$; ▷ $\boldsymbol{K}_{\boldsymbol{x}'}$ and $\Phi(\boldsymbol{x}', \boldsymbol{x}')$ are $\boldsymbol{x}'$'s corresponding column and diagonal entry in $\boldsymbol{K}$.
  $\mathcal{Z} = \mathcal{Z} \bigcup \{\boldsymbol{z}\}$;
  $\boldsymbol{K} = \boldsymbol{K} - \frac{\boldsymbol{K}_z \boldsymbol{K}_z^\top}{\Phi(\boldsymbol{z}, \boldsymbol{z}) + \mu}$;
end for
return $\mathcal{Z}$;

Algorithm 3 SoC-Tuner$(\mathcal{X}, T, n, u, b, v_{th})$

Input: $\mathcal{X}$ is the unsampled SoC design space, T is the maximal iteration number of BO, n is the trail times of importance analysis, μ is the normalization coefficient, b is the number of samples for initialization.
Output: The Pareto optimal set $\mathcal{X}^*$ and corresponding $\mathcal{Y}^*$.

$\boldsymbol{v} =$ ICD$(\mathcal{X}, n)$; ▷ Algorithm 1.
$\mathcal{Z} =$SoC-Init$(\mathcal{X}, \mu, b, \boldsymbol{v}, v_{th})$; ▷ Algorithm 2.
$\mathcal{X}' = \{\boldsymbol{v} \odot \boldsymbol{x}, \forall \boldsymbol{x} \in \mathcal{X}\}$;
$\boldsymbol{y} \leftarrow$ VLSIFlow$(\mathcal{Z})$;
for $i \in \{1, 2, ..., T\}$ **do**
  Construct the Pareto optimal set $\mathcal{Y}^*$ from $\boldsymbol{y}$;
  $\boldsymbol{x}^* \leftarrow$ IMOO$(\mathcal{X}', \mathcal{Y}^*, \boldsymbol{\theta})$; ▷ Equation (11).
  $\mathcal{Z} = \mathcal{Z} \bigcup \{\boldsymbol{x}^*\}$, $\boldsymbol{y} = \boldsymbol{y} \bigcup \{$VLSIFlow$(\boldsymbol{x}^*)\}$;
  $\boldsymbol{\theta}$ is optimized via gradient descent method;
end for
Construct Pareto optimal set $\mathcal{Y}^*$ from $\mathcal{Z}$, and restore the corresponding $\mathcal{X}^*$ from the ICD space;
return $\mathcal{X}^*$

D. Multi-Objective Exploration with Information Gain

Even though we have carefully chosen the initial set $\mathcal{Z}$ via the SoC-Init algorithm, building a model that can mimic the relationship between the configurations and the objectives is not easy. Witnessing that the Gaussian process (GP) shows robustness and non-parametric approximation in various domains [3], [21], [22], we choose GP as our surrogate model.

We have the ICD design space $\mathcal{X}'$ consisting of design parameters, and according to different $\boldsymbol{x}' \in \mathcal{X}'$, we can get metrics space $\mathcal{Y}$ with time-consuming VLSI Flow shown in Fig. 2. GP provides a prior over the function $f(\boldsymbol{x}') \sim \mathcal{GP}(\boldsymbol{\mu}, k_{\boldsymbol{\theta}})$, where $\boldsymbol{\mu}$ is the mean value and the kernel function k is parameterized by $\boldsymbol{\theta}$. All the objective functions (design metrics) can be expressed as a group of GP models and combined as Equation (3).

$$\mathbb{F} = [f(\boldsymbol{x}_1'), f(\boldsymbol{x}_2'), \cdots, f(\boldsymbol{x}_n'))]^T \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}), \tag{3}$$

where $\boldsymbol{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}$ is the intra-covariance matrix among all feature vectors and can be computed via $[\boldsymbol{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}}]_{ij} = k_{\boldsymbol{\theta}(\boldsymbol{x}_i', \boldsymbol{x}_j')}$, and Gaussian noise $\mathcal{N}(f(\boldsymbol{x}'), \sigma_e^2)$ is to model uncertainties of GP models. Given a newly sampled feature vector $\boldsymbol{x}_*'$, the predictive joint distribution f_* based on $\boldsymbol{y}$ is calculated by Equation (4).

