

A3C-S: Automated Agent Accelerator Co-Search towards Efficient Deep Reinforcement Learning

Yonggan Fu
Rice University
yf22@rice.edu

Yongan Zhang
Rice University
yz87@rice.edu

Chaojian Li
Rice University
cl114@rice.edu

Zhongzhi Yu
Rice University
zy42@rice.edu

Yingyan Lin
Rice University
yingyan.lin@rice.edu

Abstract—Driven by the explosive interest in applying deep reinforcement learning (DRL) agents to numerous real-time control and decision-making applications, there has been a growing demand to deploy DRL agents to empower daily-life intelligent devices, while the prohibitive complexity of DRL stands at odds with limited on-device resources. In this work, we propose an Automated Agent Accelerator Co-Search (A3C-S) framework, which to our best knowledge is the first to automatically co-search the optimally matched DRL agents and accelerators that maximize both test scores and hardware efficiency. Extensive experiments consistently validate the superiority of our A3C-S over state-of-the-art techniques.

Index Terms—Network Accelerator Co-design, Deep Reinforcement Learning, AutoML

I. INTRODUCTION

Recent successes in deep reinforcement learning (DRL) [1], which integrates reinforcement learning (RL) and deep neural networks (DNNs), have triggered tremendous enthusiasm in developing and deploying DRL-powered intelligence into numerous inference and control applications, including robotics and autonomous vehicles. Many of them, such as autonomous vehicles, require real-time control and decision-making policies for which the DRL agents have to derive real-time policies using real-time data for dynamic systems. However, real-time control and decision-making for DRL can be prohibitively challenging in many real-world applications due to DRL’s integrated complex DNNs and edge devices’ constrained resources, calling for DRL-powered intelligent solutions that favor both test scores and hardware efficiency.

To address the large gap between the growing need for on-device DRL and DRL’s prohibitive complexity, emerging network and accelerator co-exploration (NACoS) methods [2]–[4] are promising as they can boost DNN acceleration efficiency. However, directly applying NACoS methods to design DRL agents can easily fail due to the commonly observed vulnerability and instability of DRL training, which occur with a high variance [5]. Furthermore, such instability will be further exacerbated when considering differentiable NAS (DNAS) based NACoS methods, which requires competitively low search cost and thus can enable efficient navigation over the large DRL agent and accelerator joint space, because the success of DNAS requires unbiased gradient estimation with low variance. To this end, we aim to develop a novel NACoS framework dedicated to DRL agent acceleration to promote

fast development and highly efficient DRL-powered solutions. Specifically, we make the following contributions:

- We propose an Automated Agent Accelerator Co-Search (A3C-S) framework, which to our best knowledge is the first to automatically co-search the optimally matched DRL agents and accelerators that maximize both test scores and hardware efficiency.
- A3C-S integrates and demonstrates the first DNAS search dedicated to DRL that features a novel distillation mechanism to effectively stabilize agent search, despite the instability of DRL training.
- A3C-S incorporates a parameterized micro-architecture with over 10^{27} searchable choices of accelerators and dataflows to enable a differentiable search for DRL agent accelerators. A3C-S is generic and can be applied on top of different accelerator templates.
- Through FPGA measurements, extensive experiments and ablation studies validate A3C-S’s effectiveness in generating efficient DRL agents/accelerators that consistently outperform state-of-the-art (SOTA) agents/accelerators.

II. RELATED WORKS

Deep reinforcement learning. DRL integrates traditional RL algorithms with DNNs to handle higher-dimensional and more complex problems, e.g., DQN [1] introduces DNNs to Q-Learning and [6] utilizes DNNs to model both the actor and critic in AC-based DRL. More DRL works can be found in [7]. Despite DRL’s promising success, automating the design of DRL agents has not yet been explored, while there is a growing need for fast development of DRL-powered solutions.

Hardware-aware neural architecture search. NAS has been widely adopted to automate the design of efficient DNNs. To tackle the prohibitive search cost of previous RL-based NAS methods, DNAS [8], [9] has gained more attention thanks to its excellent search efficiency. However, the possibility of applying NAS or DNAS to DRL hasn’t been explored.

DNN accelerator design. SOTA DNN accelerators [10], [11] tackle the prohibitive complexity of DNNs through novel micro-architectures/dataflows to maximize data reuses, and thus improve acceleration efficiency. Early works require experts’ manual design, and thus were time-consuming. Later, various design flow and automation tools [12], [13] were developed. However, they mostly explored DNN acceleration, leaving automated DRL accelerator design unexplored.

Network and accelerator co-search. Jointly exploring DNNs and their accelerators is very promising towards efficient DNN solutions, as shown in pioneering works [2], [3], where the former suffers from large search time and the latter considers a limited search space. These works motivate us to explore the joint search for DRL agents and their accelerators to maximize the test scores and hardware efficiency.

