

# Compression Space Search: RL-Based Combinational Compression for Neural Networks

Yingtao Shen<sup>1</sup>, Yincheng Ni<sup>1</sup>, Jiace Zhu<sup>1</sup>, Jie Zhao<sup>2</sup>, An Zou<sup>1</sup>

<sup>1</sup>Shanghai Jiao Tong University, <sup>2</sup>Microsoft Corporation  
 {doctorcoal, niyincheng, zhujiace, an.zou}@sjtu.edu.cn, zhaojie@microsoft.com

**Abstract**—The rising demand for lightweight, high-performance models on mobile and embedded platforms has accelerated the development of model compression techniques. Among these, combinational compression methods—which integrate multiple techniques such as pruning and quantization—offer complementary advantages over using individual methods alone. However, existing research typically focuses on specific combinations designed for a particular model architecture or task. These approaches often overlook the need for a general approach capable of identifying the optimal combination strategy, including the selection, sequence, and degree of applying compression methods. In this paper, we formalize the challenge of combining compression methods—specifically their selection, ordering, and compression degree—as a customized Markov Decision Process defined in a configurable compression space. To solve this, we introduce Compression Space Search (CSS), a practical RL-based framework for automatically and efficiently discovering optimal compression strategies. Experiments across CNN and transformer based vision models demonstrate that the proposed CSS achieves a 30 to 101 times reduction in bit operations while maintaining an accuracy drop of no more than 2%.

## I. INTRODUCTION

Deep Learning models have garnered significant interest owing to their broad applicability in various fields. Nonetheless, the implementation of these models on resource-limited systems such as mobile and embedded platforms poses substantial difficulties, primarily due to their significant computational and energy demands [1]. In response to this issue, different neural network compression methods have been devised to reduce the computational expenses associated with these complex models.

These methodologies operate at different stages—either statically before deployment or dynamically during runtime. For example, prior to deployment, techniques such as pruning remove specific neurons from layers and quantization reduces the bit width used in computations [2], [3]. In contrast, dynamic inference strategies—such as early exits and layer skipping—adapt the network structure at runtime based on input characteristics during inference [4].

Several studies have combined multiple compression methods. For example, [5] applies pruning and quantization together to compress CNNs, achieving significant FLOPs reduction with minimal accuracy loss (0.15%–0.37%). More advanced approaches, like APQ [6], introduce pruning and quantization-aware Network Architecture Search (NAS), requiring architecture adjustments to improve parameter efficiency. Cutting-edge methods such as the Deep Hybrid Compression Network [7]

This work was supported in part by NSFC under Grant 62202287. An Zou is the corresponding author.

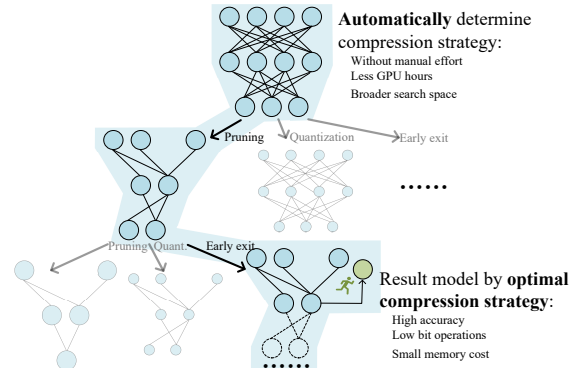


Fig. 1: Automatically determine optimal compression strategy.

integrate quantization, relaxed weight pruning, and knowledge distillation to overcome limitations of uniform quantization.

However, most of the existing work focuses on manually combining two or three compression techniques with a specific neural network. There is a lack of a general approach (as shown in Fig. 1) that can automatically determine the best combination strategy—including which compression methods to use, in which order to apply them, and to what extent. This gap leads to an important and practical research question: **Is it necessary to apply multiple compression techniques, and if so, what is the optimal strategy for doing so?** An optimal strategy should address the following:

- ① **Which compression methods to select;**
- ② **What sequence they should be applied;**
- ③ **What degree each method should be used.**

To identify the optimal compression strategy, we propose Compression Space Search (CSS), a reinforcement learning-based approach that automatically discovers the optimal combination from a given set of compression techniques. We frame this task as a Markov Decision Process specifically designed for iterative and combinational compression. By applying the Dueling Double Deep Q-Netowrk, the proposed CSS can efficiently find the optimal compression strategy.

