An Imitation Augmented Reinforcement Learning Framework for CGRA Design Space Exploration

Liangji Wu, Shuaibo Huang, Ziqi Wang, Shiyang Wu, Yang Chen, Hao Yan, Longxing Shi
{joewu, huangshuaibo, wangzq98, wu_sy, scottchen, yanhao, lxshi}@seu.edu.cn
Southeast University, Nanjing, China

***Abstract*—Coarse-Grained Reconfigurable Arrays (CGRAs) are a promising architecture that warrants thorough design space exploration (DSE). However, traditional DSE methods for CGRAs often get trapped in local optima due to singularities, *i.e.*, invalid design points caused by CGRA mapping failures. In this paper, we propose a singularity-aware framework based on the integration of reinforcement learning (RL) and imitation learning (IL) for DSE of CGRAs. Our approach learns from both valid and invalid points, substantially reducing the probability of sampling singularities and accelerating the escape from inefficient regions, ultimately achieving high-quality Pareto points. Experimental results demonstrate that our framework improves the hypervolume (HV) of the Pareto front by 23.56% compared to state-of-the-art methods, with a comparable time overhead.**

***Index Terms*—CGRA, reinforcement learning, imitation learning, design space exploration.**

I. Introduction

Coarse-Grained Reconfigurable Arrays (CGRAs) offer a compelling accelerator architecture solution for compute-intensive applications due to their reconfigurable nature. Applications are compiled into data flow graphs (DFGs) and executed spatially and temporally on CGRAs, enabling performance gains and power efficiency improvement [1], [2]. However, the design of CGRAs incorporates numerous processing elements (PEs), memory units, and intricate interconnects, resulting in an expansive design space. This vast array of possible configurations leads to a challenging trade-off between performance, power, and area (PPA) metrics. Design space exploration (DSE) methods for CGRAs typically leverage exhaustive search techniques to explore the various parameter combinations, aiming to find the optimal configuration for a given workload.

Furthermore, DSE for CGRAs presents a distinctive challenge compared to typical CPU scenarios. A critical step in the CGRA framework is the mapping process, which is not always successful [3]. The mapping success rate depends on a complex interplay of factors, including resource constraints, routing bottlenecks, and the efficiency of the mapping algorithm [4]. Mapping failures prevent applications from running on the CGRA, leading to invalid points in the design space, known as *singularities*, which occupy a certain portion of the design space, as shown in Fig. 1. These points are characterized by unavailable PPA values, further complicating the already intricate DSE process.

Conventional DSE methods typically follow a three-step process: modeling, resampling, and optimization [5]. In the modeling phase, a surrogate model is built based on the initial data set. In the resampling phase, more samples are obtained to refine the surrogate model iteratively. In the optimization phase, the optimal designs with PPA trade-offs are found based on the refined models. For existing methods in academics, Li et al. [6] employ AdaBoost with orthogonal sampling to build a surrogate model. Bai et al. [7] combine cluster sampling with domain-specific knowledge, DKL-GP, and improved Bayesian optimization (BO) to achieve the Pareto front. Wang et al. [8] propose a high-accuracy AdaGBRT modeling method paired with a uniformity-aware selector for resampling.

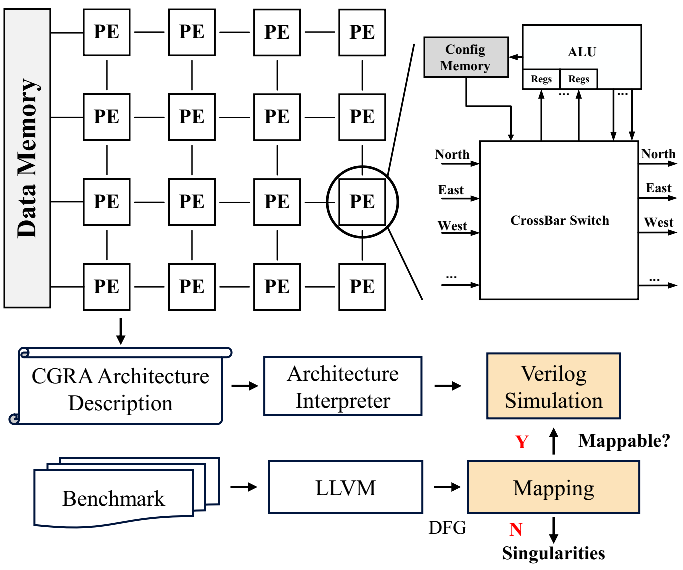

Fig. 1. A typical CGRA framework, where mapping is an essential step prior to performing Verilog simulation.