III. PRELIMINARIES OF DRL

Here we describe the preliminaries of DRL. RL can be viewed as a Markov Decision Process (MDP) determined by a tuple (S, A, T, R, γ) , where S is the state space, A is the action space, $T(s'|s, a)$ is the transition probability of ending up in state s' when executing the action a in the state s , R is the reward function, and γ is a discount factor. A policy $\pi(s_t, a_t) = p(a_t|s_t)$ defines the probability that an agent in the MDP executes the action a in the state s . In particular, the agent performs the action $a_t \in A$ sampled from the policy $\pi(s_t, a_t)$ at each time step t and the state $s_t \in S$, leading to the next state $s_{t+1} \in S$ and acquiring a reward r_t .

In DRL, a policy is parameterized by θ_π , i.e., the weights of a DNN, and the agent's goal is learning an optimal policy to maximize the expected cumulative reward:

$$\theta_\pi^* = \arg \max_{\theta_\pi} J(\pi(\cdot|\theta_\pi)) = \arg \max_{\theta_\pi} \mathbb{E}_\pi \left[\sum_{t=0}^H \gamma^t r_t \right] \quad (1)$$

where H is the time horizon, and γ is a discount factor.

To solve the optimization problem in Eq. 1, stochastic policy gradient methods [14] are widely adopted where $\nabla_{\theta_\pi} J(\pi(\cdot|\theta_\pi))$:

$$\nabla_{\theta_\pi} J(\pi(\cdot|\theta_\pi)) = \mathbb{E}_\pi \left[\sum_{t=0}^H \delta_t \nabla_{\theta_\pi} \log(\pi(a_t, s_t|\theta_\pi)) \right] \quad (2)$$

In this work, we adopt the temporal difference error (td-error) $\delta_t = r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$ to reduce the variance in policy gradients, where $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^H \gamma^t r_t | s_0 = s \right]$ is the value function which estimates the expected cumulative reward of the policy π starting from the state s . Since the value function V_π is hard to estimate, AC-based DRL methods [15] parameterize the value function (i.e., the critic) with the learnable parameter θ_v , i.e. the weights of a DNN in DRL. The objective of θ_v is to minimize the td-error of the estimated value between consecutive states:

$$\theta_v^* = \arg \min_{\theta_v} \mathbb{E}_\pi \left[\sum_{t=0}^H \frac{1}{2} (r_t + \gamma V_\pi(s_{t+1}|\theta_v) - V_\pi(s|\theta_v))^2 \right] \quad (3)$$

Therefore, in the AC framework, the actor and critic parameterized by θ_π and θ_v , respectively can be iteratively updated to lead the agent towards an optimal policy.

IV. THE PROPOSED A3C-S FRAMEWORK

There exist three main challenges in designing A3C-S: (1) the huge joint search space, (2) the non-differentiable accelerator parameters and the gap between DNAs's required layer-wise hardware-cost penalty and the optimal accelerators'

dependency on all the layers, and (3) the training instability with a high variance of DRL which may lead to the failure of applying NAS to search DRL agents. In this section, we first introduce the component techniques of A3C-S in Sec. IV-A that tackles the first two challenges with a novel differentiable search strategy and a differentiable accelerator search engine, and then A3C-S's AC-distillation mechanism in Sec. IV-B to tackle the third challenge.

A. A3C-S: the co-search pipeline

A3C-S formulation. We formulate A3C-S as below:

$$\min_{\theta_\pi, \theta_v, \alpha} L_{task}(\theta_\pi, \theta_v, net(\alpha)) + \lambda L_{cost}(hw(\phi^*), net(\alpha)) \quad (4)$$

$$s.t. \quad \phi^* = \arg \min_{\phi} L_{cost}(hw(\phi), net(\alpha)) \quad (5)$$

where α and ϕ are the variables maintaining the probability of choosing different (1) network operators and (2) accelerator parameters, with θ_π and θ_v being the supernet weights of the actor and critic in a AC-based DRL, respectively, L_{task} and L_{cost} are the task loss (see Sec. IV-B for details) and the hardware-cost loss, respectively, and $net(\alpha)$ and $hw(\phi)$ denote the network and accelerator parameterized by α and ϕ , respectively.

