MCMC: Multi-Constrained Model Compression via One-Stage Envelope Reinforcement Learning

Siqi Li, Jun Chen, Shanqi Liu, Chengrui Zhu, Guanzhong Tian, *Member, IEEE*, and Yong Liu, *Member, IEEE*

***Abstract*— Model compression methods are being developed to bridge the gap between the massive scale of neural networks and the limited hardware resources on edge devices. Since most real-world applications deployed on resource-limited hardware platforms typically have multiple hardware constraints simultaneously, most existing model compression approaches that only consider optimizing one single hardware objective are ineffective. In this article, we propose an automated pruning method called multi-constrained model compression (MCMC) that allows for the optimization of multiple hardware targets, such as latency, floating point operations (FLOPs), and memory usage, while minimizing the impact on accuracy. Specifically, we propose an improved multi-objective reinforcement learning (MORL) algorithm, the one-stage envelope deep deterministic policy gradient (DDPG) algorithm, to determine the pruning strategy for neural networks. Our improved one-stage envelope DDPG algorithm reduces exploration time and offers greater flexibility in adjusting target priorities, enhancing its suitability for pruning tasks. For instance, on the visual geometry group (VGG)-16 network, our method achieved an 80% reduction in FLOPs, a 2.31× reduction in memory usage, and a 1.92× acceleration, with an accuracy improvement of 0.09% compared with the baseline. For larger datasets, such as ImageNet, we reduced FLOPs by 50% for MobileNet-V1, resulting in a 4.7× faster speed and 1.48× memory compression, while maintaining the same accuracy. When applied to edge devices, such as JETSON XAVIER NX, our method resulted in a 71% reduction in FLOPs for MobileNet-V1, leading to a 1.63× faster speed, 1.64× memory compression, and an accuracy improvement.**

***Index Terms*— Deep neural networks (DNNs), model compression, multi-objective reinforcement learning (MORL), network pruning, Pareto frontier.**

I. Introduction

With the development of deep neural network (DNN) technology, the application of DNN in various fields is becoming increasingly widespread, such as autonomous driving and robotics. At the same time, neural networks are scaling up in size and increasing in depth, which leads to difficulties in deploying DNNs on edge devices or mobile with limited resources. To bridge the gap between the huge scale of neural networks and the limited hardware resources on edge devices, the model compression methods were developed. Extensive research has been conducted in the field of model compression, including pruning [1], [2], [3], [4], quantization [5], [6], [7], [8], [9], and knowledge distillation [10], [11], [12].

The key to network pruning is to determine the pruning strategy for each layer. Many handcrafted pruning methods require human exploration of the trade-off among model size, accuracy, and various other hardware indicators, such as inference time and energy consumption. Due to the large design space, handcrafted pruning is usually time-consuming, and the results are frequently suboptimal. Therefore, many automatic pruning methods have been proposed.

In particular, reinforcement learning methods are utilized for network pruning [13], [14], [15], [16], [17]. The most well known of which is the AutoML for model compression (AMC) [14], which employs a reinforcement learning agent to determine the pruning strategy of each layer to replace manual work. In deep compression with reinforcement learning (DECORE) [15], a multi-agent reinforcement learning method is utilized to decide on channel pruning. Similarly, reinforcement learning and Monte Carlo tree search (RL-MCTS) [16] incorporates the Monte Carlo tree search to enhance the sample efficiency of reinforcement learning training. In short, reinforcement learning techniques are well suited for network pruning tasks, because they are able to learn from the reward and penalty signals received during training, adapt to changes in the network, and achieve better performance and pruning rates than alternative approaches.

Although previous approaches enable automatic model compression and improve the quality of model compression, typically, they can only achieve the optimization for just one single objective. For example, with the least amount of accuracy loss, these kinds of approaches only consider computation compression and cannot consider other hardware constraints at the same time. However, most real-world engineering applications deployed on resource-limited hardware platforms typically have multiple hardware constraints simultaneously, such as latency, floating point operations (FLOPs), memory usage, energy consumption, and so on. In such cases, optimizing only one single hardware objective in the model compression

Manuscript received 30 March 2023; revised 16 October 2023; accepted 9 January 2024. Date of publication 30 January 2024; date of current version 6 February 2025. This work was supported by the Major Scientific and Technological Innovation Project of Hangzhou City under Grant 2022AIZD0019. *(Corresponding authors: Guanzhong Tian; Yong Liu.)*

Siqi Li, Jun Chen, Shanqi Liu, and Chengrui Zhu are with the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China (e-mail: lsq4747@zju.edu.cn; junc@zju.edu.cn; shanqiliu@zju.edu.cn; jewel@zju.edu.cn).

Guanzhong Tian is with the Ningbo Innovation Center, Zhejiang University, Hangzhou, Zhejiang 310027, China (e-mail: gztian@zju.edu.cn).

Yong Liu is with the State Key Laboratory of Industrial Control Technology and the Institute of Cyber-Systems and Control, Zhejiang University, Hangzhou 310027, China, and also with the Huzhou Institute, Zhejiang University, Huzhou 313000, China (e-mail: yongliu@iipc.zju.edu.cn).

Digital Object Identifier 10.1109/TNNLS.2024.3353763

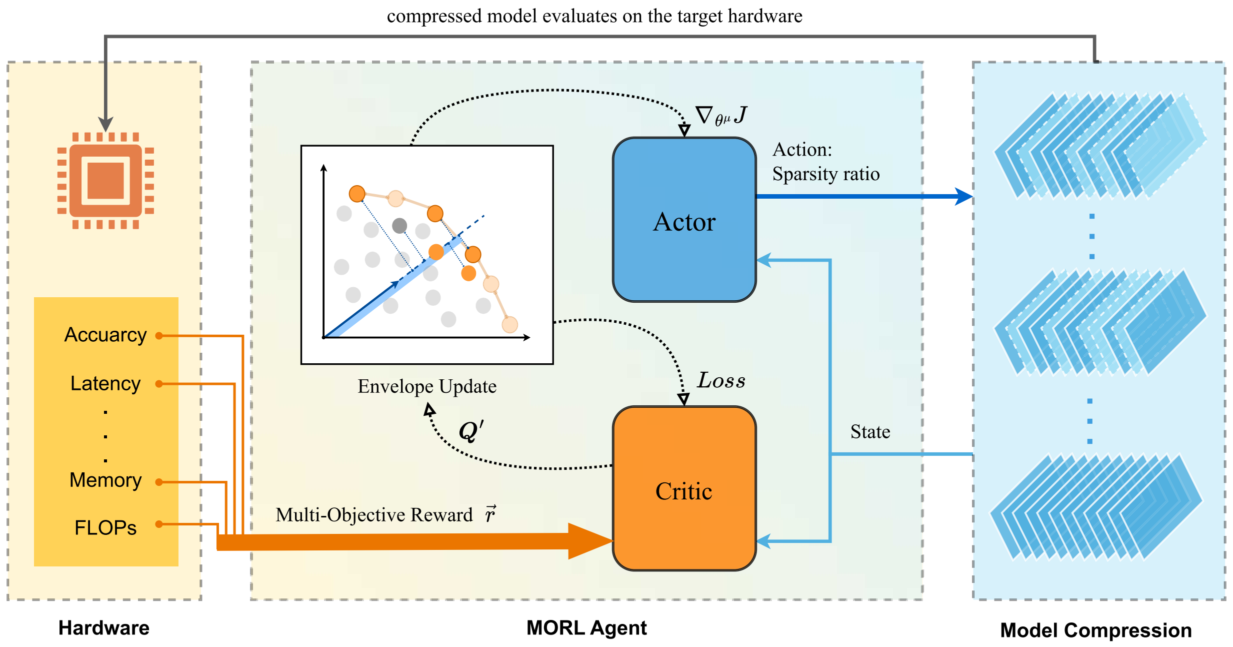

Fig. 1. Framework of MCMC via one-stage envelope DDPG algorithm. Right: sequential layer-by-layer pruning process. For each layer, the network obtains a sparsity rate from the MORL agent. Left: target hardware. After the network is pruned layer-by-layer with the sparsity rate, it runs on the target hardware. The hardware indicators we aim to optimize are measured and then returned to the MORL agent in the form of a vector as a reward. The orange arrows indicate the process of measuring hardware indicators and vectorizing rewards. Middle: MORL agent. The MORL agent receives the state from the network and the vectorized reward from the hardware, from which the agent performs envelope updating: updating the agent by using the convex envelope of the Pareto solution frontier. The dashed arrows represent internal computations within the reinforcement learning agent.

process is insufficient to meet real-time requirements and hardware constraints of realistic tasks. Notwithstanding, extant techniques for multi-objective model compression are beset with several limitations. The multi-objective neural architecture search (NAS) [18], [19], [20] consumes significant hardware resources as well as GPU time, while the existing multi-objective pruning methods [21], [22], [23], [24] restrict the number of objectives that can be optimized.

To improve the situation, we aim to develop an automatic compression algorithm, which allows full optimization for multiple hardware objectives. Ideally, all of these hardware constraints would be considered during the automated compression process, and an optimal trade-off would be taken. However, since these hardware constraints are not completely correlated with one another, using a single-objective algorithm for this task is inappropriate. Therefore, we propose the multi-constrained model compression (MCMC), an automated pruning method that leverages the multi-objective reinforcement learning (MORL) algorithm to determine the pruning strategy for DNNs, enabling simultaneous optimization of multiple hardware objectives. Fig. 1 illustrates our approach. The main contributions are summarized as follows.