$$f_*|\boldsymbol{y} \sim \mathcal{N}(\begin{bmatrix} \boldsymbol{\mu} \\ \mu_* \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}_{\mathcal{X}'\mathcal{X}'|\boldsymbol{\theta}} + \sigma_e^2 \mathbf{I} & \boldsymbol{K}_{\mathcal{X}'\boldsymbol{x}_*'|\boldsymbol{\theta}} \\ \boldsymbol{K}_{\boldsymbol{x}_*'\mathcal{X}'|\boldsymbol{\theta}} & k_{\boldsymbol{x}_*'\boldsymbol{x}_*'|\boldsymbol{\theta}} \end{bmatrix}). \tag{4}$$

By maximizing the marginal likelihood of GP, $\boldsymbol{\theta}$ is optimized to sense the entire design space. Each time we get $\boldsymbol{y}$ from VLSI Flow, $\boldsymbol{\theta}$ will be updated to better mimic the complex relationship between the design space $\mathcal{X}'$ and metrics space $\mathcal{Y}$.

Therefore, deciding the next $\boldsymbol{x}'$ to be sent to the VLSI flow is important to optimize the surrogate model. From each $\boldsymbol{y}$ obtained from VLSI, we need to maximize the information gained about the Pareto optimal set $\mathcal{Y}^*$ as much as possible. So we develop an information gain-based acquisition function $I(\boldsymbol{x}')$ expressed with entropy $H(\cdot)$ as follows.

$$I(\boldsymbol{x}') = H(\mathcal{Y}^*|\mathcal{X}') - \mathbb{E}_{\boldsymbol{y}}[H(\mathcal{Y}^*|\mathcal{X}' \cup \{\boldsymbol{x}', \boldsymbol{y}\})] \tag{5}$$

$$= H(\boldsymbol{y}|\mathcal{X}', \boldsymbol{x}') - \mathbb{E}_{\mathcal{Y}^*}[H(\boldsymbol{y}|\mathcal{X}', \boldsymbol{x}', \mathcal{Y}^*)] \tag{6}$$

$$\simeq H(\boldsymbol{y}|\mathcal{X}', \boldsymbol{x}') - \frac{1}{S}\sum_{s=1}^{S}[H(\boldsymbol{y}|\mathcal{X}, \boldsymbol{x}', \mathcal{Y}_s^*)], \tag{7}$$

where Equation (7) is approximately computed via Monte-Carlo sampling, and S is the number of samples and $\mathcal{Y}_s^*$ denote a sampled Pareto optimal set.

The value of each element of $\boldsymbol{y}$ in Equation (7) is upper bounded by the maximum value of the corresponding element in sampled point on Pareto optimal set $\mathcal{Y}^*$. We can combine the boundedness property and the fact that each sampled objective function is modeled as a GP prior, and treat each component of $\boldsymbol{y}$ as a truncated Gaussian distribution. Then we can rewrite Equation (7), and obtain the approximation of the acquisition function,

$$AF(i, \boldsymbol{x}') = \sum_{s=1}^{S} \frac{\gamma_s^i(\boldsymbol{x}')\phi(\gamma_s^i(\boldsymbol{x}'))}{2\phi(\gamma_s^i(\boldsymbol{x}'))} - ln(\phi(\gamma_s^i(\boldsymbol{x}'))), \tag{8}$$

$$I(\boldsymbol{x}') \simeq \sum_{i \in \mathcal{I}} AF(i, \boldsymbol{x}'), \mathcal{I} = \{f_1, \cdots, f_n\}, \tag{9}$$

where γ and ϕ stand for the probability density function and the cumulative density function of a standard Gaussian distribution, respectively. $\gamma_s^i(\boldsymbol{x}')$ equals $\frac{\boldsymbol{y}_s^* - \mu_s(\boldsymbol{x}')}{\sigma_s(\boldsymbol{x}')}$ with $\boldsymbol{y}_s^*$ is the maximum value among the sampled points on predicted Pareto optimal set for the i^{th} objective.

Ultimately, we can choose $\boldsymbol{x}^*$ that maximizes the Equation (10) as the next design point sent to VLSI flow:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}'}{\text{argmin}}\, I(\boldsymbol{x}'), \tag{10}$$

where $\boldsymbol{x}^*$ will bring the most information gain. To conclude Equation (3) to (10), we can design an information-gain-based multi-objective optimization (IMOO):

$$\boldsymbol{x}^* \leftarrow \text{IMOO}(\mathcal{X}', \mathcal{Y}^*, \boldsymbol{\theta}), \tag{11}$$