The contributions and insights of this paper are as follows:

- Through comprehensive empirical studies, we demonstrate that the method, order, and degree of compression operations significantly affect both model accuracy and compression ratio. However, identifying the optimal compression strategy is challenging due to the extensive manual effort and GPU resources required for exhaustive search.
- We address this by formulating combinational compression as a customized Markov Decision Process (MDP). The

formulation features a compact state encoding, a degree searchable discretized action space, and a metric variation based reward function. These design choices enable efficient exploration of the high-dimensional compression space  $(\mathbf{C} \times \mathbb{R})^n$ , while balancing accuracy retention and compression effectiveness.

- We propose Compression Space Search (CSS), a practical, automated approach to discovering optimal compression strategies using a reinforcement learning framework based on Dueling Double Deep Q-Networks (D3QN). CSS incorporates several acceleration techniques, including invalid action masking, adaptive finetuning acceleration, and trajectory memory for efficient state recovery.
- Experiments on both classic CNNs and transformer-based models show that CSS achieves up to  $101\times$  compression in BitOps, while keeping accuracy loss within 2% on Cifar10, Cifar100, Tiny-ImageNet and ImageNet.

## II. BACKGROUND AND RELATED WORKS

### A. Combinational Compression

As model compression research advances, many standalone techniques are reaching fundamental limits. Although they continue to deliver incremental improvements, their effectiveness increasingly relies on specialized hardware implementations [8]. This bottleneck has driven new research directions focused on combinational model compression.

For instance, Han et al. [9] combine pruning, quantization-aware training, and Huffman coding to reduce both the storage requirements of neural network parameters and the energy cost of inference. Similarly, Qi et al. [5] demonstrate that integrating pruning and quantization in CNNs can cut FLOPs by approximately 50% with minimal accuracy loss. Li et al. [10] propose a hybrid approach that combines quantization with early exit mechanisms in CNNs, achieving more than a 50% reduction in computational cost while limiting accuracy degradation to just 1% to 3%. Zhao et al. [7] present the Deep Hybrid Compression Network, which leverages mixed-precision quantization, relaxed weight pruning, and knowledge distillation to overcome the limitations of uniform quantization. Meanwhile, Wang et al. [6] introduce a Quantization and Pruning-Aware Network Architecture Search (NAS) method that modifies specific network structures to achieve high compression ratios without sacrificing inference accuracy.

However, most of these methods are customized for specific models and rely heavily on manual design, which limits their generalizability and makes it difficult to explore longer or more complex compression sequences. Therefore, a general approach needs to be developed that can automatically identify the optimal combination of compression techniques and be applicable to different neural network architectures.

### B. Compression Metrics

Before identifying the optimal compression combination, it is essential to define what constitutes an effective compression process. This work evaluates model compression using the key metrics summarized in Table I. Ideally, effective compression

minimizes accuracy loss while maximizing reductions in computational cost and model size. Therefore, we focus on three performance indicators: accuracy drop ( $Acc_d$ ), bit compression ratio ( $bcr$ ), and memory compression ratio ( $mcr$ ). Since model inference latency depends heavily on hardware and system environment, we use theoretical compression ratios by default to ensure evaluation consistency. The BitOps count method is consistent with the approaches presented in [11] and [12]. As for accuracy, we ensure the model has been properly trained before compression so the accuracy drop is credible.

## III. MOTIVATION AND CHALLENGES

### A. Importance of Combinational Compression Strategy

In this section, we show how the choice of compression strategy affects compression metrics by evaluating a set of simple yet illustrative examples that combine two compression methods selected from three classic techniques: Pruning [13], Quantization [14], and Early Exit [15]. As a representative task, we use ResNet34 [16] on Cifar10 dataset [17] to construct and assess example compression strategies. The construction of example strategies follows the idea of *Order of Compression* [18], an approach that uses brute-force traversal to compress and finetune all possible combinations of method selection, sequence order, and compression degree. As a result, each subfigure in Fig. 2 corresponds to a specific method selection; the differently colored regions in each subfigure represent distinct sequence orders; and each point within a region denotes a compression strategy with a unique combination of compression degrees.

See Fig. 2, **the order of compression strategy** has a significant impact on both BitOpsCR and accuracy. For example, {P,Q} results in a superior accuracy over {Q,P} while {P,E} achieves a higher compression ratio over {E,P}.

Meanwhile, **different compression degrees** also affect compression performance. Appropriate values lead to an approximate Pareto frontier (points on the right and upper boundaries of the colored region), showing the trade-off between BitOpsCR and accuracy. In contrast, improper values reduce both BitOpsCR and accuracy simultaneously.

In addition, our experiments demonstrate that the potential of combinational model compression is much higher than that of a single method. In particular, {P,E}, {Q,E} show BitOpsCR and accuracy benefits that far exceed those of the individual methods that make them up.