1) We propose a hardware-aware automated model compression method that enables full optimization across multiple hardware objectives. Specifically, we model the MCMC problem as a multi-objective optimization problem and employ MORL to explore solutions.
2) We improve the existing MORL algorithm [25] and introduce the one-stage envelope deep deterministic policy gradient (DDPG) algorithm. Compared with the original algorithm, our proposed algorithm reduces the exploration time and supports adjusting the weights of different hardware indicators based on the needs of real-world tasks, making it better suited for our MCMC task.
3) We demonstrate the general applicability of the proposed method on multiple DNNs (visual geometry group (VGG) [26], ResNet-50/56 [27], and MobileNet-V1 [28]/V2 [29]), multiple datasets (Canadian Institute for Advanced Research, 10 classes (CIFAR-10) and ImageNet), and multiple hardware devices (1080 Ti, 2080Ti, and NX). Our approach generates superior pruning results across the accuracy and multiple hardware targets, including FLOPs, latency, and memory, by simultaneously optimizing various objectives.

II. Related Works

A. AutoML for Network Pruning

Since fine-grained pruning [1] produces irregular sparse patterns and requires specialized hardware support [30], many studies have focused on structured pruning. Many structured pruning methods have been introduced [31], [32], [33]. Furthermore, numerous methods have been proposed to achieve automated model compression, especially in the field of network pruning.

Network to network compression (N2N) learning [13] employs a recurrent neural network trained with reinforcement learning to achieve channel selection during the pruning process. AMC [14] incorporates a DDPG agent in order to determine the most effective pruning strategy. AutoPruner [34] combines the pruning and fine-tuning processes into an end-to-end system that ranks filters utilizing gradient information during fine-tuning. Also, deep reinforcement learning (DRL)-based [35] proposes a DRL-based runtime pruning approach that considers both runtime importance and static importance. Besides, generative adversarial learning (GAL) [36] uses generative adversarial learning to achieve structured pruning of networks.

Various pruning methods build on the foundation of reinforcement learning [15], [16], [17]. DECORE [15] employs a multi-agent reinforcement learning method to determine whether or not to prune each channel. RL-MCTS [16] introduces Monte Carlo tree search to improve the sample efficiency of reinforcement learning training, resulting in better filter selection. Graph neural networks with reinforcement learning (GNN-RL) [17] represents DNN as a graph, applies GNN to capture its features, and then leverages reinforcement learning to search for effective pruning strategies. These methods use various approaches to achieve AutoML for network pruning, but they only compress one single hardware target without considering optimization based on other hardware indicators.

B. Multi-Objective Model Compression

Several model compression methods consider multi-objective optimization.

NAS [37], [38], [39], [40], [41], [42] aims to search the architectural building block directly on the dataset of interest. When looking for network blocks, many NAS methods consider multiple hardware indicators. Most of them employ an evolutionary algorithm to implement NAS search [18], [19], [20]. However, there are some issues with multi-objective NAS search, one of which is that it is difficult to normalize all the objective values into the same scale. Besides, NAS search consumes a large number of hardware resources as well as GPU time.

Correspondingly, there are numerous multi-objective pruning methods have been proposed [21], [22], [23], [24]. However, not all of them achieve multi-objective optimization for multiple hardware indicators. The multi-objective proposed in NEON [22] only considers the accuracy and the compression of parameters. Evolutionary multi-objective one-shot filter pruning (EMOFP) [24] and multi-objective particle swarm optimization (MOPSO) [23] only take accuracy and sparsity rate into account. The multi-objective optimization achieved by these methods only considers the accuracy and one other hardware indicator, which is no different from regular pruning. GenExp [21] claims to consider the memory footprint constraints and the number of computations, but the number of parameters of the model is not the actual memory consumption in the forward inference.

Therefore, current multi-objective model compression methods have numerous limitations, and the number of objectives that can be optimized is extremely limited.

C. Multi-Objective Reinforcement Learning

MORL approaches can be categorized into two types [43]: single-policy approaches and multiple-policy approaches. The aim of single-policy approaches is to find the optimal single policy that simultaneously satisfies the preferences among the multiple objectives. The primary distinction between single-policy approaches is how these preferences are determined and expressed [44], [45], [46].

Multiple-policy approaches aim to find a set of policies that approximate the Pareto front. Convex hull algorithm [47] can be viewed as an extension of standard reinforcement learning, which employs an extended Bellman equation to learn optimal value functions or policies for all linear preference settings in the objective space. Moreover, another idea of multiple-policy approaches is to perform multiple runs with different parameters, objective thresholds, and orderings in any single-policy approach, and this kind of approach is known as the varying parameter approach [48].

In addition, deep optimistic linear support learning (DOL) [49] uses features from high-dimensional inputs to compute a convex cover set. The convex cover set contains all potential optimal solutions for convex combinations of objectives, enabling DOL to realize the learning of multi-objective policies. The envelope MORL algorithm [25] optimizes convex envelopes of multi-objective Q-values using a generalized version of the Bellman equation, and the agent is able to infer the hidden preference after the learning phase.

However, these MORL algorithms have their own set of applications. For the model compression task we proposed, we hope that the exploration time of our mission is as short as possible and that the importance of each objective can be adjusted based on the needs of the real-world task.

III. Methodology

A. Problem Definition

Model compression is accomplished by lowering the number of parameters and computations for each layer in DNNs. Pruning algorithms can be categorized into fine-grained pruning and structured pruning depending on the level of granularity. Since fine-grained pruning produces irregular sparse patterns that call for specialized hardware support, we select channel pruning to achieve the compression of the model by reducing the input channel of each convolution and fully connected layer.

Since taking the same sparsity for each layer of the neural network may lead to performance degradation, we consider using different sparsity rates for each layer to achieve precise compression. Our goal is to determine the effective sparsity rate of each layer so as to make the network perform better in a variety of aspects, especially in terms of hardware. Considering inference accuracy, latency, FLOPs, memory usage, energy consumption, and other performance metrics of neural networks as the objectives that we aim to optimize, the problem of multi-constrained network pruning can be formulated as follows:

$$\begin{aligned}\min_{\boldsymbol{a}\in\mathcal{A}} \boldsymbol{F}(\boldsymbol{x};\boldsymbol{a},\psi) &= \begin{bmatrix} -f_{\text{acc}}(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x})) \\ f_H^1(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x})) \\ f_H^2(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x})) \\ \vdots \\ f_H^M(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x})) \end{bmatrix} \\ \text{s.t. } \boldsymbol{a} &= [a_1,\ldots,a_L]^{\mathrm{T}}, \quad a_i \in (0,1] \\ \psi &= [w_1,\ldots,w_L] \\ \boldsymbol{x} &\in \mathcal{D} \end{aligned} \tag{1}$$

where $\boldsymbol{a}$ is the vector contained sparsity rate of each layer, $\mathcal{A} \subset \mathbb{R}^{\mathrm{L}}$ is the feasible set of $\boldsymbol{a}$, ψ denotes the neural network that will be pruned, $\boldsymbol{x}$ is the input, and $\mathcal{D}$ is the dataset. L is the total number of prunable layers in the network; for each $i \in \{1, \ldots, L\}$, each w_i in ψ is the weight of the ith layer, and each a_i in the vector $\boldsymbol{a}$ is the sparsity rate of the ith layer. The $\hat{\psi}|_{\boldsymbol{a}}(\cdot)$ is obtained by pruning the ith layer with the sparsity rate of a_i in terms of $L1$ norm. We prune the output channels of the ith layer with a pruning rate of a_i, and we apply the same operation to the input channels of the $(i+1)$th layer to match the channel dimensions

$$\hat{\psi}|_{\boldsymbol{a}}(\cdot) = \psi \odot \boldsymbol{a} = [w_1 \odot a_1, \ldots, w_L \odot a_L]. \tag{2}$$

The objective function $\boldsymbol{F}(\boldsymbol{x}; \boldsymbol{a}, \psi)$, which is defined as $F\colon \mathcal{A} \mapsto \mathbb{R}^{M+1}$, consists of the following components. The first part $-f_{\text{acc}}(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x}))$ aims to maximize accuracy of the pruned network. M is the total number of the hardware constraints, and the objectives $f_H^j(\hat{\psi}|_{\boldsymbol{a}}(\boldsymbol{x}))$, where $j \in \{1, \ldots, M\}$, require minimizing hardware overheads. In this article, we choose the latency, the total number of the floating point operations, and the memory consumption in the forward inference as three hardware objectives, since they are common and important in practical engineering and can be obtained by direct feedback without the extra detection devices.

B. Multi-Objective Pruning Analysis

1) Markov Decision Process: A Markov decision process (MDP) can be represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, R \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is transition distribution, and $R\colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function. At each discrete time step t, the agent is at the state s_t where it can select from any available action a_t and proceed to the next state s_{t+1} while earning rewards $R(s_t, a_t) = r_t$. The probability of shifting to a new state s_{t+1} is determined by the transition probability $P(s_{t+1}|s_t, a_t)$. Its objective is to maximize an expectation over the discount return $R_t = \sum_t \gamma^t r_t$, where $\gamma \in [0, 1]$ is a discount factor.

In the automated channel pruning process, for each prunable layer L_t, we characterize the state s_t by using the features of this layer and the layers before. The action a_t can be defined as the sparsity of the layer L_t. After pruning the L_t layer with a_t, the agent moves to the next layer L_{t+1} and receives the next state s_{t+1}. The reward R is evaluated on the validation set after completing the final layer L_T pruning. The completion of pruning for the final layer L_T can be regarded as the terminal of this particular episode. Taking the $s_t = (\mathcal{L}_t, \text{reduced}, a_{t-1})$ as the state, where $\mathcal{L}_t$ is the set of features of this layer, reduced is the total number of reduced FLOPs in previous layers, and a_{t-1} is the sparsity rate of the front layer. It can be seen that the future behavior of the process depends only on its present state and not on its past state, which is in line with the Markov property. It can be inferred that the transition probability for an arbitrary sequence of states $s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t$ satisfies the Markov property

$$P(s_t|s_0, a_0, s_1, a_1, \ldots, s_{t-1}, a_{t-1}) = P(s_t|s_{t-1}, a_{t-1}). \tag{3}$$

Therefore, the above sequential layer-by-layer pruning process can be modeled as an MDP.