A3C-S's co-search pipeline. During co-search, A3C-S starts by updating the accelerator parameters ϕ given the current network structure $net(\alpha)$, and then updates θ_π , θ_v , and α in the same iteration based on the accelerator $hw(\phi^*)$ resulting from the previous step. Our A3C-S adopts one-level optimization [16] instead of bi-level optimization [17], considering that the one-step SGD approximation of bi-level optimization will lead to biased gradient estimation [18] which can largely suffer from the high variance of DRL training evaluated in Sec. V-D. The updates of α and ϕ follow:

$$Forward: A^{l+1} = \sum_{i=1}^N GS_{hard}(\alpha_i^l) O_i(A^l) = O_{fw}^l(A^l) \quad (6)$$

$$Backward: \frac{\partial L_{task}}{\partial \alpha_i^l} = \sum_{k=1}^K \frac{\partial L_{task}}{\partial GS(\alpha_k^l)} \frac{\partial GS(\alpha_k^l)}{\partial \alpha_i^l} \\ = \frac{\partial L_{task}}{\partial A^{l+1}} \sum_{k=1}^K O_k^l(A^l) \frac{\partial GS(\alpha_k^l)}{\partial \alpha_i^l} \quad (7)$$

$$\frac{\partial L_{cost}}{\partial \alpha_i^l} = \mathbb{1}(GS_{hard}(\alpha_i^l) = 1) L_{cost}^{\alpha_i^l}(hw(\phi^*), net(\alpha_i^l)) \quad (8)$$

where A^l and A^{l+1} are the feature maps of the l -th and $(l+1)$ -th layer, respectively, GS_{hard} is the hard Gumbel Softmax operator generating a one-hot output, i.e., only one operator O_{fw}^l will be activated during forward, N is the total number of operator choices, and O_i^l is the i -th operator in the l -th layer parameterized by α_i^l . Meanwhile, GS is a Gumbel Softmax function and K is the number of activated paths with the top K probability, and similar to [19], $K \in (1, N)$ in A3C-S to control the computational cost. In Eq. 8, $\mathbb{1}$ is an indicator denoting whether α_i^l (i.e., the i -th operator in the l -th layer) is activated during forward.

A3C-S’s co-search strategy. A3C-S integrates a novel search strategy to solve Eq. 4 for effective yet efficient search to avoid memory explosion due to the large joint search spaces:

Single-path forward: (see Eq. 6) A3C-S adopts hard Gumbel Softmax sampling [20], i.e., only the choice with the highest probability will be activated to narrow the gap between the supernet and the finally derived network thanks to the single-path property of hard Gumbel Softmax sampling.

Multi-path backward: (see Eq. 7) A3C-S activates multiple paths to approximate the gradients of α via Gumbel Softmax relaxation to balance the search efficiency (prefer fewer activated paths) and stability (prefer more activated paths), inspired by [19] which targets DNAs for DNNs.

Hardware-cost penalty: The network search in Eq. 4 requires layer-wise hardware-cost penalties assuming the layer-wise operators running on the final optimal accelerator $hw(\phi^*)$, which is not yet available at each co-search epoch as the optimal network is still unknown, i.e., the chicken-and-egg problem. To handle this, we approximate the layer-wise hardware-cost by assuming that the single-path network derived during each forward is close to the final derived network, since the network operators that have higher probabilities are also more likely to appear in the final optimal network.

A3C-S’s Differentiable accelerator search (DAS). EDD [3] made a pioneering effort to differentially co-search the network and accelerator, yet their accelerator search space is limited to the parallel factor of their template, which can be analytically fused into their computational cost, whereas this is not always applicable to other naturally non-differentiable accelerator design knobs such as PE numbers and buffer allocation strategies. A more general search engine is desirable.

A3C-S’s accelerator search algorithm: We propose a general DAS engine to efficiently search for the optimal accelerator, including the micro-architectures and dataflows, given a DNN based on the single-path sampling in Eq. 6, i.e.:

$$\phi^* = \arg \min_{\phi} \sum_{m=1}^M GS_{hard}(\phi^m) \hat{L} \quad (9)$$

where $\hat{L} = L_{cost}(hw(\{GS_{hard}(\phi^m)\}), net(\alpha))$

where M is the number of accelerator parameters. Given the network $net(\alpha)$ which is the most likely network sampled during the single-path forward, the search engine utilizes hard Gumbel Softmax GS_{hard} sampling on each design parameter ϕ^m to build an accelerator $hw(\{GS_{hard}(\phi^m)\})$ and penalize each sampled accelerator parameter with the overall hardware-cost L_{cost} through relaxation in a differentiable manner.

A3C-S’s accelerator template: We adopt a parameterized accelerator template built upon a SOTA chunk-based pipeline micro-architecture [21]. The accelerator template comprises multiple sub-accelerators (i.e., chunks) and executes DNNs in a pipeline fashion. In particular, each chunk is assigned with multiple but not necessarily consecutive layers which are executed sequentially within the chunk. Similar to Eyeriss, each chunk consists of levels of buffers/memories (e.g., on-chip buffer and local register files) and processing elements

(PEs) to facilitate data reuses and parallelism with searchable accelerator parameters, including PE interconnections (i.e., Network-on-chip), buffer sizes, and MAC operations’ scheduling and tiling (i.e., dataflows) (see more details in Sec. V-A).