In Fig. 2, the compression strategies tested did not involve repeated methods. In Table II we investigate the potential impact of repeating a specific compression technique within a strategy. Intuitively, one might expect minimal difference between directly applying 4-bit quantization {Q(4w,8a)} and sequentially applying 8-bit quantization followed by 4-bit quantization {Q(8w,8a), Q(4w,8a)}. However, **repeating fine-grained compression** can lead to either performance improvement—as seen with successive quantization—or degradation, as in the case of repeated pruning.

### B. Challenges of Combinational Compression Strategy

The above example considers only combinations of two compression methods, and these preliminary experiments do

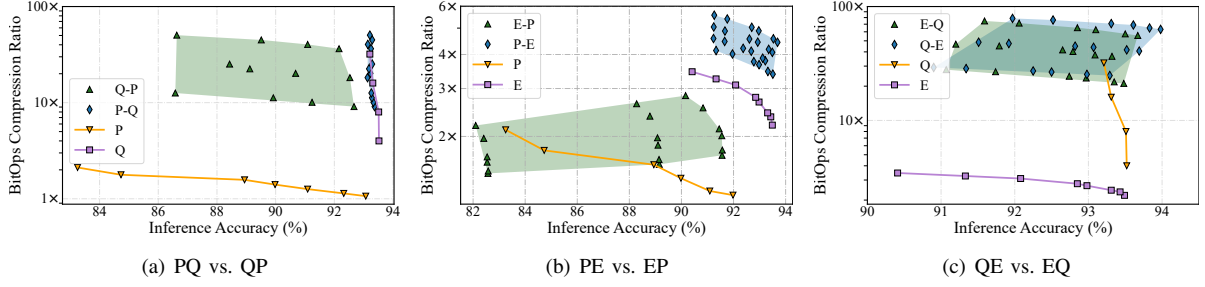


Fig. 2: Significant difference over BitOpsCR-Accuracy trade-off with different compression order.

TABLE I: Model compression metrics and notations.

Metric	Symbol	Description
Original accuracy	$Acc_o$	The original (or base) top-1 inference accuracy before compression.
Current accuracy	$Acc_c$	The current top-1 inference accuracy after compression.
Accuracy drop	$Acc_d$	$Acc_o - Acc_c$ , the inference accuracy loss caused by model compression.
BitOpsCR	$bcr$	Bit operation compression ratio, the reduction in theoretical inference computation.
MemCR	$mcr$	Memory compression ratio, the reduction in model's theoretical memory cost.
CRs	—	Compression ratios, in this work are BitOpsCR and MemCR.

TABLE II: Repeating compressions.

Compression	Acc. Loss (%)	BitOpsCR	MemCR
{P(0.05)}	0.53	1.13	1.10
{P(0.10)}	1.56	1.31	1.21
{P(0.05),P(0.05)}	1.83	1.31	1.20
{Q(8w,8a)} <sup>1</sup>	0.11	16	4
{Q(4w,8a)}	0.57	32	8
{Q(8w,8a),Q(4w,8a)}	0.32	32	8

<sup>1</sup> Q(8w,8a) denotes 8 bit weight and 8 bit activation quantization.

not account for more complex scenarios, such as interleaved applications of multiple techniques. Although we aim to extend this exploration to longer compression strategies with repeated operations, we have already reached the practical limits of *Order of Compression's* brute-force traversal. The primary challenge of manually searching for optimal compression strategies lies in its extremely high computational cost. For example, obtaining the results shown in Fig. 2 required approximately 1000 GPU hours; this number would exceed 20000 GPU hours on the ImageNet dataset. Since a complete finetuning process is required after each compression step, exhaustively evaluating various compression degrees for each strategy is not scalable. As dataset size increases (e.g., from Cifar10 to ImageNet), model complexity grows (e.g., from ResNet to ViT), or sequence length extends, the computational cost becomes prohibitive.

#### IV. MODELING OF COMPRESSION SPACE SEARCH

In order to explore the optimal strategy, including ① **Which compression methods to select;** ② **What sequence they should be applied;** ③ **What degree each method should be used,** we formalize the strategy as an optimization problem. Given a fixed set of compression methods  $\mathbf{C}$ , the goal is to find an optimal **(combinational) compression strategy**  $\{(c_1, d_1), (c_2, d_2), \dots, (c_n, d_n)\}$  with compression method  $c_t \in \mathbf{C}$ , compression degree  $d_t \in \mathbb{R}$  for  $t = 1, 2, \dots, n$ . We model the combinational compression strategy as a Markov Decision Process (MDP) to efficiently explore this vast compression space, as shown in Fig. 3. To accelerate the search and ensure accurate evaluation of the search results, the MDP in this work follows the following design specifications.