2) Multi-Objective MDP: For a multi-objective problem (MOP), our goal is to find a set of optimal trade-offs, which is known as Pareto optimal solutions. The Pareto optimal is defined formally as follows. Considering $\mathcal{Y} = \{y \in \mathbb{R}^m\colon y = \boldsymbol{F}(x), x \in X\}$, a point $y' \in \mathcal{Y}$ is said to strictly dominate $y'' \in \mathcal{Y}$, if and only if $\forall i \in \{0, \ldots, m\}\colon y_i' \le y_i''$ and $\exists j \in \{0, \ldots, m\}\colon y_j' < y_j''$, written as $y' \prec y''$. Thus, the Pareto frontier can be defined as follows:

$$P(\mathcal{Y}) = \{y' \in \mathcal{Y}\colon \{y'' \in \mathcal{Y}\colon y'' \succ y', y'' \neq y'\} = \emptyset\}. \tag{4}$$

A multi-objective MDP (MO-MDP) can be represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \boldsymbol{\mathcal{R}}, \Omega \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is transition distribution, $\boldsymbol{\mathcal{R}}\colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^N$ is the vector reward space, and Ω is the preference space. When all possible MO-MDP solutions are considered, the Pareto frontier can be defined as follows:

$$P(\boldsymbol{R}) = \{\boldsymbol{r}' \in \boldsymbol{R}\colon \{\boldsymbol{r}'' \in \boldsymbol{R}\colon \boldsymbol{r}'' \succ \boldsymbol{r}', \boldsymbol{r}'' \neq \boldsymbol{r}'\} = \emptyset\} \tag{5}$$

where the discount return is $\boldsymbol{r'} = \sum_t \gamma^t \boldsymbol{r}(s_t, a_t)$.

In addition, it is also necessary to define the convex coverage set (CCS) of the Pareto frontier. Considering the MO-MDP problems with linear preference $\boldsymbol{\omega}$, the scalar utility of the preference $\boldsymbol{\omega}$ can be defined as $u_{\boldsymbol{\omega}}(\boldsymbol{r}) = \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{r}$. With all the possible preferences in Ω, the CCS of the Pareto frontier is defined as follows:

$$\text{CCS} = \{\boldsymbol{r}' \in P(\mathcal{R}) | \exists\, \boldsymbol{\omega}, \forall\, \boldsymbol{r}'' \in P(\mathcal{R}) \text{ s.t. } \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{r}' \ge \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{r}''\}. \tag{6}$$

The CCS contains all of the returns with the highest utility [50].

As a sequential layer-by-layer pruning process can be modeled as an MDP, the multi-objective pruning problem defined in (1) can naturally be modeled as an MO-MDP. It can be solved by MORL.

3) Bellman Equation: Considering a standard single-objective reinforcement learning algorithm, the Bellman optimality operator B is defined as follows:

$$\begin{cases} (BQ)(s, a) = \mathbb{E}_{s' \sim \mathcal{P}}[r(s, a) + \gamma (HQ)(s')] \\ (HQ)(s') = \max\limits_{a'} Q(s', a') \end{cases} \tag{7}$$

where the operator H is the optimality filter. Due to the vectorization of value functions and rewards in the MO-MDP, the standard Bellman optimality operator is no longer applicable to the MORL algorithm. Therefore, we extend the standard Bellman optimality operator as follows:

$$\begin{cases} (\mathcal{B}\mathbf{Q})(s, a, \boldsymbol{\omega}) = \mathbb{E}_{s' \sim \mathcal{P}}[\boldsymbol{r}(s, a) + \gamma (\mathcal{H}\mathbf{Q})(s', \boldsymbol{\omega})] \\ (\mathcal{H}\mathbf{Q})(s', \boldsymbol{\omega}) = \arg_{Q} \max\limits_{\substack{a' \in \mathcal{A} \\ \boldsymbol{\omega}' \in \Omega}} \boldsymbol{\omega}^T \mathbf{Q}(s', a', \boldsymbol{\omega}') \end{cases} \tag{8}$$

where $\mathbf{Q}$ is the vectored Q value, $\boldsymbol{\omega}$ is the linear preference sampled from the preference distribution, and $\boldsymbol{r}$ is the vectored rewards. The new Bellman optimality operator extends scalar Q value and scalar rewards to the vector field. Furthermore, we define a new optimality filter $\mathcal{H}$, which considers not only actions but also the length of the projection of $\mathbf{Q}$ in the $\boldsymbol{\omega}$ direction. The projection of $\mathbf{Q}$ in the direction of the linear preference $\boldsymbol{\omega}$ can represent the scalar utility $u_{\boldsymbol{\omega}}(\boldsymbol{r})$.

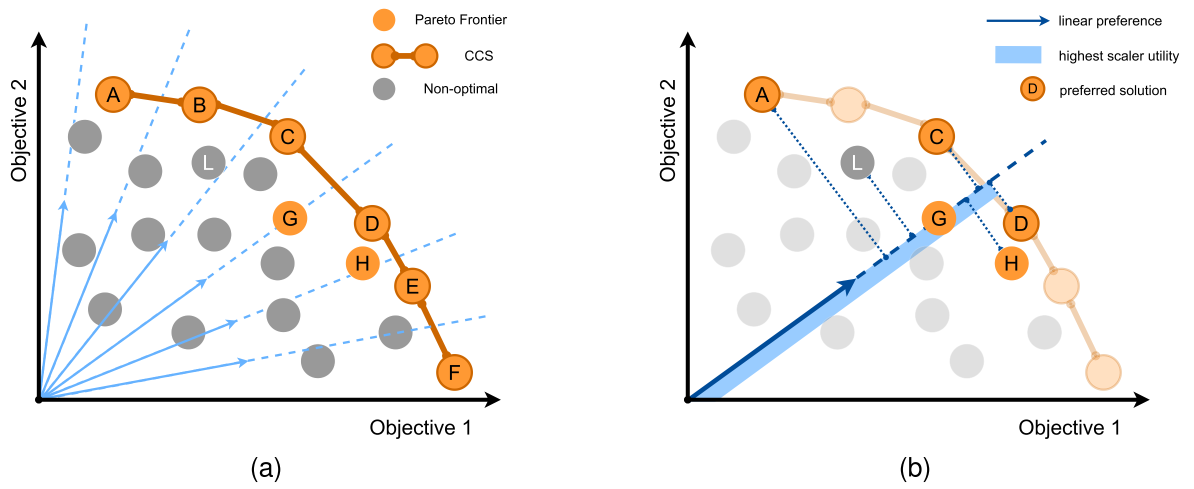

Fig. 2. (a) Pareto frontier is indicated by points A–H. The CCS, which is a convex subset of the Pareto frontier (points A–H), is represented by points A–F. Nonoptimal solutions, such as point L, are indicated by gray points. Blue arrows represent the possible linear preference vectors $\boldsymbol{\omega}$ we sampled from Ω. (b) For a point on the CCS, such as point D, there exists a linear preference vector, such as the dark blue arrow, such that its projection along the direction of the vector is higher than any point's. The projection length along the preference vector can be indicated by the scalar utility $u_{\boldsymbol{\omega}}(\boldsymbol{r})$. During the updating, we sample several linear preferences from the Ω [the blue arrows in (a)], and we then select the point with the highest scalar utility for each preference, thus realizing the sampling of the entire CCS. By updating the parameters across the entire CCS (points A–F), our approach is able to find a better solution.

The optimality filter $\mathcal{H}$ selects the Q value with the highest accumulation of scalar utility, which means it selects points in the CCS of the Pareto frontier. This approach utilizes the entire convex envelope to update parameters, which allows our algorithm to find a better, more global solution. Fig. 2 shows an example of the Pareto frontier and CCS and how the points in CCS are selected while updating.

C. MORL for Model Compression

Since the network pruning process can be modeled as an MO-MDP, It can be solved by MORL. We leverage a one-stage envelope DDPG algorithm for the multi-objective layer-by-layer sparsity rate search. Here, we introduce the details of the one-stage envelope DDPG algorithm in Algorithm 1. The improvements we made to the original method are as follows.

1) To reduce exploration time, we combine the learning phase and adaptation phase of the original envelope Q learning [25].
2) To provide flexibility in compression, we allow for the adjustment of the preference $\boldsymbol{\omega}_0$ of hardware indicators according to the requirements of real-world tasks.

1) State Space: In order to make the layer-by-layer pruning algorithm satisfy the Markov property, we select the following features as the state:

$$s_t = (\mathcal{L}_t, \text{reduced}, a_{t-1}) \tag{9}$$

where $\mathcal{L}_t$ is the set of features of the tth layer; including the index t, the layer type (the convolutional layer or the fully connected layer), the number of input channels C_{in} and output channels C_{out}, the stride s, the kernel size k, and the weight size w; the reduce is the total number of reduced FLOPs for all previous layers; and the a_{t-1} is the sparsity rate of the front layer. s_t is a complete summary of the states of all previous layers.

2) Action Space: Since model compression is highly sensitive to sparsity, we use a continuous action space $a \in (0, 1]$, which allows our method to perform a more precise search. Besides, we add noise to DDPG policies during training to improve their exploration. Finally, a is rounded to the nearest feasible fraction for each layer pruning in the compression process.