B. A3C-S: the AC-distillation mechanism

Motivation. Policy distillation [22] shows that the distillation from a teacher agent can effectively reduce the variance of gradient estimates and stabilize the training process of the student agent, motivating us to introduce a distillation mechanism to stabilize the DNAs process for DRL. However, vanilla policy distillation merely distills the policy without considering the value function which can play a critical role in both assisting the policy updates and reducing the variance of vanilla policy gradients. We conjecture that further distilling the value function from the teacher agent can better improve the training stability and the convergence.

A3C-S’s AC-distillation. In A3C-S, we propose an AC-distillation mechanism to distill knowledge from both the actor and critic of a pretrained teacher agent to the student agent, where the two distillation losses for the actor and critic are:

$$L_{actor}^{distill} = \mathbb{E}_{\pi} \left[\sum_{t=0}^H \pi(a_t, s_t | \theta_{\pi}^{tea}) \log \frac{\pi(a_t, s_t | \theta_{\pi}^{tea})}{\pi(a_t, s_t | \theta_{\pi}^{stu})} \right] \quad (10)$$

$$L_{critic}^{distill} = \mathbb{E}_{\pi} \left[\sum_{t=0}^H \frac{1}{2} (V_{\pi}(s_t | \theta_v^{stu}) - V_{\pi}(s_t | \theta_v^{tea}))^2 \right] \quad (11)$$

where $\pi(a_t, s_t | \theta_{\pi}^{tea})$ and $\pi(a_t, s_t | \theta_{\pi}^{stu})$ are the teacher and student actor, respectively, and $V_{\pi}(s_t | \theta_v^{tea})$ and $V_{\pi}(s_t | \theta_v^{stu})$ are the teacher and student critic, respectively. We adopt KL divergence to distill the knowledge from the teacher actor following [22] and the MSE loss as a soft constraint to enforce the student critic to mimic the estimated value of the teacher critic. The final objective during both search and training is:

$$L_{task} = L_{policy} + L_{value} + \beta_1 L_{entropy} + \beta_2 L_{actor}^{distill} + \beta_3 L_{critic}^{distill} \quad (12)$$

where β_1 , β_2 , and β_3 are the weighted coefficients. Here L_{policy} is the policy gradient loss as in [14], L_{value} is the value loss based on the td-error, and $L_{entropy}$ is the entropy loss on top of the policy to encourage exploration, i.e.:

$$L_{policy} = \mathbb{E}_{\pi} \left[- \sum_{t=0}^H \delta_t \log(\pi(a_t, s_t | \theta_{\pi}^{stu})) \right] \quad (13)$$

$$L_{value} = \mathbb{E}_{\pi} \left[\sum_{t=0}^H \frac{1}{2} (r_t + \gamma V_{\pi}(s_{t+1} | \theta_v^{stu}) - V_{\pi}(s_t | \theta_v^{stu}))^2 \right] \quad (14)$$

$$L_{entropy} = \mathbb{E}_{\pi} \left[\sum_{t=0}^H \pi(a_t, s_t | \theta_{\pi}^{stu}) \log(\pi(a_t, s_t | \theta_{\pi}^{stu})) \right] \quad (15)$$

The search algorithm of A3C-S is summarized in Alg. 1.

V. EXPERIMENT RESULTS

In this section, we first introduce our experiment setting, and then present ablation studies to evaluate A3C-S’s component techniques and our A3C-S framework.

Algorithm 1 Automated Agent Accelerator Co-Search (A3C-S)

Initialize the step counter $t \leftarrow 1$
repeat
 $t_{start} = t$
 Get state s_t
repeat
 Perform $a_t \sim \pi(a_t, s_t | \theta_{\pi}^{stu})$ based on Eq. 6
 Receive reward r_t and new state s_{t+1}
 $t \leftarrow t + 1$
until terminal s_t or $t - t_{start} ==$ rollout length L
 Update ϕ in Eq. 9 to acquire ϕ^*
for $i \in \{t_{start}, \dots, t - 1\}$ **do**
 $\delta_t = r_t + \gamma V_{\pi}(s_{t+1} | \theta_v^{stu}) - V_{\pi}(s_t | \theta_v^{stu})$
 Calculate L_{task} in Eq. 12 based on $\delta_t, \pi(a_t, s_t | \theta_{\pi}^{tea})$, and $V_{\pi}(s_t | \theta_v^{tea})$
 Calculate L_{cost} in Eq. 4 based on ϕ^*
 Update $\theta_{\pi}^{stu}: \theta_{\pi}^{stu} \leftarrow \theta_{\pi}^{stu} - \eta_1 \nabla_{\theta_{\pi}^{stu}} L_{task}$
 Update $\theta_v^{stu}: \theta_v^{stu} \leftarrow \theta_v^{stu} - \eta_1 \nabla_{\theta_v^{stu}} L_{task}$
 Update $\alpha: \alpha \leftarrow \alpha - \eta_2 \nabla_{\alpha} (L_{task} + \lambda L_{cost})$
end for
until $t > T_{max}$
 Derive the final agent and accelerator with the highest α and ϕ respectively
return the final agent and accelerator