However, conventional approaches to DSE for CGRAs face limitations due to the presence of singularities in the design space. Singularities cause two major issues for the application of surrogate models: 1) Blindly resampling without knowledge of the distribution of singularities would lead to a considerable number of invalid points with additional simulation overheads. 2) The model accuracy would be sacrificed if all singularities are discarded, which causes a discrepancy between model predictions and the actual design space and degrades its local exploration ability. Consequently, the resampling process may steer exploration toward suboptimal regions, increasing the risk of getting trapped in local optima and affecting the overall quality of the Pareto front.

In this paper, we develop a singularity-aware DSE framework IARL (Imitation Augmented Reinforcement Learning) for CGRAs, which fully leverages the advantages of reinforcement learning (RL) [9–11]. This work presents the following contributions:

- Based on the soft actor-critic (SAC) method, the RL agent can progressively learn to recognize the characteristics of singularity points by assigning reward values, reducing the likelihood of sampling invalid points.
- We integrate imitation learning (IL) into the SAC algorithm, using demonstration to improve RL performance and accelerate convergence.
- We validate the IARL method on the HyCUBE for CGRA microarchitecture. Experimental results demonstrate outstanding performance over conventional approaches. IARL is customized for the features of CGRA, setting it apart from conventional CPU-focused microarchitecture DSE approaches.

II. Preliminaries

A. *CGRA Template*

HyCUBE [12] is an academic CGRA microarchitecture offering highly efficient data routing and execution through its innovative single-cycle multi-hop interconnect. Unlike conventional CGRAs, HyCUBE enables communication between distant functional units (FUs) within a single clock cycle, eliminating multi-hop delays inherent in traditional designs. A key feature of HyCUBE is its dynamic virtual neighborhood configuration, allowing the interconnect to be reprogrammed on a per-cycle basis to ensure optimal data flow. The interconnect operates via a statically scheduled crossbar switch, managed by the compiler, avoiding the complexity and overhead of a network-on-chip (NoC). Multicasting support allows simultaneous data dispatch to multiple FUs, further enhancing performance. The memory hierarchy of HyCUBE includes lightweight distributed registers at each FU input, reducing power and area overhead. In this work, HyCUBE is represented using OpenCGRA [13].

B. *Problem Formulation*

Definition 1 (HyCUBE design space)**.** The HyCUBE design space, based on OpenCGRA, is defined by a 13-dimensional input vector $\boldsymbol{x}$ representing the design parameters used to accelerate specific loop kernels, such as $rows$, $columns$, $configuration\_memory\_size$, $data\_memory\_size$, *etc.*

Definition 2 (SpeedUp)**.** The primary function of CGRAs is loop acceleration, and we evaluate the performance of CGRAs by calculating the SpeedUp achieved in loop acceleration, as shown in the following equation:

$$SpeedUp = \frac{N_{OP}}{II}. \tag{1}$$

Where N_{OP} refers to the number of operations, while II represents the initiation interval, which is the minimum number of clock cycles required before the next iteration of the same loop can begin. A shorter II indicates that new operations can be initiated more frequently.

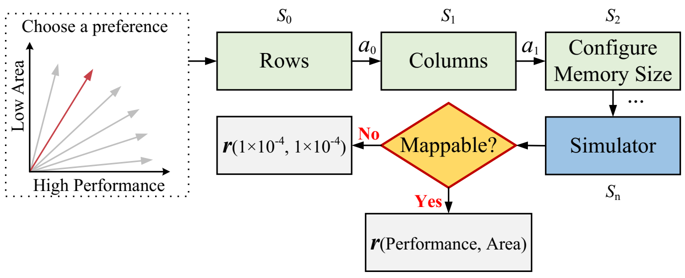

Fig. 2. Preference-guided MOMDP for CGRA DSE.

Definition 3 (Circuit Area)**.** Circuit Area refers to the physical area of the RTL after layout by EDA flows.

Definition 4 (Pareto Front)**.** The Pareto front is a set of non-dominated solutions in a multi-objective optimization problem. A solution $\boldsymbol{x}^*$ is said to be Pareto optimal if there does not exist another solution $\boldsymbol{x}$ such that $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*)$ for all $i \in \{1, \dots, n\}$ and $f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*)$ for at least one j. In this paper, the Pareto front represents the trade-off between area and performance.