- **Condensed state  $s_t$  encoding:** We encode the huge state representing the current model's parameters and architecture with a limited number of important features (observation). Then, following the convention of Partially Observable Markov Decision Process, observation restores the Markov Property by including information of previous state and action.

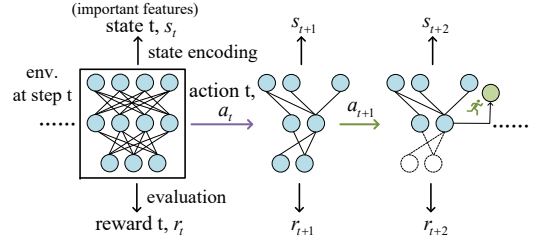


Fig. 3: Modeling Compression Space Search as MDP.

- **Degree searchable discretized action  $a_t$  space:** At each step, the agent selects an action  $a_t$  based on the current policy  $\pi$ . To reduce the dimensionality of the continuous method+degree search space  $(\mathbf{C} \times \mathbb{R})^n$ , we replace each (method  $c$ , degree  $d$ ) action with the accumulation of the same fixed low-degree compression actions, transforming the space into a discrete  $\mathbf{S}^n$ . This allows CSS to maintain degree search capability while simplifying the space into a fine-grained discrete form.
- **Metric variation reward  $r_t$  function:** The reward  $r_t$  is derived from the evaluation of the compressed model  $s_{t+1}$  at next step. As noted in Section II, accuracy drop ( $Acc_d$ ), BitOpsCR ( $bcr$ ) and MemCR ( $mcr$ ) are three important metrics for model compression. Instead of directly obtaining rewards from these metrics at each step (empirical reward function in RL based NAS methods like [19] and [20]), we care more about their change before and after an action. We represent compression metrics through a score function  $\psi_t$  with  $\psi_0 = 0$  and take  $r_t = \psi_t - \psi_{t-1}$  as the reward of each step. The insight of this metric variation reward  $r_t$  is the alignment of compression metrics improvement with MDP policy optimization target:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\pi} \left[ \sum_t \gamma^t r_t \mid s_0 \sim \mu \right], r_t = R(s_t, a_t), \quad (1)$$

In which  $r_t$  at each step finally leads the optimization of  $\sum_t \gamma^t r_t$ . In practice, the step decay factor  $\gamma$  is often close to 1, and the state transition  $\mathbb{P}(s_{t+1} \mid s_t, a_t)$  of model

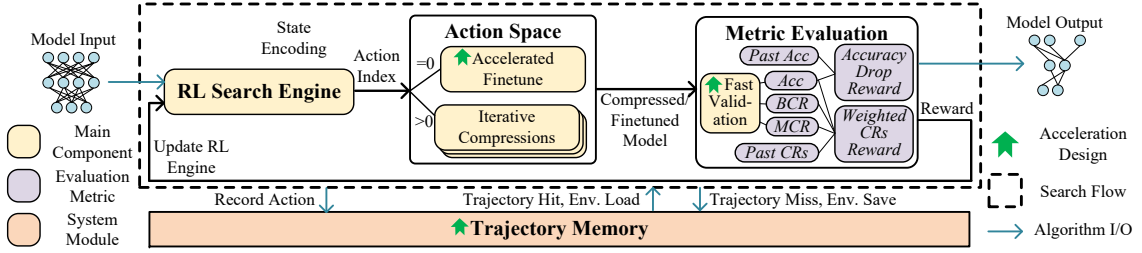


Fig. 4: Overview of Compression Space Search (CSS).

compression is almost deterministic. Therefore, ideally the policy  $\pi^*$  find by RL process will achieve an optimal score function  $\psi_t^*$ :

$$\begin{aligned} \psi_t^* &\approx \max_{\pi} \mathbb{E}_{\pi} \left[ \sum_t (\psi_t - \psi_{t-1}) \mid s_0 \sim \mu \right] \\ &\approx \mathbb{E}_{\pi^*} \left[ \sum_t \gamma^t r_t \mid s_0 \sim \mu \right]. \end{aligned} \quad (2)$$

## V. REINFORCEMENT LEARNING FOR COMPRESSION SPACE SEARCH

Based on the customized Markov Decision Process, the Compression Space Search (CSS) is designed for efficient model exploration, compression, finetuning, and evaluation. As illustrated in Fig. 4, CSS proceeds through a sequence of steps: upon receiving an encoded state, the **Reinforcement Learning (RL) Search Engine** calculates action probabilities of the current policy. The underlying algorithm of the search engine is briefly described in Section V-A. Accepting search engine output, CSS conducts **Action Selection** and executes default finetuning or a compression operation from the defined action space, whose efficiency and configurability are discussed in Section V-B. After executing the action, CSS's **Metric Evaluation** takes the resulting model and returns a reward to the search engine based on compression metrics. Section V-C details how the Accuracy Drop Reward and Weighted Compression Ratios (CRs) Reward help CSS balance accuracy and compression. To further accelerate the search process, CSS employs a **Trajectory Memory** module, which logs each step and allows system restoration across trajectories. This module is covered in Section V-D.