A. Experiment setup

Models and tasks. We evaluate the performance of the AC-based DRL when its feature extractor backbone adopts the searched networks from A3C-S or five SOTA networks with different model sizes, including the original small network in DQN [1] (termed as Vanilla) and ResNet-14/-20/-38/-74, on Atari 2600 games based on the Arcade Learning Environment. For all the ResNets, we modify the stride of the first convolution to be 2 and the output dimension of the final FC layer to be 256 to adapt them to the 84×84 resolution of Atari games.

Training settings. We use the same training and test hyperparameters settings for all the models on all the tasks in this paper. Specifically, we train a DRL agent on each task for $3e7$ steps with a discount factor (γ in Eq. 1) of 0.99 and a rollout length of 5; We use the RMSProp optimizer [1] with an initial learning of $1e-3$ which keeps constant in the first $1e7$ steps and then linearly decays to $1e-4$; and the reported test score is averaged on 30 episodes with null-op starts following [1].

Distillation Settings. For the proposed AC-distillation, we train a ResNet-20 model as the teacher agent and β_1, β_2 , and β_3 in Eq. 12 are set to be $1e-2, 1e-1$, and $1e-3$, respectively, in all the experiments.

Search settings. The supernet structure follows the network

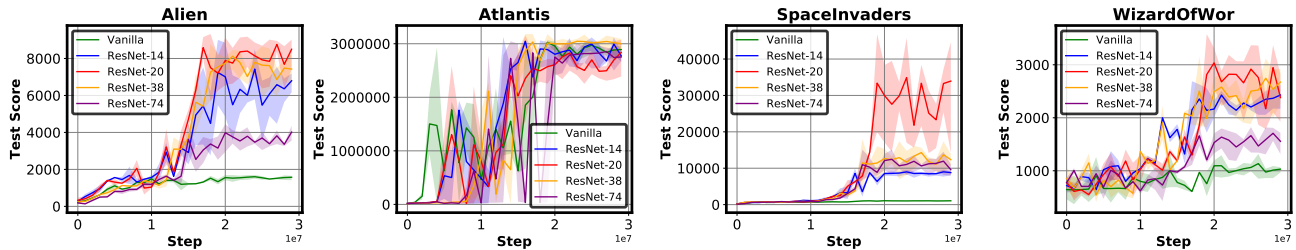


Fig. 1. Test scores averaged over 30 episodes during the training of five models on four Atari games.

TABLE I

THE TEST SCORES OF DIFFERENT MODELS ON A SUBSET OF ATARI GAMES.

Atari Games	Vanilla	ResNet-14	ResNet-20	ResNet-38	ResNet-74
Breakout	523.7	776.5	811	818.5	2.2
Alien	1724	9007	9323	8829	4456
Asterix	4850	708500	856800	756120	539060
Atlantis	3064320	3127390	3156130	3181090	3046490
TimePilot	4780	9070	9680	9500	9040
SpaceInvaders	1171	9848	46870	17962	15111
WizardOfWor	1320	2690	3580	3160	1850
Tennis	-23.7	13.8	11.5	19.6	19.3
Asteroids	2095	5690	5744	1947	4792
Assault	10164	14470	17314	12406.5	9849
BattleZone	7600	5800	13100	13300	4100
BeamRider	5530	23984	25961	29498	30048
Bowling	28.1	53	59.2	33.2	50.8
Boxing	4.2	100	100	99.3	87.1
Centipede	5025	6690	6410	6384.6	6899
ChopperCommand	1320	11170	14910	4370	8240

design (i.e., #groups and stride) of the ResNet series with 12 sequential searchable cells. The candidate operators are standard convolutions with a kernel size 3/5, inverted residual blocks with a kernel size 3/5, a channel expansion of 1/3/5, and skip connections, leading to a search space of 9^{12} choices. We update the architecture parameters using an Adam optimizer with a momentum of 0.9 and a fixed learning rate of $1e-3$. The initial temperature [9] in Gumbel Softmax is set to 5 and decayed by 0.98 every $1e5$ steps.