Definition 5 (Hypervolume)**.** The hypervolume is a metric used to evaluate the quality of a Pareto front in multi-objective optimization. It represents the volume of the objective space dominated by the solutions on the Pareto front, relative to a reference point. The formulation is defined as:

$$HV(r, \mathcal{X}) = \Lambda \left(\bigcup_{x_i \in \mathcal{X}} [f_1(x_i), r_1] \times \cdots \times [f_m(x_i), r_m] \right), \tag{2}$$

where r refers to a reference point located in a bad performance point. f_m is the value of solution x_i in objective m.

Problem 1 (HyCUBE Design Space Exploration)**.** In the 13-dimensional design space $\mathcal{X}$, each set of CGRA design parameters is represented as a feature vector $\boldsymbol{x}$. The SpeedUp and circuit area define the performance-area space $\mathcal{Y}$, derived from $\boldsymbol{x}$ using the OpenCGRA simulator. The goal of DSE is to find design parameters $\mathcal{X}_p$ on the Pareto frontier $\mathcal{Y}_p$ in $\mathcal{Y}$.

This problem is modeled as a multi-objective Markov decision process (MOMDP), represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega \rangle$, where $\mathcal{S}$ is the state space, and each state $s \in \mathcal{S}$ represents a step in selecting design parameters. The terminal state yields the complete design vector $\boldsymbol{x}$. The action space $\mathcal{A}$ represents the set of possible values for design parameters. $\mathcal{R}$ is the reward space, representing all min-max normalized performance and area values as vectors $\boldsymbol{r} \in \mathcal{R}$. The preference space Ω consists of a set of design preferences, each $\boldsymbol{\omega} \in \Omega$ mapped to a specific reward vector $\boldsymbol{r}$. The utility function $f_\Omega(\boldsymbol{r}) = \boldsymbol{\omega}^\top \boldsymbol{r}$ is used to maximize the cumulative utility, *i.e.*, find optimal design parameters in the MOMDP.

The whole flow of the construction of MOMDP is shown in Fig. 2. A complete MOMDP is the sequential decision-

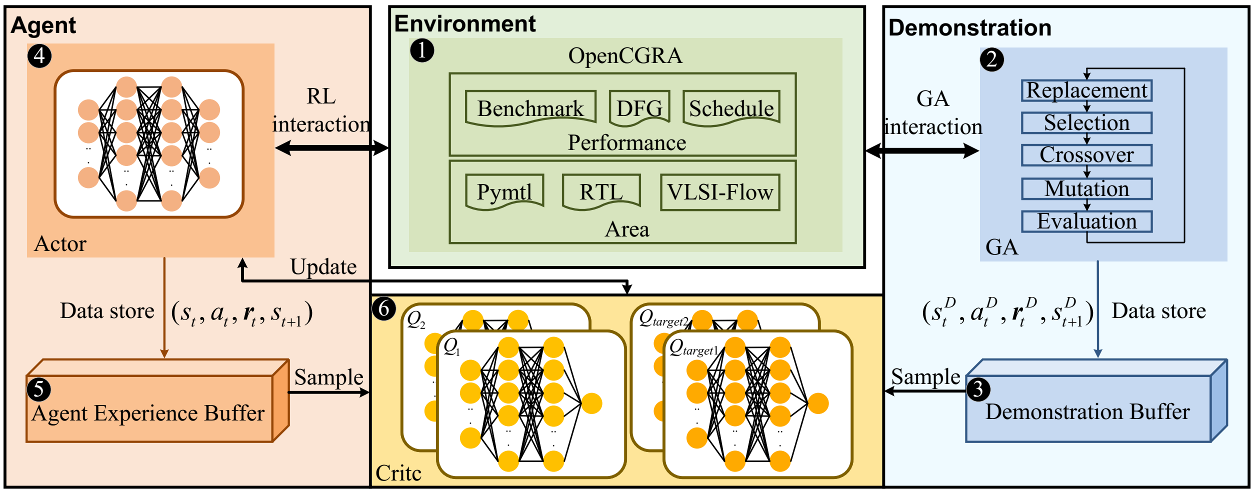

Fig. 3. An overview of IARL framework.

making process for determining each CGRA design parameter under a given set of preferences. At the end of each episode, the environment assigns reward to the agent based on the performance and area when the mapping performed by the simulator is successful. In the case of a mapping failure, the environment provides the agent with a negligible reward vector of 1×10^{-4} to prevent reward sparsity. Reward sparsity hinders the convergence of RL by weakening gradient signals and impairing the efficiency of policy updates.