3) Reward Function: In the MORL algorithm, we extend the reward function to the vector field. According to the problem defined in (1), we select the four most widely used and important targets to form the vectored reward function

$$\boldsymbol{r} = [\text{acc}, -\text{FLOPs}, -\text{latency}, -\text{memory}]. \tag{10}$$

The reward function is evaluated on the target hardware, where the acc is the accuracy evaluated on the validation set, FLOPs is the total number of the floating point operations, latency is the mean value of the measured latency, and memory is the memory consumption in the forward inference. The values of all elements are normalized between 0 and 1, which solves the problem of nonuniform scales for different objectives

$$\begin{aligned}\text{acc} &= \text{acc} \times 100\% \\ \text{FLOPs} &= \text{FLOPs}_{\text{val}}/\text{FLOPs}_{\text{org}} \\ \text{latency} &= \text{latency}_{\text{val}}/\text{latency}_{\text{org}} \\ \text{memory} &= \text{memory}_{\text{val}}/\text{memory}_{\text{org}}.\end{aligned} \tag{11}$$

The subscript org indicates the results measured on the unpruned network, and the subscript val indicates the results after pruning.

4) Loss Function: For the vectorized $\boldsymbol{Q}$ and $\mathcal{B}\boldsymbol{Q}$, we define the loss function as follows:

$$\text{Loss}^1(\theta) = \mathbb{E}[\|\mathcal{B}\boldsymbol{Q}(s, a, \boldsymbol{\omega}) - \boldsymbol{Q}(s, a, \boldsymbol{\omega})\|^2]. \tag{12}$$

Loss^1 ensures that the prediction of Q value is close to any real expected total reward, even though Loss^1 is nonsmooth

Algorithm 1 One-Stage Envelope MO-DDPG

Input: a preference sampling distribution D_ω, a preference $\boldsymbol{\omega}_0$ set by the user, the discount factor γ, value τ for soft update.
Initial: Initialize replay buffer D_τ, a random process $\mathcal{N}$ for action exploration, critic network $\boldsymbol{Q}(s, a, \boldsymbol{\omega}|\theta^Q)$, actor network $\mu(s|\theta^\mu)$, target critic network $\boldsymbol{Q}'(s, a, \boldsymbol{\omega}|\theta^{Q'})$ and target actor network $\mu'(s|\theta^{\mu'})$. Set target parameters equals to main parameters $\theta^{\boldsymbol{Q}'} \leftarrow \theta^{\boldsymbol{Q}}, \theta^{\mu'} \leftarrow \theta^\mu$. $\lambda = \lambda_0 = 0.01$, $\delta = (1000 \times (1 - \lambda_0))^{1/M}$, $\Delta\lambda = \delta/1000$.

```
1:  for $episode = 1, \ldots, M$ do
2:      Observe state $s$ and select action
            $a = \mathrm{clip}(\mu(s|\theta^\mu) + \epsilon, a_{Low}, a_{High})$
        where $\epsilon \sim \mathcal{N}$.
3:      Receive a vectorized reward $\boldsymbol{r}$, observe next state $s'$, and done signal $d$ to indicate whether $s'$ is terminal.
4:      Store transition $(s, a, \boldsymbol{r}, s', d)$ in the replay buffer $D_\tau$.
5:      if $s'$ is terminal then
6:          Reset environment state.
7:      end if
8:      if $update$ then
9:          Sample $N_\tau$ transitions $(s_j, a_j, \boldsymbol{r}_j, s_{j+1}, d_j) \sim D_\tau$.
10:         Sample $N_\omega$ preferences $W = \{\boldsymbol{\omega}_i \sim D_\omega\}$.
11:         Compute the target value according to Eq. (8):
                $(\mathcal{B}\boldsymbol{Q})_{ij} = \boldsymbol{r}_j + \gamma(1 - d_j) \arg_{Q'} \max_{a' \in \mathcal{A}, \boldsymbol{\omega}' \in W} \boldsymbol{\omega}_i^{\mathrm{T}} \boldsymbol{Q}'(s_{j+1}, a', \boldsymbol{\omega}'; \theta')$
            for all $1 \le i \le N_\omega$ and $1 \le j \le N_\tau$
12:         Update critic by minimizing the loss according to Eq. (12) and Eq. (13):
                $Loss = (1 - \lambda)Loss^1 + \lambda Loss^2$
13:         Update actor using the policy gradient according to Eq. (15):
                $\nabla_{\theta^\mu} J \approx \frac{1}{N_\tau} \sum_i [\nabla_a \boldsymbol{\omega}_0^T \boldsymbol{Q}(s, a, \boldsymbol{\omega}_0|\theta^Q) \cdot \nabla_{\theta^\mu} \mu(s|\theta^\mu)]$
14:         Update the target networks:
                $\theta^{\boldsymbol{Q}'} \leftarrow \tau\theta^{\boldsymbol{Q}} + (1 - \tau)\theta^{\boldsymbol{Q}'}$
                $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$
15:     end if
16:     $\lambda \leftarrow \lambda + \Delta\lambda$
17:     $\Delta\lambda \leftarrow (\lambda - \lambda_0) \times \delta + \lambda_0 - \lambda$
18: end for
```

and challenging to be optimal. So, we utilize the homotopic optimization method and introduce an auxiliary loss function

$$\mathrm{Loss}^2(\theta) = \mathbb{E}[\|\boldsymbol{\omega}^{\mathrm{T}}\mathcal{B}\boldsymbol{Q}(s, a, \boldsymbol{\omega}) - \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{Q}(s, a, \boldsymbol{\omega})\|^2]. \quad (13)$$

Loss^2 contributes as an auxiliary force to pull the current estimate along the direction with improved utility. Since both functions converge to the same endpoint, we can use the linear

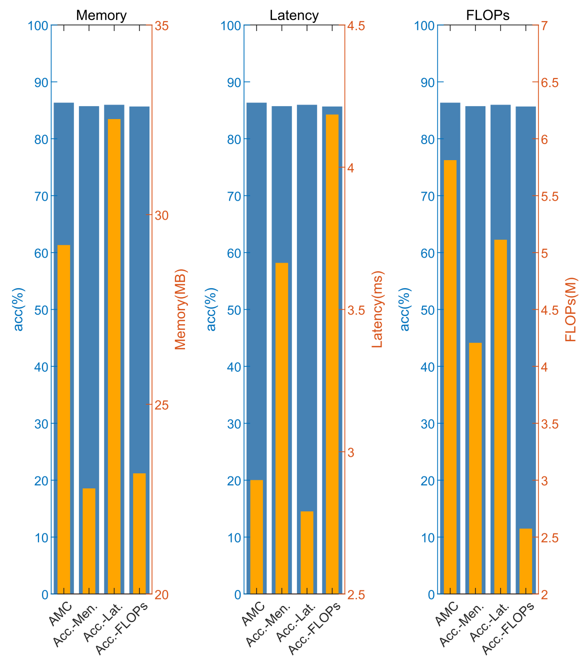

Fig. 3. Pruning results for MobileNet-V1 using MORL with two objectives. The pruning results of MobileNet-V1 using MORL with two objectives were compared with the results of the AMC [14]. The three sets of objectives are Accuracy–Memory, Accuracy–Latency, and Accuracy–FLOPs. The accuracy and memory of the pruning results above are shown in the left figure, the accuracy and latency of the pruning results are shown in the middle figure, and the accuracy and FLOPs of the pruning results are shown in the right figure. The wide blue bar and blue scale in the figure represent the accuracy, while the thin orange bar and orange scale represent memory, latency, and FLOPs in the three subfigures, respectively.

homotopy optimization to combine Loss^1 and Loss^2 together

$$\mathrm{Loss} = (1 - \lambda)\mathrm{Loss}^1 + \lambda\mathrm{Loss}^2. \quad (14)$$

Also, update the actor using the policy gradient

$$\nabla_{\theta^\mu} J \approx \frac{1}{N_\tau} \sum_i \left[\nabla_a \boldsymbol{\omega}_0^T \boldsymbol{Q}(s, a, \boldsymbol{\omega}_0|\theta^Q) \cdot \nabla_{\theta^\mu} \mu(s|\theta^\mu)\right]. \quad (15)$$

IV. Experiments

We evaluate our approach on common image classification benchmark datasets, such as CIFAR-10 and ImageNet ILSVRC2012. Also, we conduct a comprehensive analysis of our proposed MORL pruning method on the CIFAR-10 dataset. Besides, we also evaluate the performance of our approach on the edge device, such as JETSON XAVIER NX.

Within our experimental framework, the parameter configuration for our MORL agent can be outlined as follows. The Actor's learning rate is established at $1e-3$, while the Critic's learning rate is designated at $1e-4$. The discount factor γ is set to 0.99. Soft updates for the target networks are carried out using the value of τ set to 0.01, allowing for a gradual and stable update process. The batch size for the agent is set to 64, and the preference batch size N_ω is set to 8. The size of the replay buffer D_τ, which stores past experiences, is determined based on the depth of the network being pruned, specifically as the product of the number of prunable layers and a factor of 200.

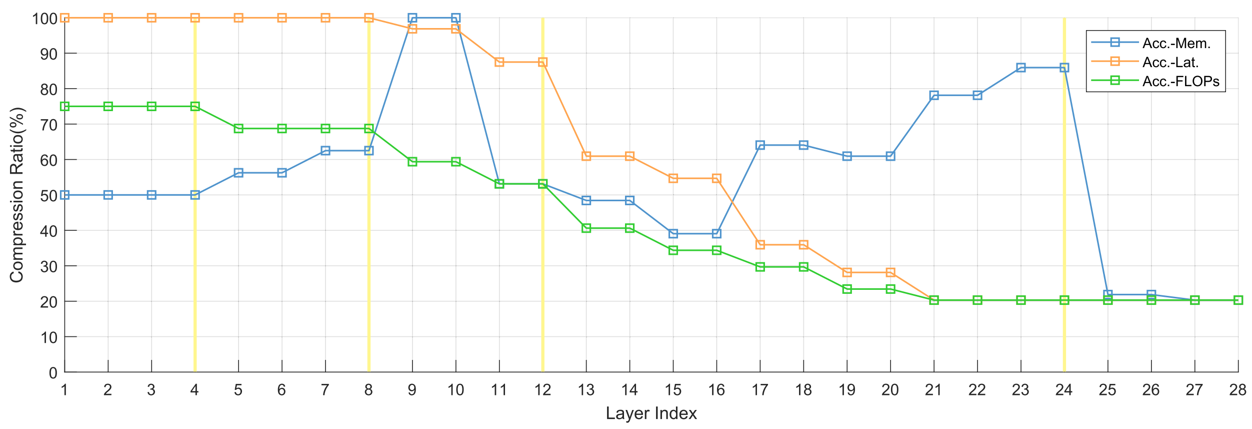

Fig. 4. Comparisons of pruning strategies under three sets of objectives. The blue line represents the pruning strategy given by the MORL agent with the Accuracy–Memory objectives. The orange line represents the pruning strategy given by the MORL agent with the Accuracy–Latency objectives. The green line represents the pruning strategy given by the MORL agent with the Accuracy–FLOPs objectives.