Accelerator settings. To evaluate A3C-S’s generated accelerators, we adopt a standard FPGA design flow, i.e., the Vivado HLS flow [24], and performance metric, i.e., frame per second (FPS). When searching for the accelerator parameter, A3C-S makes use of a SOTA accelerator performance predictor [13], [25] to obtain fast and reliable estimation during search. The accelerator parameters we optimize upon include **1)** parallel processing elements (PE) settings: the number and inter-connections of PEs, **2)** buffer management: allocation of lower level memories between inputs, weights and outputs, **3)** tiling and scheduling of MAC (Multiply and Accumulate) computations, and **4)** layer allocation: ways to assign each layer to the corresponding pipeline stage (sub-accelerator).

B. Ablation study: DRL with different model sizes

Observations and Analysis. Fig. 1 visualizes the test core evolution during the training process on various Atari games, when adopting different networks for the DRL agents, where the highest achieved test scores is listed in Tab. I. Two observations can be made. **First**, networks with a larger size in general favor the achieved test scores especially on more difficult tasks (e.g., BeamRider of Tab. I), since larger networks can provide higher test scores with the same training time steps in most

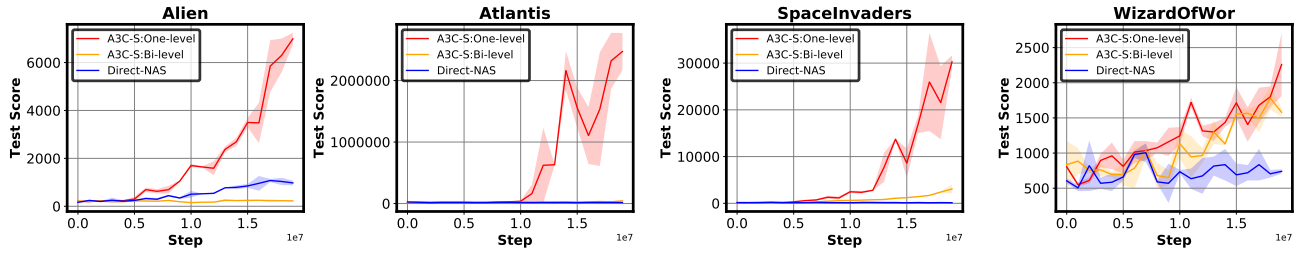


Fig. 2. Test score evolution during the search processes of three different search schemes on four Atari games [23], where Direct-NAS denotes directly applying NAS w/o distillation, and A3C-S:One-level and A3C-S:Bi-level search with the distillation loss using one- and bi-level optimization, respectively.

of the games over the smaller vanilla-network and ResNet-14; Second, there always exists a task-specific optimal network size (i.e., a further increase won't improve or even degrade the test score), which is likely due to the increased difficulty of training larger DRL agents. For instance, the vanilla network performs well in the Atlantis game (see Fig. 1), whereas ResNet-38 merely offers a marginally improved score at a cost of $13.7\times$ higher FLOPs (floating-point operations); and ResNet-74 is inferior to ResNet-20/38 in most of these experiments, since it is more difficult to be trained within the limited training steps. Note that even though more training steps or better-tuned hyper-parameters may improve the convergence of DRL with ResNet-74, the associated inefficiency makes it impractical to be widely adopted.

Extracted Insights. The observations above imply that (1) DNNs' architecture plays a critical role in DRL, which is still under-explored in existing works, and (2) designing task-specific agents is highly desired in optimally balancing the test score and processing efficiency for different tasks. Given that the manual design of dedicated agents for different tasks is not practical in handling the growing need for the fast development of DRL-powered solutions for numerous applications, we are motivated to use NAS to design DRL agents.

C. Ablation study: evaluating the proposed AC-distillation

Observations and Analysis. We compare our AC-distillation mechanism with the three baselines by applying them to the vanilla network and ResNet-14 evaluated on Atari games (see Tab. II). Three observations can be made. First, compared to designs without distillation, distillation strategies in general favor the test scores, which is consistent with [22]; Second, our AC-distillation mechanism consistently performs

best in achieving the highest test scores on most tasks, among the three distillation strategies.

D. Ablation study: one-level vs. bi-level optimization

In this set of experiments, we visualize the test score evolution in Fig. 2 during the search process of (1) directly applying NAS without distillation, (2) using AC-distillation with bi-level optimization, and (3) using AC-distillation with one-level optimization. We can see that the test scores remain low when searching with the bi-level optimization, validating our mentioned hypothesis that the supernet cannot serve as an accurate proxy to indicate the performance of the sampled subnetworks. In contrast, searching with the one-level optimization leads to a consistent improvement in the test scores during search, demonstrating the first framework that successfully makes NAS possible in DRL.