### A. RL Search Engine

Among the various algorithms in reinforcement learning, we use the Dueling Double Deep Q-Network (D3QN) algorithm [21] as our primary approach due to its good adaptability to continuous state and discrete action spaces. Specifically, we utilize a MLP to approximate the value function while integrating a replay buffer with double Q-learning (target network) and advantage function (dueling network) mechanisms. The replay buffer and target network of D3QN can mitigate training fluctuations caused by the high-variance trajectory lengths of the compression strategy, while the dueling network helps to disentangle learning between heterogeneous compression actions. The weight update procedure for D3QN is described as follows:

$$Q(s, a) = V(s) + \left( A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right). \quad (3)$$

In the figure, "Update RL engine" is a simplification of a series of operations on the replay buffer.

### B. Action Selection

Receiving RL Search Engine's output, Action Selection firstly applies Invalid Action Masking. Then an iterative compression action or the default accelerated finetune action will be selected and executed. The CSS's action space is highly flexible for reconfiguration. Incorporating a new compression method into the action space requires only minimal code modifications.

1) *Invalid Action Masking*: CSS implements an Invalid Action Masking mechanism upon RL Search Engine's action probability output. A compression action will be identified as "invalid" and masked when its cumulative degree reaches a predefined limitation. For example, with pruning, masking limits total prune ratio; with quantization, masking limits lowest bit; with early exit, masking limits the number of exit layers to attach. A finetuning action will be masked to skip when it is considered unnecessary at step  $t$ . According to Huang et al. [22], the masking mechanism does not compromise the correctness of gradient updates. In CSS, the masking mechanism plays a role in artificially restricting the search space, further improving the efficiency of the search algorithm.

2) *Iterative Compression*: As mentioned in Section IV, CSS adapts each support compression action to be repeatedly invoked in a single compression strategy. For static compression methods, for example pruning [14], CSS divides their applying into several degree stages, and each action execution will advance the model from one stage to the next, i.e., one pruning ratio to another. Repeating these actions incorporates fine-grained compression degree choices into the search space.

Beyond static methods, CSS also adapts dynamic compression methods to stage-wise execution. Dynamic compression methods are a type of compression that take effects and reduce BitOps during the inference stage. In CSS, each dynamic compression action will deepen its influence on model inference. For example for early exit [15], each action execution will connect an exit layer to a predefined location in the model.

3) *Accelerated Finetune*: In the action space, apart from the compression action, there is a default finetuning action. Instead of always taking adequate finetuning after each compression (as in Section III), CSS has the search engine responsible for considering whether to finetune at step  $t$ . Table III shows the impacts

TABLE III: Ablation study on acceleration designs in Compression Space Search.

Method	CSS	CSS w/o Invalid Action Masking (V-B1)	CSS w/o finetune early stop <sup>1</sup> (V-B3)	CSS w/o Fast Validation (V-C)	CSS w/o Traj. Memory (V-D)	CSS w/o all acceleration
Total GPU hours <sup>2</sup>	<b>29</b>	165	95	38	462	1200
Finetune action hours	<b>26</b>	160	92	26	424	1150
Compression action hours	<b>1</b>	2	1	1	6	50
RL search hours <sup>3</sup>	<b>2</b>	3	2	11	32	0
$bcr_{best}$ within 1.5% $Acc_d$	<b>101</b> ×	101×	102×	101×	103×	105×

<sup>1</sup> Without PlateauLR scheduler and early stop, i.e. apply classic static scheduler with fixed (10) finetune epoch.

<sup>2</sup> The GPU processing time in this work are obtained by running CSS on NVIDIA GeForce RTX 4090.

<sup>3</sup> The "search" encompasses all the time of D3QN algorithm, including model evaluation, reward calculation, state encoding, D3QN updating, and the majority of it is model evaluation.

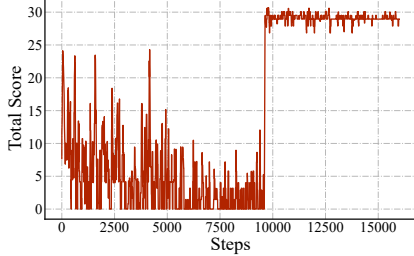


Fig. 5: Total score convergence during CSS application on DeiT-tiny.

TABLE IV: Optimal compression strategies obtained by CSS.