III. Methodology

In this section, we present the details of the IARL framework, which is enhanced with IL. By leveraging pre-collected design points with relatively high rewards, referred to as demonstrations, IL accelerates the convergence of RL and improves the final results. These demonstrations are generated via a genetic algorithm (GA), with further details provided in Section III-C. Ultimately, we construct the Pareto front based on the design points collected during RL exploration.

A. Overview of IARL Framework

Fig. 3 demonstrates our IARL framework. ❶ refers to the environment where the agent interacts, composed of OpenCGRA and the VLSI flow based on it, providing a two-dimensional reward vector that includes both performance and area. ❷ is a GA that interacts with the environment before the start of RL, obtaining a set of design points with relatively high performance and area metrics. These points are treated as demonstrations, and their design parameters are converted into episodes. Each episode consists of multiple Markov state-action-reward-next-state transition tuples $(s_t^D, a_t^D, \boldsymbol{r}_t^D, s_{t+1}^D)$, where D represents the demonstration. Then it is stored in the demonstration buffer $\mathcal{B}^D$ in ❸.

When RL begins, the actor (policy) network ϕ in ❹ is responsible for interacting with the environment and obtaining a series of Markov transition tuples $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$, which are stored in the agent experience buffer $\mathcal{B}^A$ in ❺. Subsequently, the agent dynamically samples Markov transition tuples from both ❸ and ❺, which are fed into the Q-network pairs θ_1, θ_2, as well as target networks $\bar{\theta}_1$, $\bar{\theta}_2$ in ❻. The agent selects the smaller value from the Q-network pair to update the ❹. The actor is optimized by maximizing the expected Q-value, incorporating an entropy term to encourage exploration and minimizing the IL loss. The agent gradually improves its reward and converges by learning autonomously from either the environment or demonstrations. It is worth noting that since multi-objective optimization is required, the agent is trained separately for each preference $\boldsymbol{\omega} \in \Omega$.

B. Multi-Objective Soft Actor-Critic

SAC is chosen for its ability to maximize both expected reward and policy entropy, encouraging exploration and reducing the risk of converging to local optima. Its off-policy nature allows for efficient sample reuse, improving data efficiency compared to on-policy methods, and integrated well with IL. The entropy-augmented objective also promotes action diversity, contributing to greater robustness and stability during training.

The loss function of the Q-network is updated using the following formula:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{B}}\left[\frac{1}{2}\left(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right], \tag{3}$$

where

$$Q_\theta(s_t,a_t) = \boldsymbol{\omega}^\top \boldsymbol{Q}_\theta(s_t,a_t). \tag{4}$$

It represents an approximation of the expected return for a state-action pair (s_t, a_t) under the current policy.

$$\hat{Q}(s_t,a_t) = \boldsymbol{\omega}^\top \boldsymbol{r}(s_t,a_t) + \gamma \mathbb{E}_{s_{t+1}\sim p}\left[V_{\bar{\theta}}(s_{t+1})\right]. \tag{5}$$

This is the target Q-value, which is an estimate of the true Q-value calculated based on future reward and a target network. The two-dimensional reward $\boldsymbol{r}(\text{Performance}, \text{Area})$ is obtained from OpenCGRA and normalized follows, respectively: $\text{Normalized}_{SpeedUp} = \frac{SpeedUp - \min(SpeedUp)}{\max(SpeedUp) - \min(SpeedUp)}$,

Algorithm 1: IARL for exploration.

```
Input : Initial demonstration replay buffer $\mathcal{B}^D$ generated by
        GA, Initial sampling ratio $\rho$, preference vector set $\Omega$,
        empty evaluated points set $L$, total step $T$.
1  for $\boldsymbol{\omega}_i \in \Omega$ do
2      Initialize Q network $\theta_{i1}$, $\theta_{i2}$, target Q network $\bar{\theta}_{i1} \leftarrow \theta_{i1}$,
        $\bar{\theta}_{i2} \leftarrow \theta_{i2}$ in ❻, policy network $\phi_i$ in ❹;
3      Initialize empty agent experience buffer $\mathcal{B}_i^A$ in ❺,
        demonstration replay buffer $\mathcal{B}_i^D$ in ❸ from $\mathcal{B}^D$;
4      while step $< T$ do
5          Observe state $s_t$ and sample action from the policy
            $a_t \sim \pi_{\phi_i}(a_t \mid s_t)$;
6          Take action $a_t$ to interact with the environment ❶;
7          Observe $\boldsymbol{r_t}$, next state $s_{t+1}$ from the environment;
8          Store the transition $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$ in the agent
            replay buffer $\mathcal{B}_i^A$;
9          if time to update then
10             Sample a mini-batch $\mathcal{M}$ from $\mathcal{B}_i^A$ and $\mathcal{B}_i^D$ based
                on $\rho$;
11             Update Q network $\theta_{i1}$, $\theta_{i2}$ using Eq. (3), Eq. (4),
                Eq. (5) and Eq. (6);
12             Update policy network using Eq. (8);
13             Update entropy parameter $\alpha$;
14             Update target Q network $\bar{\theta}_{i1}$, $\bar{\theta}_{i2}$;
15             Update priorities of transition tuples $p_i$ using
                Eq. (12);
16         end
17         if $s_{t+1}$ is the completion of a single design then
18             Update $\rho$ according to Eq. (11).;
19             Transform $s_t$ into design feature vector $\boldsymbol{x}$,
                transform $\boldsymbol{r}_t$ into performance-power vector $\boldsymbol{y}$;
20             $L \leftarrow L \cup \{\boldsymbol{x}, \boldsymbol{y}\}$;
21         end
22     end
23 end
24 return Pareto set $\mathcal{X}_p$ from $L$
```