E. Evaluating the proposed A3C-S framework

A3C-S vs. SOTA networks on searched accelerators. Here we evaluate A3C-S's searched agents by comparing their hardware performance (i.e., FPS) with the most competitive SOTA DRL agents, i.e., ResNet-14 with the best trade-off between test scores and hardware efficiency among all our experiments (see Tab. II). For a fair comparison, we use our proposed DAS engine to search for the optimal accelerators under the same search settings and to train both networks with our AC-distillation mechanism. From Fig. 3, we can see that the resulting hardware efficiency of A3C-S's searched agents is consistently higher than that of the most competitive SOTA DRL agents, under a comparable or better test score. This set of experiments again motivates the necessity of applying NAS to search for task-specific optimal agents that balance both the test scores and model complexity.

TABLE II

DRL WITH THE VANILLA NETWORK AND RESNET-14 UNDER: (1) NO DISTILLATION, (2) ONLY POLICY DISTILLATION, AND (3) OUR AC-DISTILLATION.

Atari Games	Vanilla			ResNet-14		
	No distillation	Policy distillation only	AC-distillation	No distillation	Policy distillation only	AC-distillation
Alien	1724	3096	3419	9007	14682	15723
SpaceInvaders	1171	26821	30124	9848	76246	111189
Asterix	4850	59020	64510	708500	749870	849400
Asteroids	2095	4131	4647	5690	15371	15947
Assault	10164	8088.4	9628.5	14470	11697	14052
BattleZone	7600	14200	14400	5800	16300	17500
BeamRider	5530	14417	21519	23984	38311	39604
Boxing	4.2	2.8	100	100	100	100
Centipede	5025	5800	6575.5	6690	7744.3	8056.9
ChopperCommand	1320	15900	19120	11170	26320	31190
CrazyClimber	118300	138610	145700	128710	135290	138470
DemonAttack	318349	463823	483490	481818	517801	521051

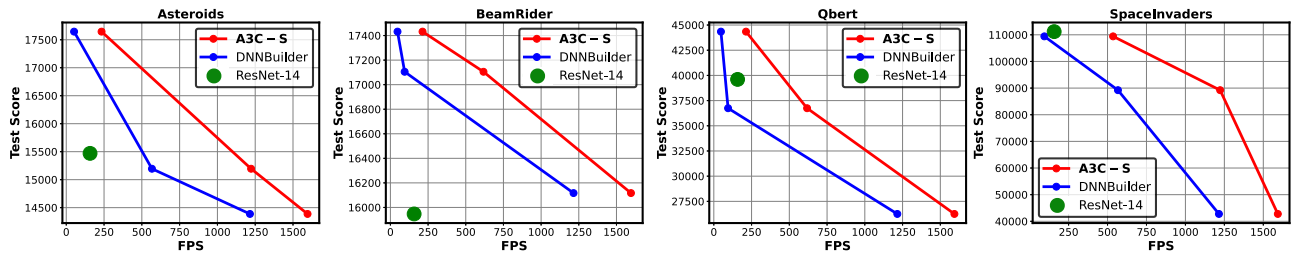


Fig. 3. Benchmark the proposed A3C-S with (1) ResNet-14 on our DAS’s searched accelerators and (2) A3C-S searched agents on A3C-S searched accelerators vs. SOTA accelerators DNNBuilder [26], in terms of test scores and FPS trade-off on four Atari games.

A3C-S searched agents on A3C-S searched accelerators vs. SOTA accelerators. Here we evaluate A3C-S’s DAS engine by comparing the hardware efficiency (i.e., FPS) of A3C-S’s searched agents, when being accelerated by both our DAS’s generated accelerators and a SOTA DNN accelerator, DNNBuilder [26]. For a fair comparison, both accelerators adopt the optimal DRL agents searched by our A3C-S. As shown in Fig. 3, the FPS achieved by DAS’s generated accelerators consistently outperforms that of the SOTA DNN accelerator, under the same DSP limit (900; which is the largest resource in our ZC706 [27]). This set of experiments validates the need for dedicated accelerators for DRL agents, of which our A3C-S is the first demonstration.

A3C-S vs. SOTA DRL solutions. To evaluate the combined benefits of our A3C-S, we evaluate A3C-S’s resulting accelerators (i.e., using A3C-S’s DAS to generate optimal accelerators for A3C-S’s searched agents) over a SOTA DRL system FA3C [28], where the performance of the latter is directly obtained from the reported data of the baseline paper. From Tab. III, we can see that our A3C-S’s resulting DRL accelerators achieve $2.1\times \sim 6.1\times$ better FPS, while offering consistently higher test scores, as compared to the SOTA DRL system [28]. As expected, jointly searching for both the DRL agents and their accelerators leads to optimal DRL solutions, achieving both higher test scores and better hardware efficiency. Furthermore, our A3C-S’s differentiable search strategy makes it more accessible to researchers without paramount computing resources and facilitates fast development of DRL-powered solutions.