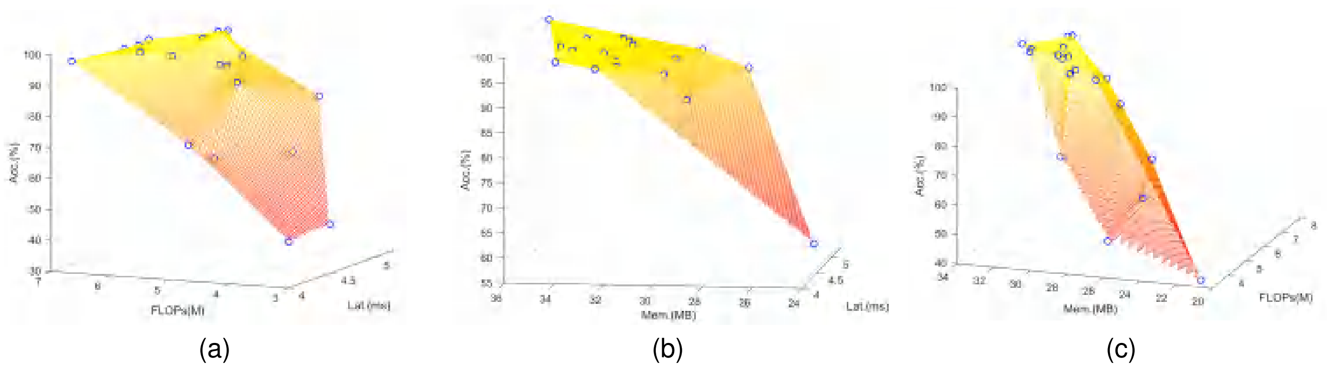

Fig. 5. Pruning results for MobileNet-V2 using MORL with three objectives. Each dark blue circle represents the optimal solution found in each experiment in the objective space. The yellow–orange plane is interpolated from these optimal points. (a) Nineteen pruning results which searched by the MORL agent taking Accuracy–FLOPs–Latency as the objectives and their interpolated planes. (b) Eighteen pruning results which searched by the MORL agent taking Accuracy–Memory–Latency as the objectives and their interpolated planes. (c) Nineteen pruning results which searched by the MORL agent taking Accuracy–FLOPs–Memory as the objectives and their interpolated planes.

A. Multi-Objective Analysis

On the CIFAR-10 dataset, we evaluate our proposed MORL pruning method, including the pruning results of multi-objective methods with two objectives, and three objectives on one NVIDIA GeForce GTX 1080 Ti.

1) Two Objectives: We pruned MobileNet-V1 net employing an MORL agent with two conflicting objectives on one NVIDIA GeForce GTX 1080 Ti, each searching for 1500 episodes. The results are shown in Fig. 3. The results of the experiment are presented in Fig. 3. The blue bars in the figure show that the accuracy of all four pruning methods is nearly identical. However, the orange bars indicate significant differences in hardware indicators' performance. The left subfigure shows that the MORL agent using the Accuracy–Memory objective outperforms the other methods under the memory hardware indicator. Similarly, in the middle and right subfigures, it can be observed that the pruning results of the MORL agents using latency and FLOPs objectives, respectively, are significantly better than those obtained using other methods on their corresponding optimization objectives. These observations suggest that the MORL pruning method proposed in this study effectively optimizes the selected targets under conflicting optimization objectives and yields better results than the single-objective pruning method (AMC).

2) Pruned Results: We are not only concerned with the accuracy and hardware indicators of pruned models but also with the network structures obtained under different objective settings. We conduct a detailed analysis of the pruning strategies obtained under three different objective settings. As depicted in Fig. 4, the pruning strategy given by the Accuracy–Memory objectives agent significantly compresses the shallow layers of the network, while the deep layers have relatively less compression. This aligns with our understanding that the first few layers of the network generate larger feature maps and consume more memory, thus compressing them results in significant memory reduction. For both the Accuracy–Latency and Accuracy–FLOPs objectives agents, the pruning strategies exhibit a similar trend in which the shallow layers of the network undergo less compression, while the deep layers experience more compression. This is reasonable, because the early layers extract more features, whereas the deeper layers tend to have more redundancy. However, the distinction between the two settings is that the Accuracy–FLOPs setting also compresses the shallow layers to achieve higher FLOPs compression. It can be observed that under the search settings with two objectives, the pruning

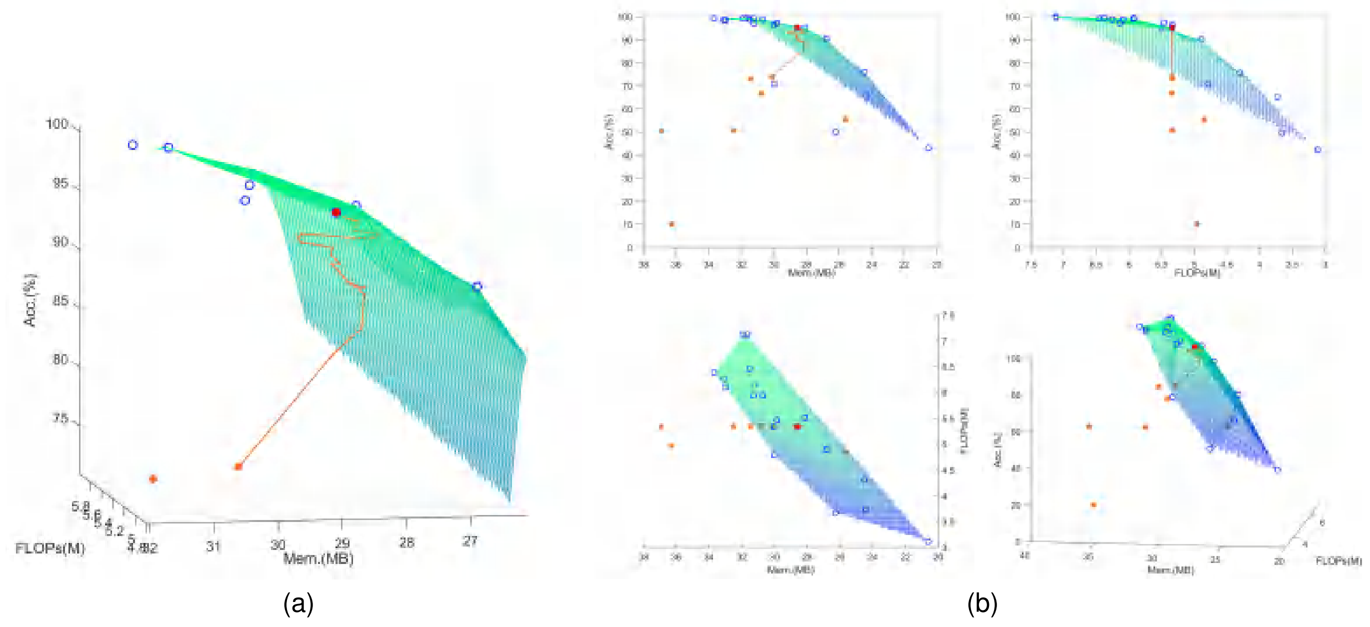

Fig. 6. Search path of an optimal point under the objective setting of Accuracy–FLOPs–Memory. In this search, the total compression ratio is set to pruning 55% FLOPs off, the preference vector $\boldsymbol{\omega}_0 = [0.6, 0.2, 0.2]$, 250 episodes are warmed up, and a total of 1500 episodes are searched. During the search, every occurrence of a solution with a greater reward is recorded and marked in orange in the figure. The scattered orange solid points represent the solutions obtained under the random pruning strategy in the warm-up phase. The orange broken line represents the search path of the MORL agent after the warm-up phase, the red point represents the optimal point finally found in this search, and the blue hollow points are the optimal points obtained under other pruning ratios and preference vectors. (a) Search path after the warm-up phase, and the optimal point. (b) Search path under the objective space and its three views.

strategies obtained under different objectives intuitively adapt to their respective goals, resulting in good performance. This pattern continues when the number of objectives expands to three or four.

3) Three Objectives: We pruned MobileNet-V2 network on one NVIDIA GeForce GTX 1080 Ti, employing an MORL agent with three objectives including accuracy and other two hardware indicators.

We conduct a total of 18 to 19 experiments for each multi-objective setting, with each experiment using a different pruning ratio and preference vector and searching for 1500 episodes. The results of these experiments are plotted in the objective space and interpolated to form a plane. It can be seen that the interpolation planes in Fig. 5 are convex, which conforms to the nature of the CCS of the Pareto frontier, confirming that our proposed MORL method produced search results on a convex solution plane. This plane can serve as an approximation of the CCS.

For any optimal point obtained by MORL agent search, taking the point in Fig. 5(c) as an example, we evaluate its search path during the process, as shown in Fig. 6. Fig. 6 shows the search path taken by the MORL agent to find an optimal solution while considering accuracy, FLOPs, and memory as objectives. The search was conducted with a total compression ratio of 55% FLOPs, using a preference vector $\boldsymbol{\omega}_0$ of $[0.6, 0.2, 0.2]$. The warm-up phase consisted of 250 episodes, while the full search involved 1500 episodes.