Model	Dataset	Optimal comp. strategy	Acc. (%)	BitOpsCR	MemCR	GPU hours
ResNet34	Cifar10	QQEEEEEEEEPPF	90.88(-1.30)	100.80	28.56	29
	Cifar100	PQQFE	78.07(-1.01)	35.50	8.23	36
	Tiny-ImageNet	QPPPPPPQFPPEPF	65.35(-1.85)	37.87	10.19	140
MobileNetV2	Cifar10	QPPFPFPFPFPFPF	93.35(-0.75)	47.48	10.85	21
	Cifar100	EQQFPF	76.82(-0.89)	35.93	8.17	32
	Tiny-ImageNet	QPPFPFPFPFPFEE	62.78(-1.98)	41.08	9.02	79
DeiT-tiny	Cifar10	QPPPPPPPPQPPF	90.80(-0.10)	41.34	10.06	18
	Cifar100	QQEPPF	69.00(-1.60)	34.19	8.37	25
	Tiny-ImageNet	QEPPQF	56.50(+1.10)	35.06	8.18	46

of CSS’s acceleration designs on the GPU hours consumed by compression, finetuning, and searching of the CSS algorithm when compressing ResNet34 on Cifar10 dataset. It can be seen that finetuning is the critical time-consuming step. By adding it to action space and masking unnecessary finetuning, CSS restricts finetuning frequency without significantly degrading model accuracy. The finetuning masking occurs when  $Acc_d$  (accuracy drop) falls below a certain threshold; meanwhile, to ensure that necessary finetuning takes place during RL’s random early exploration, CSS will force a finetuning action by masking other compression actions when  $Acc_d$  exceeds another certain threshold. In addition to finetune masking, CSS’s finetuning action employs an early stop technique (accuracy drop-based) with PlateauLR scheduler (loss curve-based) aggressively, shortening the finetune process while maintaining training sufficiency. Their acceleration effects are shown in Table III.

### C. Metric Evaluation

After executing  $a_t$  and updating the model, CSS generates reward  $r_t$  through Metric Evaluation. To reduce evaluation GPU hours, Fast Validation estimates model’s compression metrics  $Acc$ ,  $bcr$ , and  $mcr$  within a small (10%), randomly sampled proxy validation set, bringing  $10\times$  faster evaluation with lower than 1%  $Acc$  estimation error on Cifar10, Cifar100 and Tiny-ImageNet. At each step, the reward is the sum of **Accuracy Drop Reward**  $r_{a,t}$  and **Weighted CRs Reward**  $r_{c,t}$ :

$$r_t = \lambda \cdot r_{a,t} + r_{c,t}. \quad (4)$$

$$r_{a,t} \triangleq \psi_{a,t} - \psi_{a,t-1}, r_{c,t} \triangleq \psi_{c,t} - \psi_{c,t-1}. \quad (5)$$

In which scaling factor  $\lambda$  increases from 0.1 to 1 within the first certain steps, enabling a focus on the rise of the compression ratio during the early exploration and gradually elevates the emphasis on inference accuracy as the search progresses.  $r_{a,t}$ ,  $r_{c,t}$  are from the variations of the accuracy score  $\psi_{a,t}$  and the compression score  $\psi_{c,t}$ , respectively. Those score functions are the representation of compression metrics in CSS algorithm:

$$\psi_{a,t} = -w_a \cdot \log_2(Acc_{d,t} + 1) = -w_a \cdot \log_2(Acc_o - Acc_{c,t} + 1). \quad (6)$$

$$\psi_{c,t} = Acc_{c,t} / Acc_o \cdot (w_b \cdot bcr_t + w_m \cdot mcr_t). \quad (7)$$

$w_a$ ,  $w_b$  and  $w_m$  are the weights to balance the  $Acc_c$ ,  $bcr$  and  $mcr$  trade-off. During the search process, once accuracy degrades to an unacceptable level, further variations of  $Acc_c$  become inconsequential. Thus, the design concept of  $\psi_{a,t}$  is the emphasis on initial accuracy drop within a few percentage points from  $Acc_o$ . Through the log function, the accuracy score exhibits a large absolute value of its derivative only when  $Acc_c$  is close to  $Acc_o$ . Consequently,  $\psi_{a,t}$  is sensitive with initial accuracy drop. On the other hand, by multiplying  $Acc_c$ , the compression score  $\psi_{c,t}$  is sensitive to the accuracy collapse. When  $Acc_c$  is virtually undiminished, the increased  $bcr$  and  $mcr$  are directly converted to  $r_c$ . When  $Acc_c$  close to 0, the reward will never increase. In this way, the situation where the agent abandons the inference accuracy in pursuit of higher compression ratios is avoided.