$\text{Normalized}_{1/Area} = \frac{1/Area - \min(1/Area)}{\max(1/Area) - \min(1/Area)}$, where the maximum and minimum values are preset constants. $V_{\bar{\theta}}(s_{t+1})$ represents the value function for the next state s_{t+1} given by the following loss function:

$$V_{\bar{\theta}}(s_{t+1}) = \mathbb{E}_{a_{t+1}\sim\pi}[\min_{i=1,2} Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1} \mid s_{t+1})]. \tag{6}$$

The loss function of the policy network in SAC is typically defined as:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{B}} \left[\mathbb{E}_{a_t \sim \pi_\phi} \left[\alpha \log \pi_\phi(a_t \mid s_t) - Q_\theta(s_t, a_t)\right]\right], \tag{7}$$

where α is the temperature parameter that determines the trade-off between exploration and exploitation that is automatically tuned for the training.

C. SAC with Imitation Learning in Optimization

To alleviate the slow convergence by the blind application of RL without prior knowledge, we introduce the concept of IL. This accelerates the agent exploration process in the early stages by learning from demonstrations, significantly boosting the convergence of RL. Additionally, the agent can leverage the knowledge acquired from the demonstrations to reduce random exploration in regions with a large number of singularities. To prevent the agent from merely imitating demonstration actions, we apply dynamic prioritized experience replay (DPER) to strike a balance between imitating demonstration actions and autonomous exploration. Most of the demonstration data is obtained through GA. Singularities can disrupt the fitness function of GA, leading to reduced population diversity and premature convergence. However, since the GA does not rely on surrogate models, it avoids being misled from surrogate models in the optimization direction. Despite the potential for suboptimal solutions, these solutions still cover a diverse range of viable strategies within the design space. Therefore, the actions it generates can provide valuable demonstrations for the agent, serving as a foundation for RL.

We further improve the SAC algorithm by incorporating IL into the policy loss function. Specifically, we modify the loss function in Eq. (7) with a new loss function J that includes a regularization term, as shown in the following equation:

$$J(\phi) = \begin{cases} J_\pi^{RL}(\phi), & \text{if } s_t \sim \mathcal{B}^A, \\ J_\pi^{RL}(\phi) + \lambda J_\pi^{IL}(\phi), & \text{if } s_t \sim \mathcal{B}^D. \end{cases} \tag{8}$$

When the agent exploration Q-value surpasses that of the demonstration, the policy network will stop updating using imitation loss, which is shown as follows:

$$\min_{i=1,2} Q_{\theta_i}(s_t^D, a_t) \geq Q_\theta(s_t^D, a_t^D), \quad a_t \sim \pi_\phi(\cdot|s_t^D), \tag{9}$$

The loss function of IL is shown as follows:

$$J_\pi^{IL}(\phi) = -\log \pi_\phi\left(a_t^D \mid s_t\right). \tag{10}$$

It is a cross-entropy loss function used to measure the difference between the predicted action probabilities and the demonstration actions a_t^D during an episode. λ is the coefficient of the IL loss function. By minimizing this loss, the policy π_θ is trained to behave similarly to the demonstration, effectively learning to imitate the actions from the demonstration.

