RADARS: Memory Efficient Reinforcement Learning Aided Differentiable Neural Architecture Search

Zheyu Yan* Weiwen Jiang† Xiaobo Sharon Hu* Yiyu Shi*
*University of Notre Dame †George Mason University

2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC) | 978-1-6654-2135-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASP-DAC52403.2022.9712499

Differentiable neural architecture search (DNAS) is known for its capacity in the automatic generation of superior neural networks. However, DNAS based methods suffer from memory usage explosion when the search space expands, which may prevent them from running successfully on even advanced GPU platforms. On the other hand, reinforcement learning (RL) based methods, while being memory efficient, are extremely time-consuming. Combining the advantages of both types of methods, this paper presents RADARS, a scalable RL aided DNAS framework that can explore large search spaces in a fast and memory-efficient manner. RADARS iteratively applies RL to prune undesired architecture candidates and identifies a promising subspace to carry out DNAS. Experiments using a workstation with 12 GB GPU memory show that on CIFAR-10 and ImageNet datasets, RADARS can achieve up to 3.41% higher accuracy with 2.5X search time reduction compared with a state-of-the-art RL-based method, while the two DNAS baselines cannot complete due to excessive memory usage or search time. To the best of the authors' knowledge, this is the first DNAS framework that can handle large search spaces with bounded memory usage.

I. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art performance in various applications such as computer vision and machine translation. To push DNNs to achieve even higher performance, Neural Architecture Search (NAS) has been proposed to automatically identify neural architectures that can outperform their hand-crafted counterparts. Existing NAS frameworks are either based on reinforcement learning (RL), or a differentiable approach.

RL-based NAS frameworks (RL-NAS) sample one neural architecture at a time and use its performance as reward/punishment to guide an RL controller [1]. These frameworks are memory efficient and can handle a variety of optimization objectives beyond accuracy, such as power efficiency, latency, *etc.*, for hardware and neural architecture co-exploration. However, the need for sampling and training a large number of architectures leads to extremely long search time, sometimes exceeding 1,000 GPU hours.

To solve this issue, researchers propose differentiable neural architecture search (DNAS) that trains an over-parameterized network, named SuperNet, which contains all candidate paths and reduces the search time to tens of GPU hours. Yet the inclusion of different candidate paths puts great pressure on GPU memory. Existing DNAS based works [2, 3, 4] carefully control their search spaces and experiment hyper-parameters (*e.g.*, batch size) to keep the memory usage lower than the GPU memory size to avoid out-of-memory (OOM) issues. However, the bottleneck still remains, *i.e.*, the number of candidate paths grows exponentially *w.r.t.* the number of hyper-parameter types, because each combination of different hyper-parameter types (*e.g.*, convolution kernel size, and quantization precision) is considered as an instance of the search candidate. As more hyper-parameters are needed in hardware-architecture co-design, the GPU memory usage can grow orders of magnitude higher, beyond the capacity of many acceleration platforms.

One-Shot Architecture Search (One-Shot DNAS) [5] are among the few attempts to alleviate the memory pressure of DNAS by activating only one candidate path during the SuperNet training. As such, the GPU memory usage can be reduced to the level same as that of training one single neural architecture because only one candidate path is activated in the forward inference and back propagation process. However, the method does not reduce the size of the SuperNet, which still needs to be stored off the GPU memory if too large. In such a scenario, frequent communication between GPU and CPU is needed to load the candidate paths, resulting in significant search time overhead.

