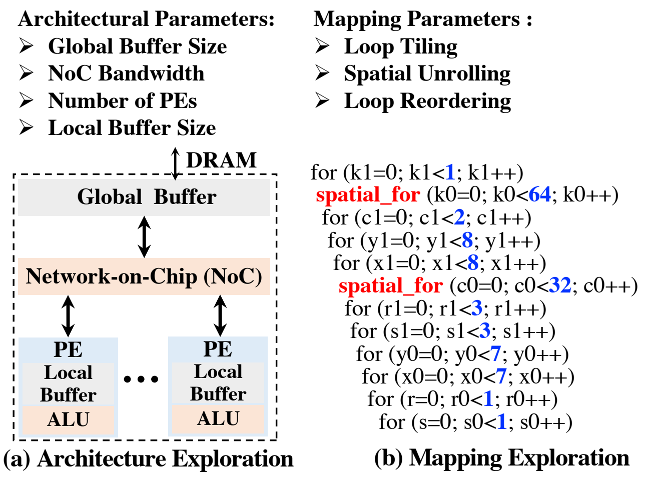
Poros: One-Level Architecture-Mapping Co-Exploration for Tensor Algorithms

Fuyu Wang
fuyuwang17@gmail.com

Minghua Shen
shenmh6@mail.sysu.edu.cn

School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China

***Abstract*—Tensor algorithms increasingly rely on specialized accelerators to meet growing performance and efficiency demands. Given the rapid evolution of these algorithms and the high cost of designing accelerators, automated solutions for jointly optimizing both architectures and mappings have gained attention. However, the joint design space is non-convex and non-smooth, hindering the finding of optimal or near-optimal designs. Moreover, prior work conducts two-level exploration, resulting in a combinatorial explosion. In this paper, we propose Poros, a one-level architecture-mapping co-exploration framework. Poros directly explores a batch of architecture-mapping configurations and evaluates their performance. It then exploits reinforcement learning to perform gradient-based search in the non-smooth joint design space. By sampling from the policy, Poros keeps exploring new actions to address non-convexity. Experimental results demonstrate that Poros achieves up to 5.32× and 2.15× better EDP compared with hand-designed accelerators and state-of-the-art automatic approaches respectively. Through one-level exploration scheme, Poros also converges at least 20% faster than other approaches.**



```
for (k1=0; k1<1; k1++)
 spatial_for (k0=0; k0<64; k0++)
  for (c1=0; c1<2; c1++)
   for (y1=0; y1<8; y1++)
    for (x1=0; x1<8; x1++)
     spatial_for (c0=0; c0<32; c0++)
      for (r1=0; r1<3; r1++)
       for (s1=0; s1<3; s1++)
        for (y0=0; y0<7; y0++)
         for (x0=0; x0<7; x0++)
          for (r=0; r0<1; r0++)
           for (s=0; s0<1; s0++)
```

Fig. 1. The architecture design and mapping in specialized accelerators.

I. Introduction

Tensor algorithm is fundamental for many applications such as deep learning [1], scientific computing [2], and social networks [3]. Due to the immense computational cost, specialized accelerators [4]–[8] are developed using FPGAs and ASICs. They organize large arrays of processing elements (PEs) and a multi-level memory hierarchy in a spatial architecture. Compared to general-purpose CPUs and GPUs, these accelerators can provide orders-of-magnitude improvements in performance and energy efficiency for tensor algorithms. On the software hand, programmers manually develop high-performance libraries [9]–[11] to harness the computational abilities of diverse accelerators. Unfortunately, designing such accelerators requires a deep understanding of architectures, compilers, and algorithms, leading to a lengthy cycle.

Modern tensor algorithms vary in input shapes and operators (e.g., general matrix multiplication, convolution), exhibiting a wide range of computational characteristics and requirements. To reduce the cost of accelerator design, recent work [12]–[14] focuses on jointly optimizing both hardware architecture and software mapping, as shown in Fig. 1. Architecture design space exploration involves determining parameters such as on-chip buffer sizes, network-on-chip (NoC) bandwidth, and the number of PEs to meet the budget and deployment requirements of tensor algorithms. Mapping space exploration determines parameters including loop tiling, spatial unrolling, and loop reordering to control computation and data movement patterns on accelerators. While architecture-mapping co-exploration is a holistic paradigm, it faces the following challenges.

This work was supported by the National Key R&D Program of China under Grant No. 2024YFB4505205, the Natural Science Foundation of China under Grant No. 62072479 and 62332021, the Guangdong Natural Science Foundation under Grant No. 2024B1515020011, the PCL-CMCC Foundation for Science and Innovation under Grant No. 2024ZY2B0060, the Open Research Fund from Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ) under Grant No. GML-KF-24-12, and the Xiaomi Young Talents Program. Minghua Shen is the corresponding author of this paper.