DPER is key to balance the agent learning from demonstrations and autonomous exploration. The agent samples from the $\mathcal{B}^D$ with probability $1-\rho$, and from the $\mathcal{B}^A$ with probability ρ. $\rho \in [0, 1]$, which is updated after an episode is done according to:

$$\rho = \rho + \frac{1}{mN_\mathcal{M}}, \text{if } \boldsymbol{\omega}^\top \boldsymbol{r}_A \geq \boldsymbol{\omega}^\top \bar{\boldsymbol{r}}_D. \tag{11}$$

Where $mN_\mathcal{M}$ represents m-times the size of the mini-batch, $\boldsymbol{\omega}^\top \boldsymbol{r}_A$ denotes the utility value derived from the multiplication of the current episodic reward vector $\boldsymbol{r}_A$ obtained by the agent under the preference vector $\boldsymbol{\omega}^\top$, and $\bar{\boldsymbol{r}}_D$ represents the average episodic reward of the demonstration. It implies that if the rewards obtained by the agent from trajectories explored autonomously in the environment exceed the average reward of the demonstrations, the probability of the agent sampling directly from the environment is increased. We define the probability of sampling a transition tuple from $\mathcal{B}^D$ and $\mathcal{B}^A$ as $P(i) = \frac{p_i^\eta}{\sum_k p_k^\eta}$, η is a hyperparameter, where

$$p_i = \begin{cases} J_\pi^{RL}(\phi) + J_Q(\theta), & \text{if in } \mathcal{B}_A, \\ J_\pi^{IL}(\phi) + J_Q(\theta), & \text{if in } \mathcal{B}_D. \end{cases} \tag{12}$$

Importance-sampling weight is used to correct the bias caused by non-uniform sampling during backpropagation.

For a more specific illustration of our end-to-end flow of IARL, the pseudo-code implementation is given in Alg. 1.

IV. Experiments

A. Experimental Setup

The proposed method is validated on a personal server with a 12-core Intel core i7 12700K CPU, an NVIDIA RTX 4060 GPU, and 128GB of main memory, operating under Ubuntu 22.04 LTS. Simulation results are derived from a lightweight CGRA DSE environment, which will be elaborated in Section IV-B. Our framework is implemented using Python3.7. The input design parameters for both networks are normalized using one-hot encoding. Given that the range of possible values for different design parameters varies, the action space for the agent is consequently state-dependent. To address this, we employ action masking to filter out invalid actions from the policy network output.

The initial coefficient ρ in Eq. (11) is set as 0.3, λ in Eq. (8) is set as 1.0, initial entropy weight α is set as 1.0 and the discount factor γ is 0.99. The Adam optimizer is used, and the learning rate of policy network is set as 0.0003. A series of representative application for loop acceleration are selected as benchmark, including FFT, SPMV, Conv2D, Conv3D, ReLU, GEMM, SUSAN, MVT, Pooling, Compress, Aggregate. We calculate the average SpeedUp of all applications.

To ensure diversity in the Pareto set, we define different performance-to-area preferences for the RL: (0.1, 0.9) for area-critical designs, (0.5, 0.5) for balanced trade-offs, and (0.9, 0.1) for performance-critical designs. The preferences reflect whether the design in the target scenario prioritizes performance or area. We collect demonstration data using MOEA/D [14] by sampling 1200 valid points. MOEA/D is used to find design points with relatively high weighted reward values corresponding to each preference, the demonstration experience buffer contains 40 diverse episodes.

B. Environment of Reinforcement Learning for CGRA

Directly obtaining PPA through a VLSI flow is a highly time-consuming and complex process. To address this, we employ OpenCGRA to obtain performance and area metrics, with performance specifically referring to SpeedUp. To enhance the efficiency of agent-environment interactions, we consider establishing a lightweight environment using supervised learning models. Experimental results indicate that supervised learning models, such as AdaBoost [15], exhibited significant modeling inaccuracies for SpeedUp, with considerable errors and issues related to application mapping failures to CGRA. However, the area modeling accuracy is more acceptable, with an R^2

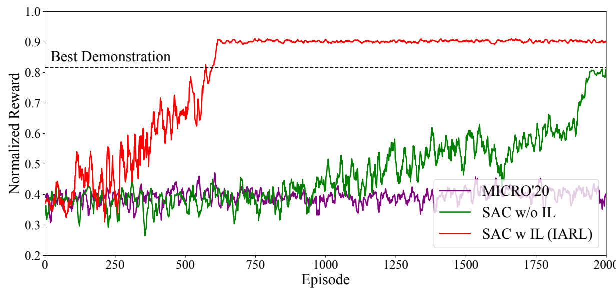

Fig. 4. Training curves of different RL methods given prefer (0.5, 0.5).