Fundamentally speaking, the major cause for memory usage explosion in DNAS is the exponential growth of search space. In vanilla DNAS, every possible candidate path is added to the search space to form a homogeneous SuperNet. However, not all candidates are reasonable choices. For example, higher bit precision is typically needed by layers closer to the input or output. As such, naively including all possible candidates for every layer leads to a significant waste of memory and computation. It is important to prune these unwanted candidates before initiating DNAS. **The challenge then is how to find a subspace in the designated search space that (1) fits in the provided acceleration platform with bounded GPU memory, and (2) contains the optimal neural architecture.**

In this work, we propose RADARS[1], an effective solution to reduce the GPU memory usage of DNAS. It prunes the target search space with the guidance of RL, using a method with iterative exploration/exploitation phases. As shown in Fig. 1, the original search space for vanilla DNAS includes an identical set of candidate operations across different layers. In each exploration phase, by analyzing high-performance

[1]This work is supported in part by National Science Foundation under grant CNS-1822099 and CNS-1919167.

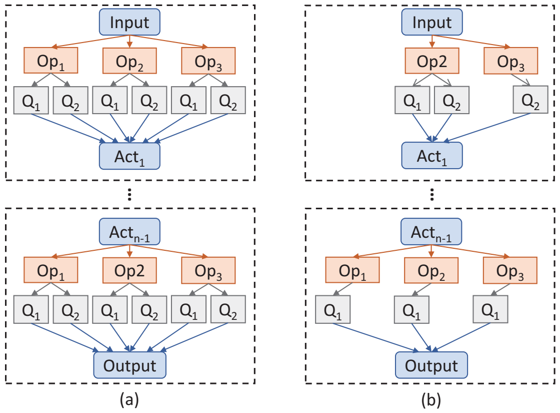

Fig. 1. Conceptual illustration of NAS search space pruning. (a) SuperNet created from the original search space, and (b) reduced SuperNet created by RL-aided pruning.

architectures generated by the controller, RL learns to better identify a subset of candidates in each layer that are promising to offer desired performance. Then in the following exploitation phase, a DNAS process identifies the best architecture in the RL-identified search subspace.

We conduct experiments on the CIFAR-10 and ImageNet datasets with large search spaces using a workstation with two RTX-2080Ti GPU cards (12 GB memory). RADARS can achieve strong empirical results on both datasets, achieving up to 3.41% lower test error and 39% lower arithmetic intensity [4] with 2.5x speedup compared with a state-of-the-art RL based NAS implementation [6]. On the other hand, the DNAS [4] and One-Shot DNAS [5] methods cannot finish due to OOM and excessive search time issues, respectively. The superiority of RADARS over existing DNAS methods is further confirmed as the resultant DNN models share similar performance with other state-of-the-art NAS framewors. To the best of our knowledge, RADARS is the first DNAS framework that can efficiently handle large search spaces with bounded memory usage.

II. Background

Neural Architecture Search (NAS) aims to explore through a pre-determined set of neural architectures (*i.e.*, search space) and select one neural architecture that can offer optimal performance. In this section, we briefly review two different categories of NAS methods and their variants, as our method effectively bridges these two methods.

RL-based Neural Architecture Search (RL-NAS) [1, 7, 8, 9, 10, 11] is well known for its memory efficiency in searching neural architectures. It is composed of three key components, a *controller*, a *trainer*, and an *evaluator*. In one iteration (episode) of RL-NAS, the *controller* predicts a neural architecture from the search space; the *trainer* trains the predicted neural architecture, named child network, from scratch on a training dataset for a number of epochs, and the *evaluator* collects the figures of merit (FOM), *e.g.*, latency and/or energy consumption of the child network and its test accuracy of the trained child network on test dataset. The FOM then forms a *reward* function, based on which the *controller* is updated so that it can predict neural architectures with higher FOM. The neural architecture that offers the highest *reward* among all the searched neural architectures is presented as the search result.

Differentiable Neural Architecture Search (DNAS) is prevailing for its ability to search neural architectures in a time-efficient manner. One typical way of DNAS is to construct and directly train an over-parameterized network, called SuperNet, that contains all possible candidate paths. Each candidate path is associated with an *architecture weight*. The *architecture weights* are differentiable with respect to the loss function of the SuperNet, so the *architecture weights* can be optimized with respect to the loss function by gradient descent. For a specific training task, the neural network weights and the *architecture weights* are updated alternately. The final chosen DNN is composed of the candidates with maximum *architecture weights* in each layer. Some variants of DNAS [12] incorporate RL algorithms in the search process to estimate gradients for non-differentiable objectives and help broaden the application field of DNAS. However, the adoption of RL does not help reduce the memory usage of DNAS.