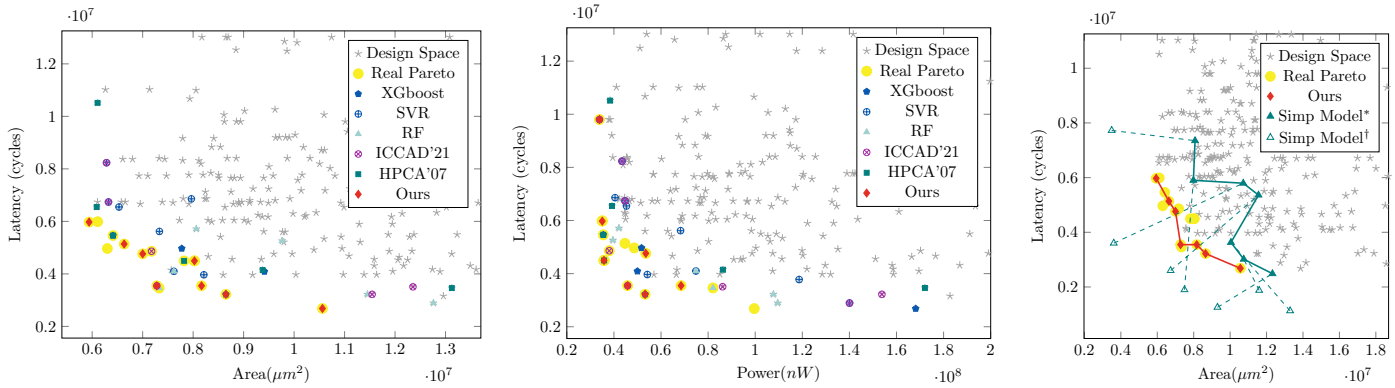
where $\mathcal{X}'$ is ICD Space given by Algorithm 2, $\mathcal{Y}^*$ is the current Pareto optimal set, and $\boldsymbol{\theta}$ is optimized to better mimic the surrogate GPs model. We combine all the proposed algorithms into the overall algorithm illustrated in Algorithm 3. The maximal number of exploration rounds is T, and the output of SoC-Tuner is the learned Pareto optimal set.

IV. Experiment & Analysis

A. Benchmarks and Baselines

To evaluate the SoC design uniformly, we randomly sample 2500 design points in TABLE I. Each design is evaluated with the VLSI flow in Fig. 2, and the metric data are collected to verify various methods. We choose some popular DNNs including ResNet50 [23], MobileNet [24], and Transformer [25] as our benchmarks.

Several representative baselines are compared with SoC-Tuner. The MicroAL-based method [9] (ICCAD'21) is used to predict the power and latency of the BOOM core. The regression-based method [19] (HPCA'07) leverages regression models with non-linear transformations to explore the power-performance Pareto curve. We implement the key methods of these works and adopt them into



(a) Learned Pareto optimal set (inference latency *vs* chip area)

(b) Learned Pareto optimal set (inference latency *vs* power consumption)

(c) Simplified model (*actual metrics, † reported metrics) *vs* RTL-simulation

Fig. 4 The Pareto optimal set is given by various methods (ResNet50). (a) and (b) demonstrate the effectiveness of our framework. (c) shows that the metrics from the simplified model have a big gap with that from the VLSI flow.

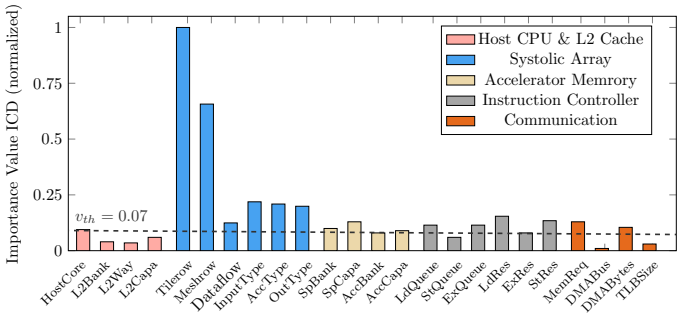


Fig. 5 Importance analysis of the parameters given by ICD algorithm($n = 30, v_{th} = 0.07$).

SoC design exploration, then compare them with our framework. Moreover, we also compare SoC-Tuner with traditional multi-objective optimization methods like XGBoost [26], random forest (RF), and support vector regression (SVR). Simulated annealing is leveraged for these traditional algorithms. We implement all methods in Python and all experiments are conducted on a Linux server with Intel Xeon CPU (E5-2630 v2@2.60GH) and 256 GB RAM.