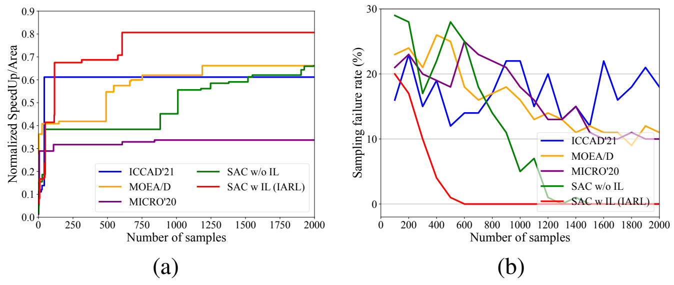

Fig. 5. (a) Convergence of normalized performance-area ratio for different methods in 2000 samples. (b) The sampling failure probability per 100 samples in 2000 samples.

value reaching 0.92. Given that a single SpeedUp simulation takes less than 15 minutes, whereas area analysis takes over six hours, we recommend modeling area while directly obtaining SpeedUp from OpenCGRA.

C. Comparison and Analysis

To comprehensively evaluate our proposed IARL approach, we compare it with several previous DSE methods and commonly used RL methods, *i.e.*, ICCAD'21 employing BO [7], SAC [9] with PER[16], MICRO'20 employing REINFORCE [17] and MOEA/D [14]. For ICCAD'21, we carefully select an set of design points as the initial dataset. For MICRO'20, we omit the GA component and retain its core REINFORCE for a more focused comparison.

We compare the training results of each RL method over 2000 episodes with a preference of (0.5, 0.5), as shown in Fig. 4. The black dashed line is the best demonstration, representing the best scalarized reward value, *i.e.*, the best design point obtained by MOEA/D under this preference. The results show that IARL achieves significant improvements over SAC by not only accelerating convergence but also enabling agents to uncover new high-reward design points through autonomous exploration of the environment, ultimately surpassing the demonstrations.

We also compare the convergence of the RL method under a specific preference of (0.5, 0.5) with those of other DSE methods, as illustrated in Fig. 5a. It presents the convergence plots showing the ratio between maximum normalized SpeedUp and area for different methods, reflecting how each method

TABLE I: Comparison of hypervolume for different methods in 1500 and 3000 samples.

Related Works	1500 samples	3000 samples
ICCAD'21	0.6822	0.6827
MOEA/D	0.6985	0.7263
MICRO'20	0.3630	0.3933
SAC	0.4658	0.5361
Proposed	0.7837	0.8435

progressively improves design parameters to offer designs with higher computational performance per unit area. While ICCAD'21 and MOEA/D perform global searches, the RL method conducts a local search based on the given preference. Although our proposed method is a local search approach given a specific preference (0.5, 0.5), it has surpassed the global search methods ICCAD'21 and MOEA/D.

The results show that BO-based method ICCAD'21 converges radpidly but has suboptimal convergence quality due to the presence of singularities, which introduce bias into the surrogate model, subsequently impairing the effectiveness of the acquisition function and ultimately leading to suboptimal solutions. MOEA/D performs well, but the singularities disrupt population diversity, limiting search efficiency. The REINFORCE-based method MICRO'20 has a significant difficulty in convergence due to high variance in gradient estimates, resulting in low efficiency in policy improvement. Although SAC mitigates the slow convergence issues prevalent in RL to some extent, it still requires a substantial number of samples to converge, owing to the need to balance exploration and exploitation. In contrast, the proposed IARL framework accelerates convergence significantly in the early stages by allowing the agent to imitate demonstrations, while encouraging autonomous exploration in later stages. This approach ensures both faster convergence and high solution quality.

Subsequently, we compare the trends in the mapping failure rates of sampled design points across different methods, where a design is considered a failure if it fails to map in even one application. Specifically, for each method, we sample 2000 design points and calculate the mapping failure rate for every 100 points, which is presented in Fig. 5b. The results show a significant decrease in the mapping failure rate for the samples selected by our method, whereas ICCAD'21 exhibits little change. This indicates that the agent is gradually learning to avoid invalid points in the space.

We compare the Pareto front obtained by different methods, evaluating the Pareto fronts generated by sampling 1500 and 3000 points for all methods. For a fair comparison, in the case of RL methods, we sample points under three different preferences described in Section IV-A. Specifically, at 1500 points, each RL method explores 500 points under each of the three preferences, which is considered equivalent to a global search over 1500 points for the resulting Pareto front. Similarly, for 3000 points, 1000 points are sampled under each preference. While RL methods primarily explore solutions under specific preferences, they still exhibit a degree of global exploration, driven by mechanisms such as randomness and soft policy updates that encourage agents to attempt diverse actions. This makes it feasible to compare them with global search methods such as ICCAD'21 and MOEA/D, which directly search for the Pareto front.