The scalability issue is a major concern when using DNAS. DNAS is not scalable because the introduction of SuperNet requires the weight values and architecture parameters of the whole search space to be stored in memory. One-Shot Architecture Search [5] alleviates this issue by reducing the training process from requiring the whole SuperNet in the forward and backward path to only including one selected candidate path of the SuperNet in one iteration of training. This greatly reduces the requirement of both memory usage and computational power. However, One-Shot Architecture Search does not reduce the size of the SuperNet, which grows exponentially *w.r.t.* the number of hyper-parameters. When it grows beyond the memory capacity of GPUs, excessive communication between GPU and CPU is needed which leads to a much longer search time.

III. RL Aided Differentiable NAS

Below, we first give an overview of RADARS, then describe its key components. We finally analyze the memory usage of DNAS and show how RADARS can reduce it.

A. Overview

In this work, we propose Reinforcement learning Aided Differentiable neural ARchitecture Search (RADARS). Different from existing works that focus on reducing the implementation overhead of DNAS, we directly prune the search space so that the GPU memory usage can be bounded. The fundamental challenge here is to identify the promising subspace in the entire search space efficiently, towards which no simple heuristics exist.

The most straightforward approach is to divide the designated search space into a group of non-overlapping subspaces, each of which can fit in the GPU memory. We can then perform DNAS in each subspace and find the best architecture [13]. The issue with this method is that it only reduces the number of subspaces needed for search by a constant factor thus the search time is the same order of magnitude as RL-NAS.

Inspired by the RL-NAS, we adopt an RL-based scheme to quickly prune undesired candidate paths and predict a promising subspace for DNAS so that it no longer needs to fully explore the entire search space.

There are two potential issues for adopting such a pruning scheme, however. First, the RL process is known to be slow. Second, if RL happens to miss a promising candidate, the following DNAS will never be able to add it back. To tackle these issues, we devise a search process that interleaves the exploration/exploitation phases, (see in Alg. 1). Specifically, each exploration phase (detailed in Section B) predicts a set of promising neural architectures **SN**. Then, the following exploitation phase (detailed in Section C) builds a SuperNet based on **SN**, reflecting a pruned search space, and conducts a DNAS search. The two phases are carried out iteratively until one of the following stopping criteria is met: 1) the search *reward* is greater than a preset *target* or 2) the number of episodes exceeds a predefined limit. In our experiments, we find that RADARS typically converges in 50 episodes. The implementation details are discussed below.

Algorithm 1 RADARS (S, D, N, P, Ep, Tar) $\rightarrow C$

```
1: // input: search space S, target task dataset D, # of
   episodes in exploration phase $N$, # of architectures to
   re-train $P$, maximum searching iteration $Ep$, and the
   target performance $Tar$;
2: // output: best neural architecture identified $C$;
3: Initialize empty result set, R;
4: Initialize RL search process using S and D;
5: while $reward < Tar$ and # of iteration $< Ep$ do
6:     //Exploration phase:
7:     Continue to run RL search for $N$ episodes. It gener-
       ates $N$ reward and architecture pairs $\mathbf{R}_i = \{< r_i, a_i > |i = 1, .., N\}$;
8:     Append $\mathbf{R}_i$ to R. ($\mathbf{R} \leftarrow \mathbf{R} \cup \mathbf{R}_i$);
9:     Identify the top $P$ rewards in R;
10:    Re-train the $P$ architectures identified;
11:    Update R with the new rewards after re-training;
12:    Initialize an empty set of Neural Architecture SN;
13:    for $< r, a >$ in R in descendent order of $r$ do
14:        Add $a$ to SN and build a SuperNet from SN;
15:        if Memory usage > GPURAM capacity then
16:            Remove $a$ from $SN$; Break;
17:        end if
18:    end for
19:    // Exploitation phase:
20:    Build a DNAS search scheme based on SN;
21:    Perform DNAS and keep track of the best architecture
       identified so far $C$ and gather the $reward$ of $C$;
22: end while
```

B. Exploration Phase

This phase aims to identify promising architectures in the search space so that the following exploitation phase can focus on a pruned space formed by these architectures instead of the entire search space.