### D. Trajectory Memory

Trajectory Memory stores previously explored models and intermediate values, enabling direct reuse in subsequent searches. A trajectory in a Markov Decision Process, denoted as  $T = \{a_1, a_2, \dots, a_t\}$ , is used as the key for indexing Trajectory Memory. Specifically, two trajectories are considered equivalent if they share the same sequence and types of action.

A trajectory miss occurs when the current action  $a_t$  leads to an unexplored trajectory. If this trajectory is valid (i.e.  $a_t$  is not masked), the current state of the environment—including model parameters, system states, rewards, and other key quantities—is saved. Conversely, a trajectory hit allows Trajectory Memory to directly load these saved quantities into the CSS algorithm, bypassing both the action application and model evaluation processes. Given the current action space’s limited sensitivity to randomness, the acceleration achieved via Trajectory Mem-

ory outweighs the potential drawbacks of omitting stochastic variations (see Table III).

A new trajectory begins when CSS truncates the current one. The truncation occurs when:

- The trajectory exceeds a predefined step limit.
- All compression actions have reached their maximum allowable degree.
- The accuracy drop after a finetuning action, denoted as  $Acc_d$ , surpasses a specified threshold, indicating that further exploration of the current trajectory is unproductive.

By controlling trajectory length and avoiding low-yield explorations, Trajectory Memory can maximize acceleration benefits.

## VI. EVALUATION

### A. Evaluation Setup

**Action Space** The action space we selected to evaluate Compression Space Search (CSS) is: {P (Pruning), Q (Quantization), E (Early Exit), F (Finetune)} as pruning, quantization and early exit are the most common and popular compression methods. This set can be viewed as a basic illustration of utilizing CSS.

**Model and Dataset** We perform an extensive evaluation of end-to-end performance on popular CNN architectures, namely ResNet34 [16] and MobileNetV2 [23] and emerging Vision Transformer architectures, namely DeiT-tiny (train from scratch) [24], across various commonly used benchmark datasets, including Cifar10, Cifar100 [17] and Tiny-ImageNet [25].

### B. Optimal Compression Strategies for CNN and ViT

Table IV summarizes the results of Compression Space Search (CSS) on ResNet34, MobileNetV2 and DeiT-tiny. The "Optimal comp. strategy" column lists the highest-scoring compression strategies discovered by CSS on each dataset, which consistently emerge in the final convergence phase. For example, the optimal strategy {QEPPF} for compressing DeiT-tiny on Cifar100 was found after 10,000 RL training steps, when the reinforcement learning had already converged (as shown in Fig. 5). By trading off accuracy score  $\psi_a$  and compression score  $\psi_c$ , CSS derives compression strategies close to the Pareto frontier. As shown in Table IV, CSS achieves  $30\times$  to  $101\times$  BitOpsCR and  $8\times$  to  $28\times$  MemCR across various models and datasets, while the accuracy drop of compressed models remains within 2%. Notably, for less complex tasks (e.g., Cifar10) and redundancy-rich models (e.g., ResNet34), CSS achieves up to  $101\times$  BitOpsCR. The execution of CSS eliminates the need for manual selection of compression methods, order, and degrees. Crucially, CSS automates the entire process—from model compression, evaluation to strategy optimization—within a short period of GPU hours, significantly facilitating the adaptation of lightweight models to edge applications.

### C. CSS Scalability to Larger Dataset

In addition to fully executing CSS to search for the optimal compression strategy, we also explored Compression Strategy Transfer. Specifically, we apply the optimal strategy {QEPQF}, identified by CSS on DeiT-tiny model using Tiny-ImageNet dataset, to compress DeiT-tiny model pre-trained on ImageNet-1k [40]. The compressed model achieves a  $18.02\times$  BitOpsCR

TABLE V: Comparing CSS with other methods involving multiple compression techniques.