First, the architectural and mapping parameters form a large design space. This leads to significant variability in accelerator performance under different architecture-mapping configurations. The optimal configuration is also sensitive to the specific tensor algorithm. As a result, it is infeasible to find the optimal configuration through exhaustive search when retargeting new tensor algorithms. Second, the joint design space is characterized by both non-convexity and non-differentiability. Relying on black-box optimization [15] methods in such a high-complexity space struggles to converge quickly to high-quality configurations. Finally, prior work [13], [14] conducts two-level exploration to derive ultimate solutions, as illustrated in Fig.2(a). In the outer level, they sample a batch of architecture configurations. The performance of each architecture configuration is measured by exploring the mapping space in the inner level. This two-level explorer requires a large number $(n * m)$ of measurement trials, where n is the number of architecture configurations and m is the number of mapping configurations for each architecture.

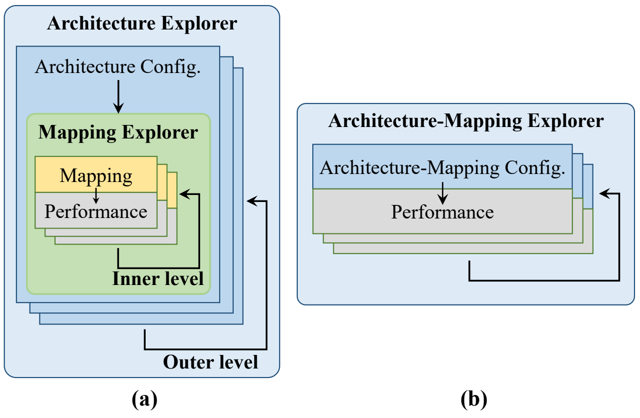


Fig. 2. Two-level (a) and One-level (b) exploration approaches.

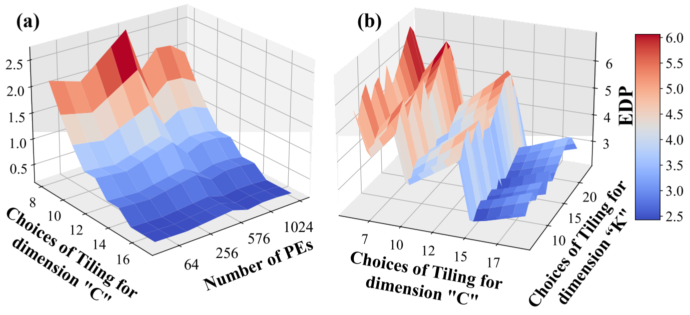


Fig. 3. Cost surface of the accelerator when using different number of PEs and choices of loop tiling for a 2D convolution of ResNet50 [26]. Z-axis represents the normalized cost (EDP).

This paper addresses the challenges by proposing a one-level architecture-mapping co-exploration framework, as shown in Fig. 2(b). Different from two-level approaches, our framework directly explores a batch of architecture-mapping configurations and evaluates their performance at each iteration. The key idea is to construct a constrained architecture-mapping joint design space and apply a reinforcement learning-based exploration algorithm. The joint space filters out low-quality candidates based on user constraints (e.g., power and area). Reinforcement learning (RL) is a goal-oriented decision-making process, where the agent can 1) leverage deep neural networks to model a policy and keep exploring new actions to handle non-convexity. 2) perform gradient-based search in the non-smooth joint design space.

We evaluate Poros using different tensor algorithms. Compared with hand-designed accelerators including Eyeriss [5] and NVDLA [7], Poros achieves 5.32× to 3.36× better EDP on average. Compared with state-of-the-art automatic approaches such as HASCO [13] and Spotlight [14], Poros achieves 2.40× to 2.15× better EDP on average.

This paper makes the following contributions:

- We introduce Poros, a one-level architecture-mapping co-exploration framework to automatically design specialized accelerators for target tensor algorithms. Through one-level co-exploration, Poros converges at least 20% faster than state-of-the-art approaches.
- We formulate the architecture-mapping co-exploration as a reinforcement learning problem, enabling the agent to handle non-convexity and perform gradient-based search.

II. Background

A. Specialized Accelerators and Tensor Mappers

Recent specialized accelerators [6], [16], [17] leverage spatial architectures to accelerate matrix multiplications. The spatial architecture in Fig. 1(a) consists of an array of processing elements (PEs) to exploit computation parallelism, and a multi-level memory hierarchy to exploit data reuse. Each PE contains multiply-accumulate (MAC) units for computation. In the multi-level memory hierarchy, the data of matrices are tiled and transferred between DRAM and PE. In addition, commercial accelerators have more complex architecture components including switches, PE interconnections, and controller [18]. We consider such architectural parameters in future work.