TABLE III
TEST SCORES / FPS OF OUR A3C-S COMPARED WITH FA3C ON SIX ATARI GAMES REPORTED BY FA3C [28].

Atari Games	FA3C	A3C-S
BeamRider	3100 / 260	36745 / 617.7
Breakout	340 / 260	670 / 1596.3
Pong	0 / 260	20.9 / 787.4
Qbert	6100 / 260	15194 / 1222.9
Seaquest	170 / 260	478940 / 778.1
SpaceInvaders	830 / 260	109417 / 535.6

VI. CONCLUSION

We propose, design, and validate a DRL agent accelerator co-search framework dubbed A3C-S, which to our best knowledge is the first to (1) automatically co-search for the optimally matched DRL agents and accelerators that maximize both test scores and hardware efficiency, and (2) demonstrate the first successful DNAs for DRL, for which a vanilla DNAs fails due to DRL’s training instability.

ACKNOWLEDGEMENTS

The work is supported by the National Science Foundation (NSF) CCRI-2016727 and CAREER-2048183 Awards.

REFERENCES

- [1] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [2] W. Jiang *et al.*, “Hardware/software co-exploration of neural architectures,” *arXiv preprint arXiv:1907.04650*, 2019.
- [3] Y. Li *et al.*, “Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions,” *DAC*, 2020.
- [4] Y. Zhang *et al.*, “DNA: Differentiable network-accelerator co-search,” 2020.
- [5] R. Cheng *et al.*, “Control regularization for reduced variance reinforcement learning,” *arXiv preprint arXiv:1905.05380*, 2019.
- [6] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” in *ICML*, 2016, pp. 1928–1937.
- [7] K. Arulkumaran *et al.*, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [8] Y. Fu *et al.*, “Autogan-Distiller: Searching to compress generative adversarial networks,” in *ICML’20*.
- [9] B. Wu *et al.*, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in *CVPR*, 2019.
- [10] Y. Zhao *et al.*, “SmartExchange: Trading higher-cost memory storage/access for lower-cost computation,” in *ISCA’20*, p. 954–967.
- [11] W. Li *et al.*, “Timely: Pushing data movements and interfaces in pim accelerators towards local and in time domain,” in *ISCA’20*.
- [12] R. Venkatesan *et al.*, “MAGNet: A Modular Accelerator Generator for Neural Networks,” in *ICCAD*, 2019.
- [13] P. Xu *et al.*, “Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics,” *arXiv preprint arXiv:2001.03535*, 2020.
- [14] R. Sutton *et al.*, “Policy gradient methods for reinforcement learning with function approximation,” in *NeurIPS*, 2000, pp. 1057–1063.
- [15] V. Konda *et al.*, “Actor-critic algorithms,” in *NeurIPS*, 2000.
- [16] S. Xie *et al.*, “Snas: stochastic neural architecture search,” *ICLR*, 2019.
- [17] H. Liu *et al.*, “Darts: Differentiable architecture search,” *arXiv preprint arXiv:1806.09055*, 2018.
- [18] C. He *et al.*, “Milenas: Efficient neural architecture search via mixed-level reformulation,” in *CVPR*, 2020, pp. 11993–12002.
- [19] H. Cai *et al.*, “Proxylessnas: Direct neural architecture search on target task and hardware,” *arXiv preprint arXiv:1812.00332*, 2018.
- [20] E. Jang *et al.*, “Categorical reparameterization with gumbel-softmax,” *arXiv preprint arXiv:1611.01144*, 2016.
- [21] Y. Shen *et al.*, “Maximizing cnn accelerator efficiency through resource partitioning,” in *ISCA*. IEEE, 2017, pp. 535–547.
- [22] A. Rusu *et al.*, “Policy distillation,” *arXiv:1511.06295*, 2015.
- [23] M. G. Bellemare *et al.*, “The arcade learning environment: An evaluation platform for general agents,” *JAIR*, vol. 47, pp. 253–279, 2013.
- [24] Xilinx Inc., “Vivado High-Level Synthesis,” shorturl.at/vyOPR.
- [25] Y. Zhao *et al.*, “DNN-Chip Predictor: An analytical performance predictor for dnn accelerators with various dataflows and hardware architectures,” in *ICASSP’2020*.
- [26] X. Zhang *et al.*, “Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas,” ser. ICCAD, 2018.
- [27] Xilinx Inc., “Xilinx zynq-7000 soc zc706 evaluation kit,” <https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html>.
- [28] H. Cho *et al.*, “Fa3c: Fpga-accelerated deep reinforcement learning,” in *ASPLOS’19*, 2019, pp. 499–513.