Similar to a conventional RL-NAS as discussed in Section II, We use a *controller*, a *trainer*, and an *evaluator* for the exploration phase. However, different from the RL-NAS that trains the *controller* until an optimal architecture is found, which can be extremely slow, each time this phase only runs for N episodes (continuing from the previous run) and then predicts a set of architectures **SN**. The episode number N is specified by the user. Smaller N means higher granularity, thus offering higher performance but requiring longer search time.

Reckoning the fact that the performance of RL in its early stages may not be good, we keep track of all the architectures and the associated rewards identified in the exploration phase. At the end of the t^{th} exploration phase, a total of $N \cdot t$ architectures and rewards are explored and the top architectures in them form the pool to identify the most promising subset **SN**.

Moreover, the *trainer* only trains the predicted architecture for a limited number of epochs for the sake of time efficiency. Thus, the rewards gathered by the *evaluator* (described in Section D) are inaccurate, and selecting the top ones based on such inaccurate rewards can easily miss out on good candidates. To address this issue, we take a two-step pruning approach. We first identify P architectures with the highest rewards out of the $N \cdot t$ architectures in the pool as the candidates, where P is specified by the user. These P architectures are then trained to converge to get their exact rewards. Apparently, a larger P provides better performance (with more precise rewards) at the cost of longer search time. P is chosen in early experiments so that the execution time of the exploration and exploitation phase are equal and can thus be executed in parallel without waiting for each other's output. With the P candidate architectures and their exact rewards, we can further identify a subset of them with the highest rewards to form **SN**, which are then used in the exploitation phase to form a SuperNet and identify the best architecture.

C. Exploitation Phase

In the exploitation phase, we first take in the neural architecture set **SN** generated in the exploration phase to build a SuperNet that is under GPU memory capacity and reflects the pruned search space.

DNAS search space is organized in a layer-by-layer manner. In each layer, every candidate operation is added into the SuperNet, associated with an *architecture weight*. Different layers are then stacked on top of each other to form a SuperNet. There are $|\mathbf{SN}|$ candidates in **SN** and assume that each candidate has L hyper-parameters. Then they can form a 2D matrix of $L \times |\mathbf{SN}|$, as shown in Fig. 2 (a). In each layer, redundant candidates are removed and a compact SuperNet (Fig. 2 (b)) is formed on which a differential architecture search can be conducted to identify the best architecture.

D. Reward Evaluation

We use RADARS to search for neural architectures that are both accurate and hardware efficient. In this paper, we use the widely adopted arithmetic intensity [4] indicator, number of total MAC operations, as our FOM to indicate the

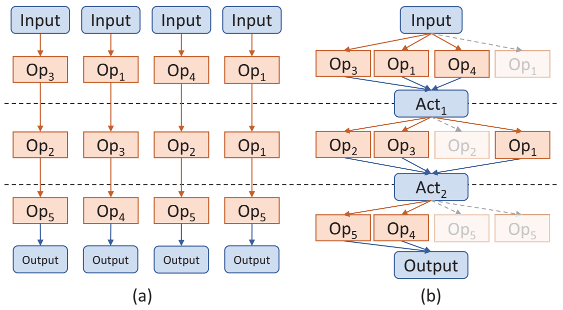

Fig. 2. Illustration for SuperNet building. (a) Architectures generated by the exploration phase, and (b) SuperNet built upon the architectures in (a). In each layer, candidate operations are added to the SuperNet and the redundant ones are discarded. *E.g.*, in layer 3, Op_4 and Op_5 are added to the SuperNet and the repeated Op_5's are not.

hardware performance of a certain neural architecture. Note that it is widely acknowledged that, although arithmetic intensity does not necessarily represents model latency, it is an appropriate indicator of energy consumption [6]. Accordingly, our framework identifies a DNN architecture that is both accurate and energy-efficient. It is understood that other metrics can also be used if desired.