Traditional works [9], [10] provide hand-tuned libraries by manipulating optimization options, which imposes non-trivial labor effort and long development cycles. To ease the process, recent automatic mappers [19]–[22] perform mapping exploration shown in Fig. 1(b) to find optimized mappings for target tensor algorithms. TVM [19] is a full-stack tensor compiler for diverse hardware platforms. It defines a mapping search space through templates, coupled with a cost model using XGBoost. Based on TVM, FlexTensor [21] first prunes the mapping space and then employs simulated annealing [23] to select the starting point. Subsequently, it uses Q-learning to forecast the most favorable search direction. CoSA [24] and Mind Mappings [25] leverage integer linear programming and a learnable cost model respectively to generate high-performance programs.

B. Motivation

Architecture-mapping co-exploitation faces the following major challenges:

Large Joint Design Space. The joint design space complexity for co-exploration is the Cartesian product of the architecture design space complexity (O_{arch}) and the mapping search space complexity (O_{map}): $O_{arch} \times O_{map}$. For instance, consider ResNet50 [26] as a tensor algorithm, consisting of multiple operators. The order of the joint space is up to 10^{40} [13], [14], [27]. To find the optimal configuration, a performance cost model (e.g., MAESTRO [28] and Timeloop [29]) is employed to evaluate the quality of each architecture-mapping candidate. Although evaluating each candidate takes only milliseconds, the large joint design space makes it time-consuming to evaluate all candidates. Previous work [12], [27] reduces space complexity by fixing the loop tiling size and choosing one dataflow from three candidates: weight-stationary, row-stationary, and output-stationary. As a result, this approach may miss better architecture-mapping configurations.

Non-Convex and Non-Differentiable Space. The goal of this problem is to optimize a performance metric of the accelerator, such as energy-delay product (EDP). Fig. 3 plots the cost surface for a 2D convolution in ResNet50. In this example, we vary the number of PEs and loop tiling sizes for the "K" and "C" dimensions, and present the cost in terms of EDP. The joint design space is non-convex, containing many sub-optimal points. It is also non-differentiable, as minor changes to parameters can result in non-smooth changes to the cost. This challenge makes black-box optimization approaches [15] struggle to find high-quality solutions within a few iterations.

Two-level Explorers. As depicted in Fig. 2, most prior efforts [13], [14], [30] consider the architecture-mapping co-exploration as a two-level process, exploring each design space alternately. They start by sampling architecture configurations in the outer level and then explore optimized mappings for each architecture in the inner level. The returned mappings are used to produce performance feedback for the outer-level architecture design space exploration. When exploring n architecture configurations in the outer level and m mappings in the inner level, the two-level approaches require $n{*}m$ measurement trials at each iteration. Spotlight [14] sets both m and n to 100, leading to high search overheads.

To address the challenges, we propose Poros, a one-level architecture-mapping co-exploration framework based on reinforcement learning (RL) method. First, Poros directly explores a batch of architecture-mapping configurations and evaluates their performance at each iteration, as shown in Fig. 2(b). In practice, this batch size is set to 32, which is considerably smaller than $n * m$ (i.e., 10000) required by previous approaches. Second, Poros constructs a constrained design space based on user constraints to filter out low-quality candidates. Finally, Poros exploits RL to perform gradient-based search in the non-smooth joint design space. By sampling from the policy (or probability distribution), Poros keeps exploring new actions to handle non-convexity. In this way, Poros can significantly enhance exploration efficiency compared to state-of-the-art approaches.

III. Poros

A. Overview

Fig. 4 shows an overview of our Poros. The input describes the operators in a tensor algorithm along with the user constraints. Poros leverages reinforcement learning to provide an optimized architecture-mapping configuration through two key processes: generation and feedback. Based on the resulting configuration, we can implement a specialized accelerator with intrinsics for the tensor algorithm using off-the-shelf hardware generators [6], [7], [18], [31], and write a tensor program for each operator.

Generation. Poros begins by constructing a state space composed of all architecture-mapping parameters. The policy-value network is employed to sample a batch of values for each parameter from its action space. These parameters are then collected to form a state, which serves as the input to the policy-value network. Poros constructs an action space for each parameter based on the analysis of tensor shapes and computations. This action space is constrained to filter out low-quality candidates. In this process, a batch of architecture-mapping configurations is generated.