B. Experiment Setting

We utilize the Chipyard to generate SoC hardware design as shown in Fig. 2. We use the ASAP7 process design kit (PDK) as the standard cell library and use tools integrated into Hammer [27] to execute the VLSI flow. The RTL-level simulation tool Verilator can obtain the accurate overall inference latency of various DNNs.

For the experimental setting, we set $v_{th} = 0.07$ for pruning design space, $u = 0.1$, and $b = 20$ for the SoC-Init algorithm. For a fair comparison, we keep the exploration iteration the same for baselines and our methods. All experiments and baselines are repeated 10 times to get the corresponding average results.

In comparison with baselines, the average distance to reference set (ADRS) shown in Equation (12) is widely used in design space exploration to measure the distance between the learned Pareto optimal set with the real Pareto-optimal set of the design space.

$$ADRS(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma, \omega \in \Omega} \min \quad f(\gamma, \omega), \tag{12}$$

where f is the Euclidean distance function, Γ is the real Pareto optimality set and Ω is the learned Pareto optimal set.

C. Experimental Result and Analysis

Fig. 5 demonstrates the result of importance analysis based on the ICD algorithm ($n = 30$, $v_{th} = 0.07$). With the ICD algorithm, the whole design space points are pruned by about 30.16%.

Learned Pareto Optimal Set. We choose the benchmark ResNet50 as an example to show the superiority of SoC-Tuner in finding the Pareto optimal set of SoC design. In Fig. 4, gray points represents various design configuration of SoC, and colorful points are learned Pareto optimal set explored by SoC-Tuner and other methods. We only draw the design points close to the real Pareto optimal set to show the result clearly. Both in latency-area space shown in Fig. 4(a) and latency-power space shown in Fig. 4(b), yellow circles represent the real Pareto design, and red diamonds represent the learned Pareto design by SoC-Tuner. The learned Pareto optimal set of SoC-Tuner is much closer to the real Pareto optimal set than other methods, demonstrating that SoC-Tuner's effectiveness outperforms other methods in finding the Pareto optimal set.

Moreover, we use the simplified model [6] to explore space according to its inaccurate metrics. To show the gap between the simplified model and RTL simulation, we simultaneously draw their learned Pareto optimal set in Fig. 4(c). The green hollow triangles represent Pareto optimal set found by the simplified model, while solid triangles represent the actual metrics from VLSI flow with the same design parameter. Fig. 4(c) proves that the simplified model cannot effectively guide the design of space exploration.

Learning Convergence and Optimal SoC Design. Fig. 7(a) shows the ADRS in each exploration round. In each exploration round, SoC-Tuner outperforms previous methods. Fig. 7(a) demonstrates that SoC-Tuner has higher exploration efficiency than other methods, giving a better SoC design in a shorter time.

The optimal design points from the learned Pareto optimal sets given by various methods are listed in Fig. 4. Since Transformer has too many parameters to be simulated in an acceptable time, we evaluate the inference latency on the 6 basic structures, *i.e.*, Decoder. Fig. 6 compares the inference cycles of optimal SoC designs explored by various methods, showing that the SoC designed by SoC-Tuner can get the least inference latency on various DNN workloads. The inference speed demonstrates our framework can find the optimal SoC design to obtain high performance in DNN acceleration. Moreover, our framework can facilitate SoC designers to design practical SoCs for DNN acceleration, instead of staying inaccurate simulation stage like

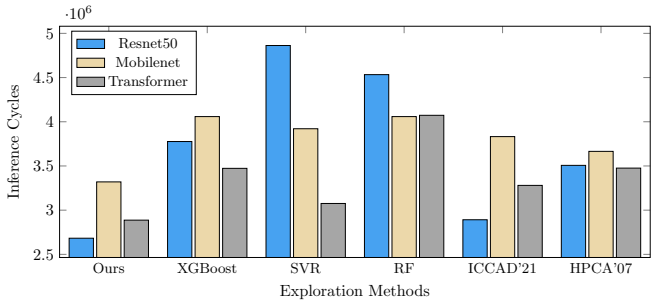
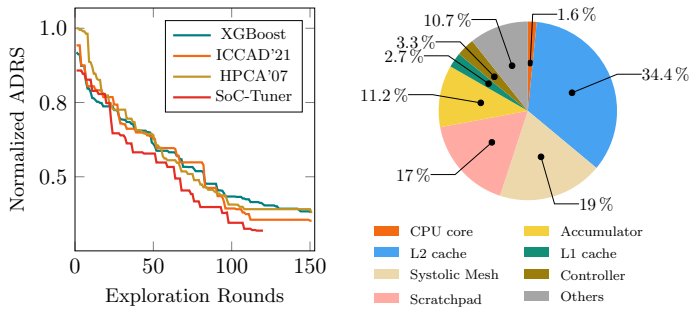


Fig. 6 Inference cycles of DNNs on the learned optimal SoC designs obtained by different exploration methods.