Moreover, in quantized DNNs, the cost of MAC operations for different bit precisions is not the same in terms of arithmetic intensity. Thus, assuming integer MAC units are used in quantized DNN accelerators, we define adjusted number of operations (AOPS) as an arithmetic intensity indicator for quantized DNNs. AOPS can be defined as:

$$AOPS = MACOPS \times Activation\ bits \times Weight\ bits \quad (1)$$

where the number of MAC operations, bit precision of weights, and input activations are taken into consideration.

The performance evaluation metric (reward function) used in the search process can be defined as:

$$Reward = \alpha \times Acc + (1-\alpha) \times (1 - (AOPS - \beta)/\gamma) \quad (2)$$

where α, β, and γ are user-defined normalization parameters.

E. Memory Usage Analysis

DNAS is memory-hungry because its memory usage increases drastically with the expansion of the search space. Below, we show that the memory requirement of DNAS grows exponentially *w.r.t.* the number of hyper-parameters. We also show that RADARS can change this growth to constant.

Without loss of generality, we analyze a DNAS search space with L convolutional layers and T different types of hyper-parameters (*e.g.*, kernel size, bits of quantization) for each layer. For the i^{th} hyper-parameter type, there are D_i candidates to choose from.

In the l^{th} layer of the SuperNet, it gets an input sized $B \times CI_l \times WI_l \times HI_l$, where B is batch size, CI is the number of input channels, WI and HI are input width and height. It also creates an output sized $B \times CO_l \times WO_l \times HO_l$. This layer uses a kernel size of K.

In the l^{th} layer, the total number of convolution weights and gradient data can be represented by NW_l and the total number of output activation data can be represented by NA_l, respectively.

$$NW_l = CI_l \times CO_l \times K^2 \times \prod_{i=0}^{T} D_i \quad (3)$$

$$NA_l = B \times CO_l \times WO_l \times HO_l \times \prod_{i=0}^{T} D_i \quad (4)$$

Finally, the total memory usage of the whole SuperNet can be calculated by:

$$\begin{aligned} NM &= \sum_{l=0}^{L} \eta \times NW_l + \theta \times NA_l \\ &= \prod_{i=0}^{T} D_i \cdot \left(\sum_{l=0}^{L} CO_l(\eta CI_l \cdot K^2 + \theta B \cdot WO_l \cdot HO_l) \right) \end{aligned} \quad (5)$$

where η and θ are coefficients related to the data representations of weights and activations.

It can be observed from Eq. 5 that memory usage grows exponentially *w.r.t.* the number of hyper-parameter types (T), which todoblocks the exploration of more different types of hyper-parameters. For example, with the search space specified in Table I, the memory usage grows up to 112 GB. This is beyond the capacity of many acceleration platforms.

The proposed RADARS framework efficiently reduces the memory usage *w.r.t.* hyper-parameter types from exponential growth to a user-defined constant. As shown in Alg. 1, neural architecture set **SN** used to build the SuperNet is a subset of P candidates, so $|\mathbf{SN}| \le P$. Let the memory usage of a single path be M_{sp}. The memory usage of this SuperNet is $|\mathbf{SN}| \times M_{sp} \le P \times M_{sp}$. Thus, the memory usage of RADARS is bounded by a constant independent of the growth of the design space.