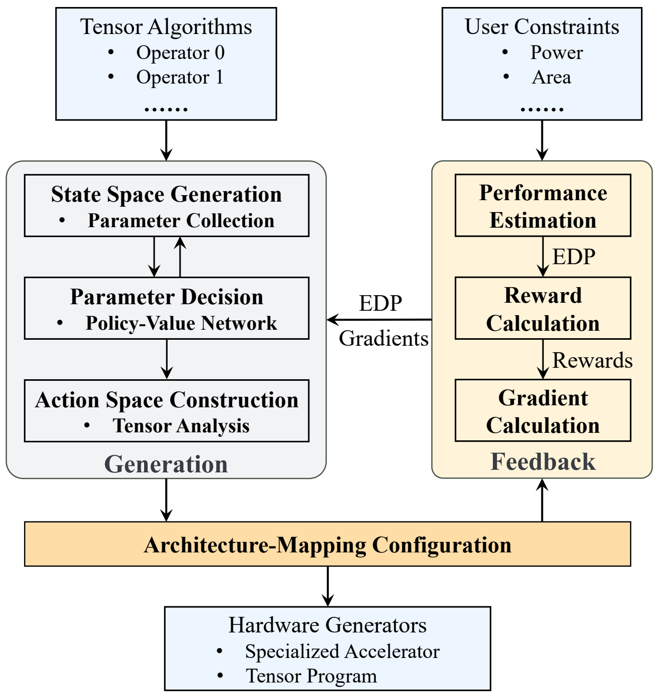


Fig. 4. Overview of Poros.

Feedback. Poros executes a performance model to evaluate the energy-delay-product (EDP) of a batch of architecture-mapping configurations from the generation process. Based on the EDP metric, rewards and gradients are calculated to optimize the policy network. The best-performing configuration is selected from the batch.

B. Problem Setup

Given the target tensor algorithm $\mathcal{T}$, we define an architecture-mapping configuration $\boldsymbol{amc}$ and the joint design space $\mathcal{AMC}$.

Definition 3.1: We define an architecture-mapping configuration as a D-tuple: $amc \in X_0 \times ... \times X_{D-1}$, where D is the number of architecture-mapping parameters, and X_d ($d \in [1, D]$) represents the domain of the d-th parameter.

As shown in Fig. 1, architectural parameters consist of the number of PEs, the size of local buffer in each PE, the size of global buffer shared among PEs, and the bandwidth of NoC, which supports which unicast and multicast. Mapping parameters involve three key loop transformations: loop tiling, spatial unrolling, and loop reordering. Loop tiling splits tensors into sub-tensors to fit into the on-chip buffer, enhancing data reuse. Spatial unrolling determines the parallelism for each loop based on the number of PEs. Loop reordering aims to further improve data reuse by optimizing the execution orders of loops. These parameters can be customized independently for each operator of the target tensor algorithm.

Definition 3.2: We define the joint design space $\mathcal{AMC}$ as:

$$\mathcal{AMC}_{\mathcal{T}} = \{\boldsymbol{amc} \in X_0 \times ... \times X_{D-1} | 0 < \rho(\mathcal{T}, \boldsymbol{amc}) < \rho^*\},$$

where ρ^* represents user-defined constraints, such as maximum delay and power under different scenarios.

In this way, the size of the joint design space (denoted by $|\mathcal{AMC}|$) depends on the architecture-mapping parameters as well as the target tensor algorithm. The complexity of $|\mathcal{AMC}|$ is the Cartesian product of the sub-spaces of all parameters.

Given the target tensor algorithm $\mathcal{T}$, Poros aims to find the configuration $\boldsymbol{amc}^*$ ($\boldsymbol{amc}^* \in \mathcal{AMC}$) that minimizes the cost function $\boldsymbol{f}$. This combinatorial optimization problem can be formulated as:

$$\boldsymbol{amc}^* = \underset{\boldsymbol{amc} \in \mathcal{AMC}_{\mathcal{T}}}{\arg\min} \ \boldsymbol{f}(\mathcal{T}, \boldsymbol{amc}). \tag{1}$$

We assume that $\boldsymbol{f}$ depends on the structure of $\mathcal{T}$ rather than its input. The cost function characterizes the relationship among the target tensor algorithm, the architecture-mapping configuration, and various performance metrics (e.g., delay, power, and energy). As a multi-objective optimization function, $\boldsymbol{f}$ can be formulated as a weighted sum or product of multiple performance metrics by the designer. In this paper, $\boldsymbol{f}$ is formulated as the energy-delay product (EDP), which is a common metric in related work.

C. Generation

State. A state $\mathcal{S}$ is a D-tuple: $[x_0, x_1, ..., x_{D-1}]$, where D is the number of architecture-mapping parameters, and x_d ($d \in [0, D-1]$) represents the value of the d-th parameter. In practice, the state collects the number of PEs ($\#PEs$), the size of local buffer (lbs), the size of global buffer (gbs), the bandwidth of NoC (bw), unrolling loops ($unroll$), and tiling size ($tile$) and ordering ($order$) for each loop:

$$\mathcal{S} = [\#PEs, lbs, gbs, bw, unroll, tile, order].$$

To pre-process the state and stabilize the exploration process, we perform an encoding operation. First, the state is normalized relative to the maximum value of each parameter. Then, we encode the state as a feature vector, which is fed into the policy-value network. When an action $\mathcal{A}_t$ is taken using the network, the state is updated as $\mathcal{S}_{t+1} \leftarrow [\mathcal{S}_t, \mathcal{A}_t]$.

TABLE I
THE CHOICES FOR ARCHITECTURAL PARAMETERS.