Fig. 6(b) depicts the search path and the three views of the objective space. During the warm-up phase, the random pruning strategy resulted in widely scattered and trendless solutions (the orange solid points), which were far from the CCS plane approximation (the green–blue plane). Subsequently, in Fig. 6(a), the search path (the orange broken line) shows that the solutions generated by the MORL agent quickly approached the approximate CCS plane. As the solutions approached the plane, the search path began to zigzag until it reached the optimal solution (represented by the red solid point).

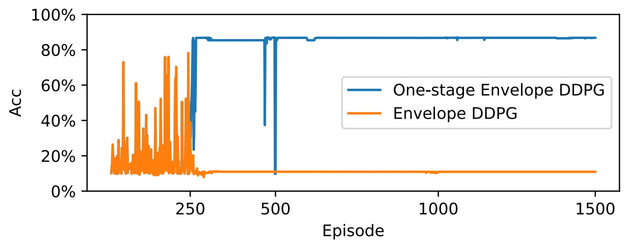

Fig. 7. Accuracy curves of the one-stage envelope DDPG and envelope DDPG algorithms when searching for 1500 episodes.

The three views in Fig. 6(b) provide a more intuitive analysis of the search path. Since the compression ratio was set to prune 55% of FLOPs, the value of FLOPs remained fixed after the warm-up phase. In the top view, it can be seen that memory decreased gradually with each search, while the left view shows that accuracy increased with each search.

B. Search Times

In addition, we validate the efficiency of our proposed one-stage envelope DDPG algorithm. In the scenario with four objectives searching, we conducted searches using both the one-stage envelope DDPG and the original algorithm for 1500 episodes each on MobileNet-V1. Since the rewards in MORL are vectors, we compare one of its components, the accuracy curve. As depicted in Fig. 7, during the warm-up

TABLE I

Pruning Results of VGG-16 on CIFAR-10 With Four Objectives Searching on NVIDIA GeForce GTX 1080 Ti. * Indicates the Experimental Results We Reproduced Using the Open Source Code. MO Indicates the Pruning Methods for Multi-Objective Optimization

Model	Policy	MO	FLOPs(M)	Lat.(ms)	Mem.(MB)	Acc.(%)	ΔAcc.(%)
VGG16	Original	-	313.20	4.6	111.5	92.64	-
	AMC*(0.5 FLOPs) [14]	✗	157.07	2.5	89.4	92.80 → 91.90	-0.90
	DPF* [51]	✗	175.32	2.8	87.8	93.74 → 93.88	+0.14
	CURATING [52]	✓	157.29	-	-	93.25 → 93.57	+0.32
	DECORE-500 [15]	✗	203.08	-	-	93.96 → 94.02	+0.06
	RL-MCTS [16]	✗	170.70	-	-	93.51 → 93.90	+0.39
	GAL-0.05 [36]	✗	189.49	-	-	93.96 → 93.77	-0.19
	GAL-0.1 [36]	✗	171.89	-	-	93.96 → 93.42	-0.54
	Ours	✓	**154.97**	**1.9**	**85.8**	**92.64 → 93.77**	**+1.13**
	AMC*(0.2 FLOPs) [14]	✗	62.82	2.4	56.2	92.64 → 92.52	-0.12
	DPF* [51]	✗	71.19	2.5	55.8	93.74 → 92.62	-1.12
	DECORE-200 [15]	✗	110.51	-	-	93.96 → 93.56	-0.40
	DECORE-100 [15]	✗	51.20	-	-	93.96 → 92.44	-1.52
	Ours	✓	**63.17**	**2.4**	**48.2**	**92.64 → 92.73**	**+0.09**

TABLE II

Pruning Results of ResNet-56 and ResNet-50 on CIFAR-10 With Four Objectives Searching on NVIDIA GeForce GTX 1080 Ti

Model	Policy	FLOPs(M)	Lat.(ms)	Mem.(MB)	Acc.(%)	ΔAcc.(%)
ResNet-56	Original	125.49	14.4	203.1	93.39	-
	AMC*(0.75 FLOPs) [14]	94.23	9.3	184.2	93.39 → 93.55	+0.16
	DPF* [51]	93.94	9.5	176.8	94.02 → 92.29	-1.73
	DECORE-450 [15]	92.48	-	-	93.26 → 93.34	+0.08
	Ours	**88.33**	**8.7**	**166.6**	**93.39 → 93.69**	**+0.30**
	AMC*(0.55 FLOPs) [14]	70.34	10.4	157.0	93.39 → 93.40	+0.01
	DPF* [51]	76.64	9.6	156.7	94.02 → 90.62	-0.40
	DECORE-200 [15]	62.93	-	-	93.26 → 93.26	0.00
	Ours	**63.56**	**8.3**	**155.8**	**93.39 → 93.40**	**+0.01**
ResNet-50	Original	1297.83	11.4	1309.4	93.62	-
	AMC*(0.5 FLOPs) [14]	649.17	6.7	1132.8	93.62 → 93.74	+0.12
	Ours	**647.36**	**6.4**	**1056.8**	**93.62 → 93.99**	**+0.37**
	AMC*(0.2 FLOPs) [14]	259.57	8.2	693.5	93.62 → 93.47	-0.15
	Ours	**259.69**	**7.3**	**621.7**	**93.62 → 93.59**	**-0.03**

phase of the first 250 episodes, the accuracy curves of both algorithms overlap. However, after 250 episodes, the one-stage envelope DDPG starts regular search and pushes the accuracy to around 86%, while the original algorithm struggles to converge, with accuracy hovering around 10%. Even after 1500 episodes, the original algorithm fails to converge. This illustrates that our method significantly outperforms the original method in terms of convergence speed. This is because our algorithm combines the learning phase and adaptation phase. As shown in (15), our algorithm updates the actor network using the user-defined preference vector $\boldsymbol{\omega}_0$, which makes the actor network more inclined to generate targeted actions. This efficiency enhancement reduces the number of episodes needed for exploration.

C. CIFAR-10

On CIFAR-10 datasets, experiments were conducted on an NVIDIA GeForce GTX 1080 Ti to evaluate the performance of the proposed approach on various networks, including MobileNet-V1/V2 [28], [29], VGG-16 [26], and ResNet-56/50 [27]. The MORL agent assessed four search objectives, namely, accuracy, FLOPs, latency, and memory in each experiment, which searched 1500 episodes and fine-tuned within 200 epochs.

1) VGG: Table I shows the pruning results of VGG-16 on CIFAR-10 with four objectives searching. For VGG-16, we conducted experiments with pruning 50% FLOPs off and pruning 80% FLOPs off. Compared with the baseline, our method at the compression ratio of 50% FLOPs achieved 1.30× memory compression and 2.42× acceleration with an accuracy improvement of 1.13%. Our method yields better results than the manually designed method dynamic pruning with feedback (DPF) [51] across all four metrics. Furthermore, our multi-objective method outperformed the single-objective reinforcement learning method AMC [14] in all four indicators, with higher accuracy, 2.1M fewer FLOPs, 3.6-MB less memory, and 0.6-ms faster latency. As compared with other reinforcement learning algorithms, such as DECORE [15] and RL-MCTS [16], as well as generative adversarial network (GAN) algorithms, such as GAL [36], our method achieved higher compression with improved accuracy. Compared with the multi-objective method CURATING [52], our approach also outperforms it on multiple metrics. At the compression ratio of pruning 80% FLOPs off, compared with the baseline,

TABLE III
PRUNING RESULTS OF MOBILENET-V1 AND MOBILENET-V2 ON CIFAR-10 WITH FOUR OBJECTIVES SEARCHING ON NVIDIA GEFORCE GTX 1080 Ti

Model	Policy	FLOPs (M)	Lat. (ms)	Mem. (MB)	Acc. (%)
MobileNet-V1	Original	11.6	7.2	39.2	86.49
	AMC* [14]	5.8	4.2	32.2	86.43
	uniform(0.75)	6.6	5.1	29.4	86.13
	Ours	**3.6**	**3.2**	**25.8**	**86.53**
MobileNet-V2	Original	6.1	11.4	52.2	85.64
	AMC* [14]	3.1	5.8	35.1	85.06
	uniform(0.75)	4.1	8.2	40.4	84.88
	Ours	**3.1**	**4.9**	**33.6**	**86.31**

our method achieved a 2.31× decrease in memory usage, and a 1.92× increase in acceleration, with a corresponding accuracy improvement of 0.09%. Compared with the single-objective approach AMC [14], our method achieves better accuracy and requires 8.0-MB less memory, while achieving similar FLOPs compression (4.96× versus 4.98×) and acceleration (2.4 ms versus 2.4 ms). Experiments show that our method has stable optimization of the four indicators under different pruning ratios, although the network's performance is affected when more FLOPs are cut.

2) ResNet: Table II shows the pruning results of ResNet-50 and the smaller variant ResNet-56 on CIFAR-10 datasets utilizing the MORL agent with four objectives.

We performed experiments on ResNet-50 to prune FLOPs 50% and 80% off. Our method achieved a compression ratio of 50% FLOPs, resulting in a 1.24× compression of memory, and a 1.78× acceleration, while also improving accuracy by 1.37% compared with the baseline. In addition, with comparable FLOPs compression, our approach surpassed the single-objective approach AMC [14] in accuracy, using 76.0-MB less memory and achieving a faster latency by 0.3 ms. Besides, our method is able to achieve a compression ratio of 5.00× FLOPs by pruning 80% of them, resulting in a 2.11× compression of memory and a 1.56× acceleration. Our approach achieved higher accuracy than the single-objective approach AMC. Specifically, our approach uses 71.8-MB less memory and achieves a latency that is 0.9 ms faster while providing similar FLOPs compression with AMC.