IV. Experiments

We apply our proposed RADARS to two different tasks: CIFAR-10 [14] and ImageNet [15] classification. All search experiments discussed in this paper are performed on a platform with two RTX-2080Ti GPU cards whose memory capacity is **12 GB**. We should point out that the platform used in our experiment is a lower-end one and is used to illustrate the potential advantage of the proposed method. Some of the algorithms that fail to run on our platform may be able to run on higher-end platforms because they have access to larger GPU memory. However, being able to reduce the memory requirements is always needed because of the ever-increasing size of neural networks [4] and the accompanied exponential growth of search space.

We compare our method with three state-of-the-art NAS approaches: one state-of-the-art RL-NAS approach, Quant-NAS [6], and two DNAS methods, DNAS [4] and One-Shot DNAS [5], using their respective implementations but on larger search spaces. If any of the methods need more than the GPU capacity, an out-of-memory (OOM) error is reported. If any of the methods cannot converge in 10× of the search time of the fastest method among them, its search process is terminated and an out-of-time (OOT) error is reported. For all data presented in this paper, the *accuracy* is an averaged result out of three different runs, and *time*

TABLE I
QUANTIZED CNN FOR CIFAR-10 SEARCH SETUPS. UPPER HALF: CONFIGURATIONS FIXED TO BE THE SAME AMONG ALL ARCHITECTURES; LOWER HALF: HYPER-PARAMETERS TO BE SEARCHED.

Hyper-Parameter type	Settings
Block type	Quantized Convolution
# of convolution layers	6
# of channels per layer	[64, 64, 128, 128, 256, 256]
Stride	[1, 2, 1, 2, 1, 2]
Kernel size choices	(1, 3, 5, 7)
# of integer bits choices	(1, 3)
# of fraction bits choices	(1, 3, 6)

TABLE II
COMPARISON BETWEEN DIFFERENT METHODS ON CIFAR-10. RADARS ACHIEVES HIGHER TOP-1 ACCURACY THAN QUANTNAS IN LESS TIME. DNAS AND ONE-SHOT DNAS CANNOT GENERATE RESULTS: DNAS FACES OUT-OF-MEMORY (OOM) ISSUE AND ONE-SHOT DNAS SUFFERS FROM EXTREMELY LONG SEARCH TIME (OOT).

Model	Top-1 (%)	AOPS (G)	Time (h)	Memory (GB)
QuantNAS [6]	84.92	4.09	22.9	1.430
One-Shot [5]	OOT	OOT	>240	1.537
DNAS [4]	OOM	OOM	OOM	112.0
RADARS (ours)	88.33	2.50	9.15	10.43

consumption refers to the time needed to identify the optimal architecture, excluding the time needed to train the identified architecture from scratch. For user-defined hyper-parameters in Alg. 1, we set $N = 10$ and $P = 6$. Other parameter settings are provided below where appropriate.

A. Experiments on CIFAR-10

We demonstrate the effectiveness of RADARS by searching for an optimal quantized CNN for CIFAR-10. The fixed design parameters and hyper-parameters included in the search space are shown in Table I.

For reward specifications, we use $\alpha = 0.5$, $\beta = 0$ and $\gamma = 10^9$ so that both accuracy and $AOPS$ are normalized to $[0, 1]$. Different runs of the search are conducted with different specifications and a variety of neural architectures are identified by RADARS and quantNAS. The results of RADARS, as well as those of quantNAS [6], DNAS [4] and One-Shot DNAS [5] are reported in Table II. Because of the memory usage explosion, DNAS requires 10.7× more memory than RADARS, far exceeding the memory capacity and leading to an OOM error. On the other hand, One-Shot DNAS requires frequent data movement between the GPU memory and CPU and is still running after 10 days, more than 26× that of RADARS (OOT error). While quantNAS also converges, RADARS achieves 3.41% lower test error and 39% lower arithmetic intensity, with 2.5× search time reduction.