Parameters	Choices
#PEs	16, 64, 256, 1024, 4096
Global Buffer (KB)	32, 64, 128, 256, 512
Local Buffer (B)	256, 512, 1024, 2048, 4096
Bandwidth (Words/Cycle)	32, 64, 128, 256, 512

Action Space. In Poros, an action corresponds to determining the value of each parameter. Since the original joint design space is both large and dense, taking proximate actions within this space can result in similar performance metrics. For example, increasing the number of PEs from 16 to 64 could yield a 4× performance improvement, while increasing the number of PEs from 64 to 65 would likely provide almost no benefit. To enable efficient exploration, we construct a sparse action space by pruning candidates that are unlikely to bring good performance through the following ways. For architectural parameters, the action space is defined in Table I according to user constraints. Notably, our framework

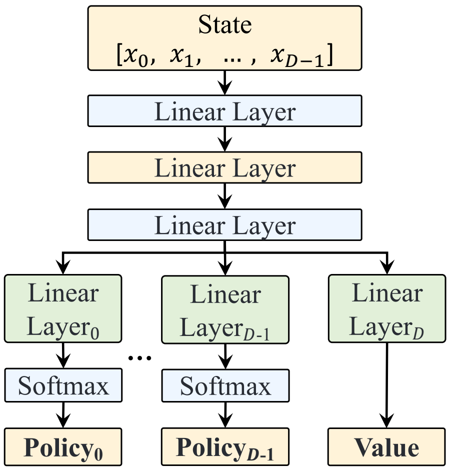
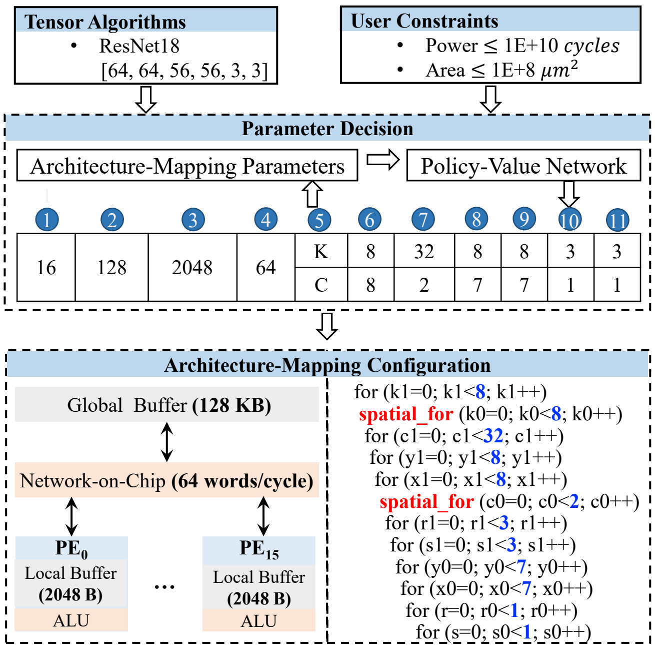


Fig. 5. Neural structure of the policy-value network.

is scalable to support larger architectures (e.g., more PEs and larger buffers), as demonstrated in Section IV-D. For mapping parameters, we fix loop order, and mainly focus on spatial unrolling and loop tiling. Poros begins by selecting specific unrolling options from all tensor dimensions. For example, there are $\binom{7}{2}$ possible choices when unrolling two loops for the 2D convolution shown in Fig. 1. We then formulate tiling as a prime-factor allocation problem since divisible tiles are generally more efficient. Considering that a loop of size l contains c prime factors in total, we can partition the loop to s tiles: $g_1 \times g_2 ... \times g_p = h_1 \times h_2 ... \times h_s = l$. Here g_i represents a prime factor, and h_i represents a tile size. There are $\binom{p}{s-1}$ tiling choices for each loop. In practice, s is set to 2, corresponding to the three-layer memory hierarchy: DRAM, global buffer, and local buffer.

Policy-Value Network. Poros includes a policy-value network to formulate an Actor-Critic algorithm [32]. The policy-value network takes the updated state as its input and produces a policy $\mathcal{P}$ and a value $\mathcal{V}$ at each time step. The policy, represented as a probability distribution over the action space, is sampled to explore new actions. The value is considered as the expected reward based on the current policy. Since the action space for each parameter varies, we design a policy-value network with multiple output branches. Fig. 5 shows the network in detail, composed of several linear layers. First, three fully connected layers followed by ReLU activation encode the state into high-dimensional features. These layers are shared between different parameters. For each parameter, we employ additional linear layers to produce a vector and a value. Finally, the Softmax function is executed over the vector to produce a policy. These layers are unshared between different parameters, allowing for flexibility in handling the variable action space. The hidden size of linear layers is set to 128, making the neural structure relatively simple. Together with the low-dimensional state, the simple network can enhance exploration efficiency.