We applied compression rates of 75% and 55% to the ResNet-56 network in Table II. When using a compression rate of 75%, 30% of the network's FLOPs were eliminated, resulting in a 0.30% increase in accuracy. In addition, we observed a 1.22× reduction in memory usage and a 1.66× speedup. Similarly, with a compression rate of 55%, 50% of FLOPs were eliminated, and we observed a 0.01% increase in accuracy, a 1.3× reduction in memory usage, and a 1.73× speedup. It is noteworthy that the network was further compressed once it reached the compression rate, as the incentive for FLOPs reduction remained when the rest of the indicators were not affected. Compared with the manually designed method DPF [51], our approach performs better on all four indicators. We also have comparisons at two compression ratios using single-objective reinforcement learning methods AMC [14] and DECORE [15]. Our results demonstrate that our approach achieves superior delta accuracy (+0.30% versus +0.08% and +0.01% versus 0.00%) compared with the DECORE method at lower (88.33M versus 92.48M) or similar (63.56M versus 62.93M) FLOPs. In addition, our method performs equivalently or better than the AMC approach with respect to accuracy and outperforms it in all three hardware indicators.

3) MobileNet: Table III shows the pruning results of MobileNet-V1 and MobileNet-V2 using the MORL method with four objectives on the CIFAR-10 dataset.

For the MobileNet-V1 network, when we set 50% FLOPs pruning, our method achieves 69% FLOPs pruning with a memory compression of 1.52× and acceleration of 2.25× while maintaining an accuracy that is 0.04% higher. Across all four indicators, our method outperforms handcrafted pruning with a uniform rate and the single-objective reinforcement learning method AMC. Specifically, when compared with the AMC, which reduces 50% of FLOPs, our method outperforms it with a 0.10% accuracy increase, a 1-ms faster acceleration, and a 6.4-MB reduction in memory.

When applying a 50% FLOPs reduction to the MobileNet-V2 network, our proposed method achieves a memory compression of 1.55× and acceleration of 2.33×, with a corresponding 0.75% increase in accuracy. Compared with the handcrafted pruning method with a uniform rate, our method outperforms it across all four indicators. Moreover, our method outperforms the AMC, which also cuts 50% FLOPs, by increasing accuracy by 1.25%, accelerating latency by 0.9 ms, and reducing memory by 1.55 MB. These results demonstrate that our approach can effectively search for a pruning strategy that achieves superior performance across all evaluation metrics.

D. ImageNet

Table IV shows the pruning results of MobileNet-V1 and MobileNet-V2 utilizing our approach on the ImageNet ILSVRC2012 dataset using two NVIDIA GeForce RTX 2080 Ti. In each experiment, the reinforcement learning agent has four objectives and searches for 1500 sets, fine-tuned within 200 epochs.

Our method achieved a 50% reduction in FLOPs for MobileNet-V1, resulting in a 4.7× faster speed and 1.48× memory compression while maintaining the same accuracy as the baseline. Furthermore, compared with the single-objective reinforcement learning method AMC, which also reduces FLOPs by 50%, our proposed approach achieved a higher top-1 accuracy by 0.5% and a higher top-5 accuracy by 0.6%, reduced memory usage by 75.21 MB, and had a faster speedup of 12.1 ms.

For MobileNet-V2, our method reduced FLOPs by 50%, resulting in 2.6× faster speed and 1.48× memory compression compared with the baseline. Furthermore, our proposed method outperforms the single-objective reinforcement learning method AMC, which also reduces FLOPs by 50%. In comparison, our method improves top-1 accuracy by 0.5%, and top-5 accuracy by 0.3% reduces memory by 58.17 MB, and has a faster speedup of 1.7 ms. Compared with the multi-objective pruning method GenExp [21], our approach also

TABLE IV
PRUNING RESULTS OF MOBILENET-V1 AND MOBILENET-V2 ON IMAGENET ILSVRC2012 WITH FOUR OBJECTIVES SEARCHING ON TWO NVIDIA GEFORCE RTX 2080 Ti. MO INDICATES THE PRUNING METHODS FOR MULTI-OBJECTIVE OPTIMIZATION

Model	Policy	MO	FLOPs(M)	Lat.(ms)	Mem.(MB)	Top-1 Acc.(%)	ΔTop-1 Acc.(%)	Top-5 Acc.(%)	ΔTop-5 Acc.(%)
MobileNet-V1	Original	-	568.74	21.7	1923.82	70.9	-	89.9	-
	AMC* [14]	✗	287.03	16.7	1372.63	$70.9 \rightarrow 70.5$	-0.4	$89.9 \rightarrow 89.3$	-0.6
	DRL-based [35]	✗	284.37	-	-	$70.9 \rightarrow 70.6$	-0.3	$89.5 \rightarrow 89.6$	+0.1
	Ours	✓	**286.98**	**4.6**	**1297.42**	**$70.9 \rightarrow 71.0$**	**+0.1**	**$89.9 \rightarrow 89.9$**	**0.0**
MobileNet-V2	Original	-	300.78	32.3	2559.65	70.4	-	89.2	-
	AMC* [14]	✗	150.49	14.3	1529.57	$70.4 \rightarrow 69.4$	-1.0	$89.2 \rightarrow 88.3$	-0.9
	Slimmable NN [53]	✗	209.08	-	-	$72.2 \rightarrow 68.9$	-3.3	$90.5 \rightarrow 88.3$	-2.2
	AutoPruner [34]	✗	207.93	-	-	$72.2 \rightarrow 71.2$	-1.0	$90.5 \rightarrow 89.9$	-0.6
	MetaPruning [32]	✗	150.39	-	-	$72.7 \rightarrow 68.2$	-4.5	-	-
	GNN-RL [17]	✗	174.45	-	-	$71.8 \rightarrow 70.0$	-1.8	-	-
	GenExp [21]	✓	150.39	-	-	$71.9 \rightarrow 70.3$	-1.6	$90.5 \rightarrow 89.3$	-1.2
	Ours	✓	**146.97**	**12.6**	**1471.40**	**$70.4 \rightarrow 69.9$**	**-0.5**	**$89.2 \rightarrow 88.6$**	**-0.6**

TABLE V
PRUNING RESULTS OF MOBILENET-V1 AND MOBILENET-V2 ON JETSON XAVIER NX WITH FOUR OBJECTIVES SEARCHING ON CIFAR-10 DATASETS

Model	Policy	FLOPs (M)	Lat. (ms)	Mem. (MB)	Acc. (%)
MobileNet-V1	Original	11.6	40.8	39.2	86.49
	AMC* [14]	5.8	28.1	30.8	86.41
	uniform(0.75)	6.6	30.0	29.4	86.13
	Ours	**3.3**	**25.1**	**23.9**	**86.52**
MobileNet-V2	Original	6.1	68.9	52.2	85.64
	AMC* [14]	3.1	42.2	41.4	85.15
	uniform(0.75)	3.8	56.4	40.4	84.88
	Ours	**3.0**	**40.5**	**31.0**	**85.08**

TABLE VI
PRUNING RESULTS OF VIT-CIFAR ON CIFAR-10 WITH FOUR OBJECTIVES SEARCHING ON NVIDIA GEFORCE RTX 2080 Ti. THE SYMBOL † INDICATES THAT THE VIT-CIFAR MODEL WE EMPLOYED ORIGINATES FROM THE FOLLOWING SOURCE: https://github.com/omihub777/ViT-CIFAR

Model	Policy	FLOPs (M)	Lat. (ms)	Mem. (MB)	Acc. (%)
ViT-CIFAR	Original†	51.6	13.7	772.3	89.52
	AMC* [14]	25.8	13.3	502.4	89.06
	Ours	**25.5**	**12.9**	**496.3**	**89.25**
	AMC* [14]	19.7	15.3	453.0	88.76
	Ours	**19.3**	**12.9**	**438.4**	**88.79**
	AMC* [14]	15.3	14.2	398.6	88.37
	Ours	**15.3**	**12.7**	**397.5**	**88.93**

outperforms it on multiple indicators. In addition, our method has the least decrease in accuracy among other pruning methods, such as AMC [14], Slimmable NN [53], AutoPruner [34], MetaPruning [32], and GNN-RL [17].

E. Edge Device

The JETSON XAVIER NX is equipped with an NVIDIA Xavier system-on-chip (SoC), which contains an eight-core Advanced RISC Machines (ARM) processor, an NVIDIA Volta GPU with 384 compute unified device architecture (CUDA) cores, and dedicated hardware for deep learning acceleration, including tensor cores for performing high-speed matrix operations. It also includes six CPU cores, 6-MB $L2$ + 4-MB $L3$, and 8-GB 128-bit LPDDR4x memory. Table V shows the pruning results of MobileNet-V1 and MobileNet-V2 on JETSON XAVIER NX using the MORL agent with four objectives, which is also obtained on the JETSON XAVIER NX. The pruning process is carried out on the edge device, and the pruned network is subsequently fine-tuned for 200 epochs on NVIDIA GeForce GTX 1080 Ti.

When tested on JETSON XAVIER NX devices, our approach resulted in a 71% reduction of FLOPs for MobileNet-V1, leading to a 1.63× faster speed, 1.64× memory compression, and a slight 0.03% increase in accuracy. In comparison with the handcraft method that uses the uniform pruning rate, our proposed approach achieved a 0.39% increase in accuracy, reduced FLOPs by 3.3M, memory by 5.5 MB, and achieved a speedup of 4.9 ms. Furthermore, compared with the single-objective reinforcement learning method AMC [14], our proposed approach obtained a 0.11% higher accuracy, reduced FLOPs by 2.5M, memory by 6.9 MB, and achieved a faster speedup of 3 ms.