Fig. 3 shows the FOM of the resulting architectures in terms of accuracy and AOPS by the two methods that are able to complete (RADARS and QuantNAS). In this figure, the x-axis and y-axis represent the AOPS and error rate, respectively. Each rectangle stands for an architecture identified by RADARS and each cross stands for one identified by quantNAS. A solution is better as it moves towards the bottom-left corner. From the results, we can see that by adopting a more efficient search scheme, RADARS can significantly push forward the Pareto frontier between accuracy and AOPS.

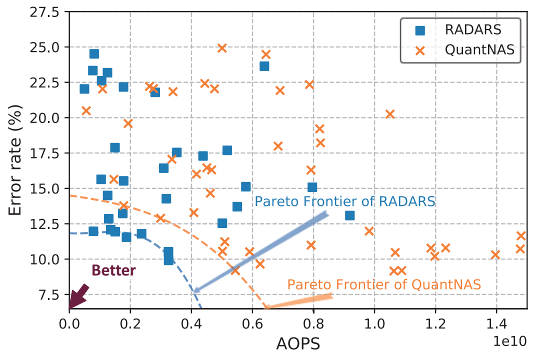

Fig. 3. Comparison between search results of RADARS and quantNAS for CIFAR-10. Each point represents one searched architecture. RADARS achieves superior Pareto frontier.

TABLE III
MOBILENET LIKE SEARCH SPACE FOR IMAGENET CLASSIFICATION EXPERIMENT SETUPS. UPPER HALF: CONFIGURATIONS FIXED TO BE THE SAME AMONG ALL SEARCHED ARCHITECTURES; LOWER HALF: HYPER-PARAMETERS TO BE SEARCHED.

Hyper-Parameter type	Settings
Block type	Mobile Invtd Res Block
# of blocks	6
# of channels per block	[24, 40, 80, 96, 192, 320]
# of cells per block	[4, 4, 4, 4, 4, 1]
Stride	[2, 2, 2, 1, 2, 1]
Kernel size choices	(3, 5, 7)
# of convolution groups	(3, 6)

B. Experiments on ImageNet

For ImageNet experiments, we focus on learning efficient CNN architectures that can offer high accuracy and at the same time require low arithmetic intensity. Thus, we use MobileNetV2 [16] as the backbone. We build a search space similar to the one used in [3] and use similar search setups as detailed in Table III.

To reduce the search time, similar to existing works [2, 4, 17], we use a proxy dataset, which is a subset of ImageNet with only 100 classes, in the exploration phase of RADARS, and use the full ImageNet dataset in the exploitation phase. For the reward function, we use $\alpha = 0.9$, $\beta = 3 \times 10^8$ and $\gamma = 3 \times 10^8$.

The experimental results are shown in Table IV. Similar to CIFAR-10 experiments, DNAS and One-Shot DNAS face OOM and OOT issues respectively, and the proposed RADARS achieves 1.4% higher top-1 accuracy than QuantNAS using only 54% of the search time.

To further study the quality of RADARS produced networks, we also compare the model identified by

TABLE IV
COMPARISON BETWEEN DIFFERENT METHODS ON IMAGENET. RADARS ACHIEVES HIGHER ACCURACY THAN QUANTNAS IN LESS TIME. ONE-SHOT DNAS SUFFERS FROM EXTREMELY LONG SEARCH TIME (OOT) AND DNAS ACES OUT-OF-MEMORY (OOM).

Model	Top-1 (%)	Top-5 (%)	AOPS (G)	Time (h)	Mem (GB)
QuantNAS [6]	72.4	90.4	0.409	138	6.73
One-Shot [5]	OOT	OOT	OOT	>740	7.01
DNAS [4]	OOM	OOM	OOM	OOM	58.2
RADARS	73.8	91.5	0.386	74	11.0

TABLE V
COMPARISON BETWEEN THE SEARCH RESULTS OF THE PROPOSED AND BASELINE METHODS ON IMAGENET. NOTE THAT THE RESULTS FROM EXISTING NAS FRAMEWORKS ARE DIRECTLY OBTAINED FROM THE RESPECTIVE LITERATURE [3, 17, 18], EACH OF WHICH USES A DIFFERENT SEARCH SPACE OF SIMILAR DIMENSION.