Example. Fig. 6 presents an example architecture-mapping configuration generated by Poros. To generate such configuration, the agent takes actions to decide each parameter through the policy-value network. At time steps ①-④, the agent decides



```
for (k1=0; k1<8; k1++)
 spatial_for (k0=0; k0<8; k0++)
  for (c1=0; c1<32; c1++)
   for (y1=0; y1<8; y1++)
    for (x1=0; x1<8; x1++)
     spatial_for (c0=0; c0<2; c0++)
      for (r1=0; r1<3; r1++)
       for (s1=0; s1<3; s1++)
        for (y0=0; y0<7; y0++)
         for (x0=0; x0<7; x0++)
          for (r=0; r0<1; r0++)
           for (s=0; s0<1; s0++)
```

Fig. 6. An example of using the policy-value network in Poros to generate an architecture-mapping configuration for a tensor operator in ResNet18 [26].

the number of PEs, the size of local buffer, the size of global buffer, the bandwidth of NoC, respectively. Based on these architectural parameters, the agent decides spatial unrolling and tiling at the following time steps. The agent can tune these parameters after obtaining feedback from Section III-D.

Multi-Operator and Multi-Algorithm Co-Design. Tensor algorithms comprise multiple operators, each of which may require distinct architectural and mapping configurations. During the exploration of architecture-mapping configurations for the target algorithm, architectural parameters are set to the maximum values across all operators, while mapping parameters remain independent for each operator. When an accelerator is developed as an ASIC, it requires to support a variety of tensor algorithms. If target algorithms are known at design-time, architectural parameters are set to the maximum values across all algorithms. If these algorithms are known compile-time or run-time, only mapping parameters are tuned.

D. Feedback

Performance Estimation. To evaluate the cost of each architecture-mapping configuration for the target tensor algorithm, we employ MAESTRO [28] to estimate delay, energy, power, and area. Due to its high accuracy and speed, MAESTRO is widely used in hardware/software co-design [12]–[14].

Reward and Gradient Calculations. A reward serves as guidance to take better actions. In Poros, reward is a function to address EDP: $\mathcal{R} = E_{max} - E$, where E represents the EDP of the current architecture-mapping configuration, and E_{max} represents the maximum EDP observed during the exploration process. This formulation ensures a positive reward for valid configurations, guiding the policy-value network to minimize EDP. When generating invalid configurations that exceed user constraints, the reward is negative to penalize the network: $\mathcal{R} = 2 * E_{max}$. In this way, Poros balances exploration and exploitation for efficient architecture-mapping configurations.

We follow the Actor-Critic algorithm, which combines both policy-based and value-based learning for better exploration efficiency. The gradients of the policy-value network's learnable parameters, $G(\theta)$ are calculated as follows:

$$J(\theta) = -\nabla_\theta \sum_{t=1}^{D} \gamma^t (\mathcal{R}_t - \mathcal{V}_t) \log \mathcal{P}_t, \tag{2}$$

where D represents the number of time steps, $\mathcal{R}_t$, $\mathcal{V}_t$, and $\mathcal{P}_t$ represent the reward, value, and policy at time step t, respectively. The difference $\mathcal{R}_t - \mathcal{V}_t$ is considered as an estimate of the advantage of taking action $\mathcal{A}_t$ under state $\mathcal{S}_t$. This advantage function is beneficial to reduce the variance of estimated gradients. The learnable parameters are then updated through gradient descent.

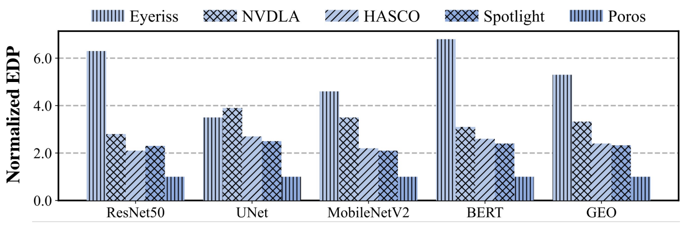


Fig. 7. EDP comparison of Poros against hand-designed accelerators and automatic frameworks for different tensor algorithms.

IV. Evaluation

A. Experimental Setup

We collect a set of tensor algorithms that are popular for image and language processing as benchmarks, including ResNet50 [26], UNet [33], MobileNetV2 [34], and BERT [35]. We compare Poros with both hand-designed accelerators and automatic co-design frameworks. The hand-designed accelerators include Eyeriss [5] and NVDLA [7], which employ row-stationary and weight-stationary mappings, respectively. The automatic frameworks include HASCO [13] and Spotlight [14], both of which perform two-level exploration. We present the EDP results of Poros under iso-sample and iso-time. For iso-sample comparisons, all automatic approaches are performed for a fixed number of performance evaluations. For iso-time comparisons, all approaches are allowed to run until a fixed wall-clock time. We use the microarchitectural model MAESTRO [28] to evaluate the delay, energy, power, and area for each accelerator.