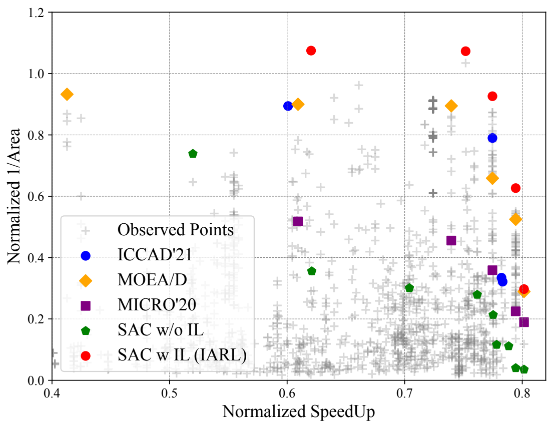

Fig. 6. The visualizations of selected Pareto points in performance-area design space in 3000 samples, the closer to the upper-right corner, the better.

Table I presents the hypervolume (HV) achieved by each method after sampling 1500 and 3000 points. IARL outperforms ICCAD'21, MOEA/D, MICRO'20, SAC by 23.56%, 16.11%, 114.50%, 57.34% in HV of 3000 points, respectively. Experimental results show that our proposed method significantly accelerates the convergence speed and achieves the highest-quality Pareto points after sampling 3000 points. Fig. 6 presents a visualization of the Pareto fronts generated by sampling 3000 points using the method in Table I.

V. Conclusion

In this paper, we present a novel RL framework IARL that significantly enhances the efficiency of DSE for CGRAs. By integrating RL with IL, our method effectively navigates singularities, which have traditionally posed major challenges in CGRA optimization. We develop the demonstration mechanism to accelerate convergence and lead to higher-quality Pareto-optimal solutions with minimal additional simulation overhead. The experimental results on the HyCUBE CGRA architecture confirm the superiority and robustness of the IARL framework, with a 23.56% improvement in the hypervolume of the Pareto front, setting a new solution for CGRA DSE.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant (NSFC 62474038, 62274034), and the Natural Science Foundation of Jiangsu Province BK20232005, the Key Research and Development Program of Jiangsu Province under Grant BE2023003-2, the Fundamental Research Funds for the Central Universities.

References

[1] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2016, pp. 235–244.

[2] Z. Li, D. Wijerathne, and T. Mitra, *Coarse-Grained Reconfigurable Array (CGRA)*. Singapore: Springer Nature Singapore, 2022, pp. 1–41. [Online]. Available: https://doi.org/10.1007/978-981-15-6401-7_50-1

[3] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 184–189.

[4] Z. Zhao, W. Sheng, Q. Wang, W. Yin, P. Ye, J. Li, and Z. Mao, "Towards Higher Performance and Robust Compilation for CGRA Modulo Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2201–2219, 2020.

[5] S. Huang, Y. Ye, H. Yan, and L. Shi, "ARS-Flow: A Design Space Exploration Flow for Accelerator-rich System based on Active Learning," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024, pp. 213–218.

[6] D. Li, S. Yao, Y.-H. Liu, S. Wang, and X.-H. Sun, "Efficient design space exploration via statistical sampling and AdaBoost learning," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[7] 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 *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[8] D. Wang, M. Yan, X. Liu, M. Zou, T. Liu, W. Li, X. Ye, and D. Fan, "A High-accurate Multi-objective Exploration Framework for Design Space of CPU," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," 2019. [Online]. Available: https://arxiv.org/abs/1812.05905

[10] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. M. O. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards," *ArXiv*, vol. abs/1707.08817, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:23192910

[11] S. Reddy, A. D. Dragan, and S. Levine, "{SQIL}: Imitation Learning via Reinforcement Learning with Sparse Rewards," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1xKd24twB

[12] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[13] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "OpenCGRA: An Open-Source Unified Framework for Modeling, Testing, and Evaluating CGRAs," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 381–388.

[14] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[15] D. Solomatine and D. Shrestha, "AdaBoost.RT: a boosting algorithm for regression problems," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 2, 2004, pp. 1163–1168 vol.2.

[16] T. Schaul, "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, 2015.

[17] S.-C. Kao, G. Jeong, and T. Krishna, "ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 622–636.