(a) (b)

Fig. 7 Experimental results. (a) ADRS curves of various methods; (b) Area breakdown of optimal SoC.

previous simplified analytical tools. With the learned Pareto optimal set, we can implement the optimal SoC design and Fig. 7(b) shows the area breakdown of the design given by VLSI flow.

V. Conclusion

In this paper, we have proposed SoC-Tuner, a novel exploration framework that utilizes a series of importance-guided algorithms to reduce the design iterations and find the Pareto optimal set of SoC configurations. Our framework thoroughly constructs a huge design space and analyzes the importance of design parameters in a typical DNN-targeting SoC. The framework provides a group of efficient algorithms to prune the original design space and initialize the exploration. Moreover, we utilize a novel multi-objective exploration with information gain to find the optimal SoC design for DNN accelerations. For designers, our framework can help them design A high-performance and low-cost SoC for various DNN applications on edge devices. For researchers, our framework brings more insights into the community of hardware design space. In future work, we plan to extend our framework to support more complicated DNN models like large language models and introduce more design constraints such as reliability, security, and robustness.

Acknowledgements

This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, Hong Kong SAR.

References

[1] Q. Huang, C. Hong, J. Wawrzynek, M. Subedar, and Y. S. Shao, "Learning a continuous and reconstructible latent space for hardware accelerator design," in *Proc. ISPASS*. IEEE, 2022, pp. 277–287.

[2] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators," *IEEE TC*, vol. 70, no. 8, pp. 1160–1174, 2021.

[3] W. Zhao, Y. Bai, Q. Sun, W. Li, H. Zheng, N. Jiang, J. Lu, B. Yu, and M. D. Wong, "A high-performance accelerator for super-resolution processing on embedded gpu," *IEEE TCAD*, 2023.

[4] Y. Bai, X. Yao, Q. Sun, and B. Yu, "Autogtco: Graph and tensor co-optimize for image recognition with transformers on gpu," in *Proc. ICCAD*, 2021.

[5] Y. Bai, X. Yao, Q. Sun, W. Zhao, S. Chen, Z. Wang, and B. Yu, "Gtco: Graph and tensor co-design for transformer-based image recognition on tensor cores," *IEEE TCAD*, 2023.

[6] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic cnn accelerator simulator," *arXiv preprint*, 2018.

[7] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *Proc. ISPASS*. IEEE, 2019, pp. 304–315.

[8] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proc. MICRO*, 2019.

[9] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *Proc. ICCAD*, 2021.

[10] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, 2016.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, 2017.

[12] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS, UCB Tech Repo*, 2016.

[13] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proc. DAC*. IEEE, 2012, pp. 1212–1221.

[14] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, 2020.

[15] K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," University of California at Berkeley Berkeley United States, Tech. Rep., 2015.

[16] Y. Lee, C. Schmidt, A. Ou, A. Waterman, and K. Asanovic, "The hwacha vector-fetch architecture manual, version 3.8. 1," *EECS, UCB Tech Repo*, 2015.

[17] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*. IEEE, 2021, pp. 769–774.

[18] T. Chen, Q. Guo, K. Tang, O. Temam, Z. Xu, Z.-H. Zhou, and Y. Chen, "Archranker: A ranking approach to design space exploration," *ACM SIGARCH Computer Architecture News*, 2014.

[19] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *Proc. HPCA*. IEEE, 2007.

[20] K. Yu, J. Bi, and V. Tresp, "Active learning via transductive experimental design," in *Proc. ICML*, 2006.

[21] Q. Sun, C. Bai, H. Geng, and B. Yu, "Deep neural network hardware deployment optimization via advanced active learning," in *Proc. DATE*. IEEE, 2021.

[22] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A Pareto driven machine learning approach," *IEEE TCAD*, 2018.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint*, 2017.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proc. NeurIPS*, 2017.

[26] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. KDD*, 2016.

[27] H. Liew, D. Grubb, J. Wright, C. Schmidt, N. Krzysztofowicz, A. Izraelevitz, E. Wang, K. Asanović, J. Bachrach, and B. Nikolić, "Hammer: a modular and reusable physical design flow tool," in *Proc. DAC*, 2022.