B. Result Analysis

We first provide an iso-sample comparison. For each tensor operator, we run all automatic approaches to up to 500 performance evaluations. Fig. 7 shows the EDP results of Poros for different tensor algorithms. Compared with Eyeriss and NVDLA, the architecture-mapping design points found by Poros achieve 5.32× and 3.36× better EDP on average, respectively. Each tensor algorithm comprises diverse operators, such as transposed convolution, depthwise convolution, and

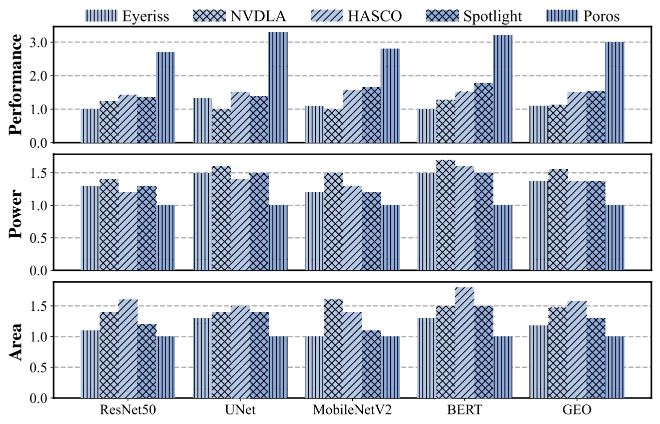


Fig. 8. Comparisons of performance, power and area of Poros against hand-designed accelerators and automatic frameworks.

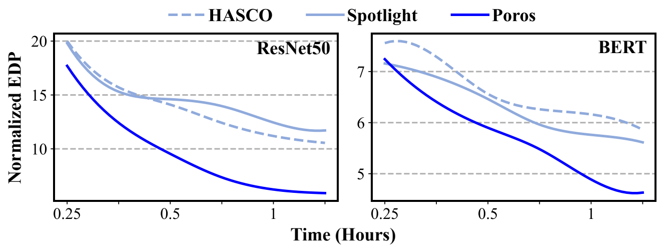


Fig. 9. Normalized EDP of Poros over time for ResNet50 and BERT.

attention mechanism, requiring distinct architecture-mapping configurations. It is non-trivial for hand-designed accelerators to achieve optimal EDP for different algorithms. Compared with HASCO and Spotlight, Poros achieves 2.40× and 2.15× better EDP on average, respectively. Moreover, Poros provides 85.6%, 27.3%, and 23.1% improvements in performance, power, and area respectively, over these two automatic co-design approaches. The results of performance, power, and area are illustrated in Fig. 8.

Moreover, Poros incurs at least 20% lower search costs compared to HASCO and Spotlight. Poros can converge to better or comparable design points within a fixed wall-clock time. Fig. 9 presents the EDP results of each co-exploration approach over wall-clock time for two tensor algorithms. This advantage of Poros primarily stems from its one-level exploration scheme with batch sampling in the joint design space.

C. Multi-Algorithm Co-Exploration

Fig. 10 shows the EDP results of Poros when targeting multiple tensor algorithms simultaneously. We first assume that all tensor algorithms are known at design-time. We set the architectural parameters of the accelerator to the maximum values across all algorithms. Compared with Eyeriss, NVDLA, HASCO, and Spotlight, Poros achieve 2.81×, 2.70×, 2.16×, and 1.87× better EDP on average, respectively. Then we assume BERT is unknown at design time. Poros automatically designs a specialized accelerator for ResNet50, UNet, and MobileNetV2, and tunes the mapping parameters for BERT on the resulting accelerator. In this scenario, Poros still provides better EDP compared to hand-designed accelerators.

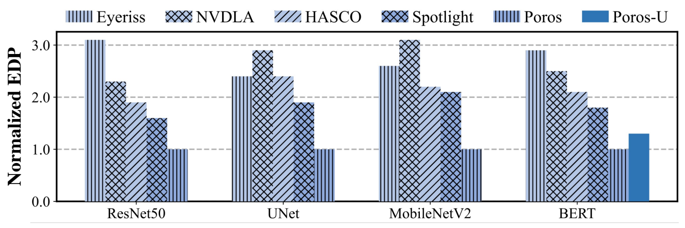


Fig. 10. EDP comparison of Poros against hand-designed accelerators and automatic frameworks for multi-algorithm co-design. “Poros-U” represents that BERT is unknown at design-time.

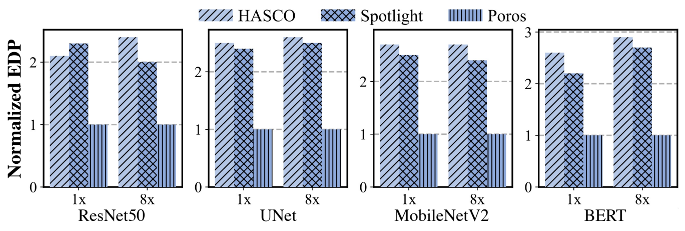


Fig. 11. EDP comparison between HASCO, Spotlight, and Poros when scaling up the power constraint from $10W$ to $80W$.