We achieved a 50% reduction in FLOPs for MobileNet-V2 using our approach, resulting in a 1.70× faster speed and 1.68× memory compression, but also a 0.56% decrease in accuracy compared with the baseline. In contrast, when compared with the handcrafted method that uses uniform pruning rates, our approach resulted in a 0.20% higher accuracy, reduced FLOPs by 0.8M, memory by 9.4 MB, and provided a speedup of 15.9 ms. Furthermore, our proposed approach achieved similar accuracy with the single-objective reinforcement learning method AMC [14] (85.08% versus 85.15%), while reducing memory usage by 10.4 MB and achieving a faster speedup of 1.7 ms. The inverted residual module, introduced by MobileNet-V2, must be aligned during the pruning process. This limitation, combined with the fact that MobileNet-V2 is already a lightweight network, results in a loss of accuracy compared with the baseline in the final search outcome, especially on edge devices.

F. Vision Transformer

Table VI shows the pruning results of vision transformer (ViT) [54] utilizing our approach on the CIFAR-10 dataset on an NVIDIA GeForce RTX 2080 Ti. In each experiment, the reinforcement learning agent has four objectives and searches for 1500 sets, fine-tuned with 200 epochs. We conducted a comparative analysis between our approach and the single-objective reinforcement learning method AMC, with the same FLOPs compression ratios. It is evident that our method outperforms AMC in all the metrics, such as latency, memory usage, and accuracy when considering fewer or equivalent FLOPs. This result indicates that our proposed MORL algorithm is not only applicable to convolutional neural networks but also suitable for transformer-based network architectures.

We have conducted theoretical analysis and multiple experiments to evaluate the effectiveness of MCMC. However, there are still pending experiments to be conducted. Nevertheless, our method has limitations, and there are still experiments to be conducted. In the future, these experiments will focus on exploring the application of MCMC in various tasks, including natural language processing.

V. Conclusion

In this article, we propose MCMC, an automated pruning method designed to overcome the challenge of model compression under multiple hardware constraints. Our method employs a one-stage envelope DDPG algorithm to optimize multiple hardware targets while minimizing the effect on accuracy. The one-stage envelope DDPG algorithm updates the policy network by leveraging the convex envelope of the Pareto solution frontier, leading to more efficient solutions. MCMC shows consistent efficiency on a range of hardware devices, including those with limited computing resources, allowing for automated optimization that can accommodate various hardware constraints and features. Our method enables automated MCMC, and it yields substantial improvements across different networks and datasets.

References

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1135–1143.
[2] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.
[3] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2736–2744.
[4] S. Bai, J. Chen, X. Shen, Y. Qian, and Y. Liu, "Unified data-free compression: Pruning and quantization without fine-tuning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023, pp. 5876–5885.
[5] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, Amsterdam, The Netherlands. Cham, Switzerland: Springer, Oct. 2016, pp. 525–542.
[6] J. Chen, L. Liu, Y. Liu, and X. Zeng, "A learning framework for n-bit quantized neural networks toward FPGAs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1067–1081, Mar. 2021.
[7] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," 2016, *arXiv:1605.04711*.
[8] J. Chen, S. Bai, T. Huang, M. Wang, G. Tian, and Y. Liu, "Data-free quantization via mixed-precision compensation without fine-tuning," *Pattern Recognit.*, vol. 143, Nov. 2023, Art. no. 109780.
[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 4114–4122.
[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
[11] Y. Liu, J. Chen, and Y. Liu, "DCCD: Reducing neural network redundancy via distillation," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–12, 2023, doi: 10.1109/TNNLS.2023.3238337.
[12] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," 2016, *arXiv:1612.03928*.
[13] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2N learning: Network to network compression via policy gradient reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1-21.
[14] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 815–832.
[15] M. Alwani, Y. Wang, and V. Madhavan, "DECORE: Deep compression with reinforcement learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 12339–12349.
[16] Z. Wang and C. Li, "Channel pruning via lookahead search guided reinforcement learning," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2022, pp. 2029–2040.
[17] S. Yu, A. Mazaheri, and A. Jannesari, "Topology-aware network pruning using multi-stage graph embedding and reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 25656–25667.
[18] C. Peng, Y. Li, R. Shang, and L. Jiao, "ReCNAS: Resource-constrained neural architecture search based on differentiable annealing and dynamic pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–15, 2022, doi: 10.1109/TNNLS.2022.3192169.
[19] R. Cai and J. Luo, "Multi-task learning for multi-objective evolutionary neural architecture search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 1680–1687.
[20] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (EVVC)*. Cham, Switzerland: Springer, 2020, pp. 35–51.
[21] K. Xu, D. Zhang, J. An, L. Liu, L. Liu, and D. Wang, "GenExp: Multi-objective pruning for deep neural network based on genetic algorithm," *Neurocomputing*, vol. 451, pp. 81–94, Sep. 2021.
[22] L. Hirsch and G. Katz, "Multi-objective pruning of dense neural networks using deep reinforcement learning," *Inf. Sci.*, vol. 610, pp. 381–400, Sep. 2022.
[23] T. Wu, J. Shi, D. Zhou, Y. Lei, and M. Gong, "A multi-objective particle swarm optimization for neural networks pruning," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 570–577.
[24] T. Wu, J. Shi, D. Zhou, X. Zheng, and N. Li, "Evolutionary multi-objective one-shot filter pruning for designing lightweight convolutional neural network," *Sensors*, vol. 21, no. 17, p. 5901, Sep. 2021.
[25] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems*, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2019, pp. 14610–14621.
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[28] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[30] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, 2016.
[31] Z. Zhuang et al., "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 883–894.
[32] Z. Liu et al., "MetaPruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3296–3305.
[33] T.-W. Chin, R. Ding, C. Zhang, and D. Marculescu, "Towards efficient model compression via learned global ranking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1518–1528.
[34] J.-H. Luo and J. Wu, "AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognit.*, vol. 107, Nov. 2020, Art. no. 107461.

[35] J. Chen, S. Chen, and S. J. Pan, "Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 14747–14758.
[36] S. Lin et al., "Towards optimal structured CNN pruning via generative adversarial learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2790–2799.
[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[38] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.
[39] Y. Guo et al., "Towards accurate and compact architectures via neural architecture transformer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6501–6516, Oct. 2022.
[40] Y. Guo et al., "NAT: Neural architecture transformer for accurate and compact architectures," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.
[41] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, "Overcoming multi-model forgetting in one-shot NAS with diversity maximization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7806–7815.
[42] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 7827–7838.
[43] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 3, pp. 385–398, Mar. 2015.
[44] S. Mannor and N. Shimkin, "A geometric approach to multi-criterion reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, pp. 325–360, Dec. 2004.
[45] P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker, "Empirical evaluation methods for multiobjective reinforcement learning algorithms," *Mach. Learn.*, vol. 84, nos. 1–2, pp. 51–80, Jul. 2011.
[46] Y. Zhao, Q. Chen, and W. Hu, "Multi-objective reinforcement learning algorithm for MOSDMP in unknown environment," in *Proc. 8th World Congr. Intell. Control Autom.*, Jul. 2010, pp. 3190–3194.
[47] L. Barrett and S. Narayanan, "Learning all optimal policies with multiple criteria," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, Jul. 2008, pp. 41–47.
[48] C. Shelton, "Balancing multiple sources of reward in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 13, 2000, pp. 1038–1044.
[49] H. Mossalam, Y. M. Assael, D. M. Roijers, and S. Whiteson, "Multi-objective deep reinforcement learning," 2016, *arXiv:1610.02707*.
[50] D. M. Roijers, S. Whiteson, and F. A. Oliehoek, "Computing convex coverage sets for multi-objective coordination graphs," in *Proc. Int. Conf. Algorithmic Decis. Theory*. Cham, Switzerland: Springer, 2013, pp. 309–323.
[51] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," 2020, *arXiv:2006.07253*.
[52] S. Pattanayak, S. Nag, and S. Mittal, "CURATING: A multi-objective based pruning technique for CNNs," *J. Syst. Archit.*, vol. 116, Jun. 2021, Art. no. 102031.
[53] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–12.
[54] A. Dosovitskiy et al., "An image is worth 16 × 16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

Siqi Li received the B.Eng. degree in automation from Xi'an Jiaotong University, Xi'an, China, in 2022. She is currently pursuing the Ph.D. degree with the Institute of Cyber-Systems and Control, Department of Control Science and Engineering, Zhejiang University, Hangzhou, China.

Her research interests include neural network compression and deep learning.

Jun Chen received the B.S. degree from the Department of Mechanical and Electrical Engineering, China Jiliang University, Hangzhou, China, in 2016, and the M.S. degree from Zhejiang University, Hangzhou, China, in 2020, where he is currently pursuing the Ph.D. degree with the Institute of Cyber-Systems and Control, Department of Control Science and Engineering.

His research interests include neural network quantization and deep learning.

Shanqi Liu received the B.S. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2019, where he is currently pursuing the Ph.D. degree with the Institute of Cyber-Systems and Control, Department of Control Science and Engineering.

His research interests include reinforcement learning and large language models.

Chengrui Zhu received the B.Eng. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2022, where he is currently pursuing the M.S. degree with the Institute of Cyber-Systems and Control, Department of Control Science and Engineering.

His research interests include reinforcement learning and intelligent control.

Guanzhong Tian (Member, IEEE) received the B.S. degree from the Harbin Institute of Technology, Harbin, China, in 2010, and the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2021.

He is currently a Research Associate with the Ningbo Innovation Center, Zhejiang University. His research interests include computer vision, model compression, and embedded artificial intelligence (AI).

Yong Liu (Member, IEEE) received the B.S. degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 2001, and the Ph.D. degree in computer science from Zhejiang University in 2007.

He is currently a Professor with the Institute of Cyber-Systems and Control, Department of Control Science and Engineering, Zhejiang University. He has published more than 30 research papers in machine learning, computer vision, information fusion, and robotics. His latest research interests include machine learning, robotics vision, information processing, and granular computing.