Type	Model	Top-1 (%)	Top-5 (%)	AOPS (G)
Hand crafted	ResNet-34 [19]	73.3	91.4	3.600
	CondenseNet [20]	73.8	91.7	0.529
	MobileNet V2 [16]	72.0	91.0	0.300
NAS identified	NASNet-A [18]	74.0	91.6	0.564
	PNASNET-5 [17]	74.2	91.9	0.588
	Proxyless-GPU[3]	75.1	92.5	0.465
Proposed	RADARS (ours)	73.8	91.5	0.386

RADARS with hand-crafted neural network baselines and models identified by existing NAS frameworks that require much longer search time or larger GPU memory . As shown in Table V, the model identified by RADARS requires 8x lower arithmetic intensity than ResNet-34 and ShuffleNet, achieves similar accuracy with 30% lower arithmetic intensity compared with CondenseNet, and achieves 1.8% higher accuracy with only 30% higher arithmetic intensity compared with MobileNet V2. Compared with the models identified by existing NAS methods, RADARS achieves lower but comparable accuracy with no less than 20% arithmetic intensity reduction (*e.g.*, 0.2% lower accuracy but 30% lower arithmetic intensity than NASNet-A). This is because, in the RADARS setup, we focus more on energy efficiency in the choice of our search space and thus it is different from those used in NASNet-A, PNASNET-5, and Proxyless-GPU. On the other hand, though the search spaces of all four NAS frameworks are of similar dimension, RADARS is 650x faster than NASNet-A and PNASNET-5, and uses 30% lower memory compared with Proxyless-GPU, based on the statistics reported in the respective literature.

To summarize, for the ImageNet dataset, RADARS can efficiently search large hardware-related design spaces, while DNAS and One-Shot DNAS do not work. On the other hand, RADARS also achieves performance comparable with other NAS baselines with design spaces that are different but of similar dimensions, while using significantly shorter search time and lower memory usage.

V. CONCLUSIONS

In this work, we propose RADARS, an RL-aided DNAS framework that can efficiently explore a large hardware-aware neural network search space while keeping the memory consumption scalable. RADARS iteratively uses an RL-based exploration phase to identify the subspaces in which the most promising architectures reside, followed by an exploitation phase to search the sub-space. Experiments on CIFAR-10 and ImageNet demonstrate the superiority of RADARS over the state-of-the-art.

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[2] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *ICLR*, 2018.

[3] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *ICLR*, 2018.

[4] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-m. Hwu, and D. Chen, "Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions," *DAC*, 2020.

[5] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *ICCV*, 2019, pp. 3681–3690.

[6] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," *ICCAD*, 2019.

[7] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *DAC*, 2019, pp. 1–6.

[8] W. t. Jiang, "Hardware/software co-exploration of neural architectures," *TCAD*, vol. 39, no. 12, pp. 4805–4815, 2020.

[9] W. Jiang, Q. Lou, Z. Yan, L. Yang, J. Hu, X. S. Hu, and Y. Shi, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 595–605, 2020.

[10] L. t. Yang, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *DAC*. IEEE, 2020, pp. 1–6.

[11] Z. Yan, D.-C. Juan, X. S. Hu, and Y. Shi, "Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 859–864.

[12] A. Vahdat, A. Mallya, M.-Y. Liu, and J. Kautz, "Unas: Differentiable architecture search meets reinforcement learning," in *CVPR*, 2020, pp. 11 266–11 275.

[13] Y. Zhao, L. Wang, Y. Tian, R. Fonseca, and T. Guo, "Few-shot neural architecture search," in *ICML*. PMLR, 2021, pp. 12 707–12 718.

[14] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*. Ieee, 2009, pp. 248–255.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[17] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *ECCV*, 2018, pp. 19–34.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[20] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *CVPR*, 2018, pp. 2752–2761.