D. Scalability Analysis

We evaluate Poros’s scalability on platforms that support higher power. In this experiment, we increase the power constraint from $10W$ to $80W$, and lift the area constraint. Correspondingly, we refine the action space for architectural parameters. For example, the number of PEs can be chosen from [256, 1024, 4096, 8192, 32768], and the size of global buffer can be chosen from [128, 256, 512, 1024, 2048]. Fig. 11 shows the EDP results. When scaling the maximum power up to $80W$, Poros achieves 2.65× and 2.37× better EDP on average, relative to HASCO and Spotlight respectively. For a fair comparison, we also refine the architectural design space in HASCO and Spotlight. By exploiting one-level exploration and RL, Poros can faster identify high-quality solutions. This experiment demonstrates Poros’s ability to design both edge-scale and could-scale accelerators.

V. Conclusion

In this paper, we introduce Poros, a one-level architecture-mapping co-exploration framework to automatically design specialized accelerators for tensor algorithms. We model the architecture-mapping co-exploration as a reinforcement learning problem. The agent can perform gradient-based search and handle non-convexity in the joint design space. Experimental results demonstrate that Poros achieves up to 5.32× and 2.40× better EDP compared with hand-designed accelerators and other automatic approaches respectively. Through one-level exploration scheme, Poros converges at least 20% faster than state-of-the-art approaches. In the future, we will extend Poros to support additional architecture primitives such as interconnection patterns between PEs and the burst length of memory access controllers. We also plan to integrate Poros with neural architecture search (NAS).

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[2] J. E. Terán, Y. Marrero-Ponce, E. Contreras-Torres, C. R. García-Jacas, R. Vivas-Reyes, E. Terán, and F. J. Torres, "Tensor algebra-based geometrical (3d) biomacro-molecular descriptors for protein research: Theory, applications and comparison with other methods," *Scientific Reports*, 2019.

[3] S. Fernandes, H. Fanaee-T, and J. Gama, "Tensor decomposition for analysing time-evolving social networks: An overview," *Artificial Intelligence Review*, 2021.

[4] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[5] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[6] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," *arXiv preprint arXiv:1911.09925*, 2019.

[7] "Nvdla deep learning accelerator," http://nvdla.org., 2017.

[8] N. 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 *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[9] Nvidia, "Cudnn," https://developer.nvidia.com/cudnn, 2022.

[10] ——, "Cutlass," https://github.com/NVIDIA/cutlass, 2022.

[11] Intel, "oneapi deep neural network library," https://github.com/oneapisrc/oneDNN.

[12] S. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[13] Q. Xiao, S. Zheng, B. Wu, P. Xu, X. Qian, and Y. Liang, "Hasco: Towards agile hardware and software co-design for tensor computation," in *Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[14] C. Sakhuja, Z. Shi, and C. Lin, "Leveraging domain information for the efficient automated design of deep learning accelerators," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[15] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487–1495.

[16] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. 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 (MICRO)*, 2019.

[17] Y. Chen, T. Yang, J. Emer, and V. 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*, 2019.

[18] J. Weng, S. Liu, V. Dadu, Z. Wang, P. Shah, and T. Nowatzki, "Dsagen: Synthesizing programmable spatial accelerators," in *Proceedings of the 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

[19] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[20] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen *et al.*, "Ansor: Generating {High-Performance} tensor programs for deep learning," in *Proceedings of the 14th USENIX symposium on operating systems design and implementation (OSDI)*, 2020.

[21] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, "Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[22] S. Feng, B. Hou, H. Jin, W. Lin, J. Shao, R. Lai, Z. Ye, L. Zheng, C. H. Yu, Y. Yu *et al.*, "Tensorir: An abstraction for automatic tensorized program optimization," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[23] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, 1983.

[24] Q. Huang, A. Kalaiah, M. Kang, J. Demmel, G. Dinh, J. Wawrzynek, T. Norell, and Y. S. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[25] K. Hegde, P. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[27] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[28] 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 *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[29] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[30] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, Y. Zhang, B. Zimmer, W. J. Dally, J. Emer, S. W. Keckler, and B. Khailany, "Magnet: A modular accelerator generator for neural networks," in *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[31] L. Jia, Z. Luo, L. Lu, and Y. Liang, "Tensorlib: A spatial accelerator generation framework for tensor algebra," in *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[32] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," 1999.

[33] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of the International Conference on Medical image computing and computer-assisted intervention (MICCAI)*. Springer, 2015.

[34] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.

[35] J. D. M.-W. C. Kenton and L. K. 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 (NAACL)*, 2019.