Dataset	Model	Method	Acc. (%)	BitOpsCR	MemCR
Cifar10	ResNet34	Baseline	92.40	1.00	1.00
		Pred-exit [10]	82.00	190.48	-
		CEA-MOP [26]	92.01	1.70	1.62
		LQP [27]	92.43	30.52	22.81
		<b>CSS (Ours)</b>	<b>90.88</b>	<b>100.80</b>	<b>28.56</b>
Cifar100	MobileNetV2	Baseline	94.10	1.00	1.00
		Single-shot [28]	84.35	34.20	3.27
		OED [29]	93.35	1.93	1.25
		CoQuant [30]	94.00	-	8.00
		<b>CSS (Ours)</b>	<b>93.35</b>	<b>47.48</b>	<b>10.85</b>
Cifar100	ResNet34	Baseline	79.08	1.00	1.00
		Pred-exit [10]	60.00	86.49	-
		HMC [31]	67.88	-	5.56
		OICSR-GL [5]	73.10	35.71	-
		<b>CSS (Ours)</b>	<b>78.07</b>	<b>35.50</b>	<b>8.23</b>
ImageNet	DeiT-tiny	Baseline	77.71	1.00	1.00
		Single-shot [28]	56.65	34.20	3.27
		NetsPresso [32]	73.68	1.59	2.88
		OCNNA [33]	74.72	-	4.19
		<b>CSS (Ours)</b>	<b>76.82</b>	<b>35.93</b>	<b>8.17</b>
ImageNet	DeiT-tiny	Baseline	72.20	1.00	1.00
		Bi-ViT [34]	28.70	61.50	22.80
		SViT [35]	70.10	1.31	1.36
		UVC [36]	70.60	2.55	-
		Q-ViT [37]	71.50	11.53	9.91
		PPT [38]	72.10	1.63	1.00
MiniViT [39]	72.80	1.00	1.70		
<b>CSS<sup>†</sup> (Ours)</b>	<b>71.45</b>	<b>18.02</b>	<b>4.07</b>		

<sup>†</sup> represents applying CSS through Compression Strategy Transfer.

and a  $4.07\times$  MemCR with only a 0.75% accuracy loss, demonstrating that CSS-derived strategies can be transferred across similar target models and downstream tasks without re-searching — thereby extending the practical utility of CSS.

In Table V, we compare our proposed CSS with other state-of-the-art model compression techniques. Most are combinational methods that use two or more compression approaches, such as pruning, quantization, early exit, and network architecture search (NAS). Some, like Pred-exit [10], Single-shot [28], and Bi-ViT [34], achieve a similar or higher BitOpsCR than CSS but at the cost of significant accuracy loss. However, when evaluating overall performance in terms of both accuracy and BitOpsCR, CSS outperforms all listed methods.

## VII. CONCLUSION

In this work, we propose Compression Space Search (CSS), a practical reinforcement learning method to automatically discover optimal compression strategies, including selection, order and compression degree, within a predefined compression space  $(C \times \mathbb{R})^n$ . By customizing Markov Decision Processes to effectively represent compression strategies and integrating specialized acceleration mechanisms into the D3QN algorithm, CSS can efficiently explore the compression space while accurately evaluating model's compression metrics after each compression. Evaluated on CNNs and ViTs across different datasets, the CSS achieves a  $30\times$  to  $101\times$  BitOpsCR with  $\leq 2\%$  accuracy drop. Furthermore, we validated the transferability of CSS-derived strategies, showing that a strategy optimized on Tiny-ImageNet can compress ImageNet-1k models with comparable efficacy. CSS's automatic discovery of optimal compression strategies overcomes the limitations of manual heuristic compression design and offers a scalable solution for deploying lightweight models on resource-constrained platforms.

## REFERENCES

- [1] Liu Ke, Xin He, and Xuan Zhang. Nnest: Early-stage design space exploration tool for neural network inference accelerators. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 1–6, 2018.
- [2] Rahul Mishra, Hari Prabhath Gupta, and Tanima Dutta. A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954*, 2020.
- [3] James O’Neill. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.
- [4] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7436–7456, 2021.
- [5] Qi Qi, Yan Lu, Jiashi Li, Jingyu Wang, Haifeng Sun, and Jianxin Liao. Learning low resource consumption cnn through pruning and quantization. *IEEE Transactions on Emerging Topics in Computing*, 10(2):886–903, 2021.
- [6] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2078–2087, 2020.
- [7] Zhi Zhao, Yanxin Ma, Ke Xu, and Jianwei Wan. Deep hybrid compression network for lidar point cloud classification and segmentation. *Remote Sensing*, 15(16):4015, 2023.
- [8] Pierre Vilar Dantas, Waldir Sabino da Silva Jr, Lucas Carvalho Cordeiro, and Celso Barbosa Carvalho. A comprehensive review of model compression techniques in machine learning. *Applied Intelligence*, 54(22):11804–11844, 2024.
- [9] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [10] Xiangjie Li, Chenfei Lou, Yuchi Chen, Zhengping Zhu, Yingtao Shen, Yehan Ma, and An Zou. Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8657–8665, 2023.
- [11] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2019.
- [12] Xingchao Liu, Mao Ye, Dengyong Zhou, and Qiang Liu. Post-training quantization with multiple points: Mixed precision without mixed precision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8697–8705, 2021.
- [13] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization, 2021.
- [14] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning, 2020.
- [15] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.
- [18] Yingtao Shen, Mingqing Sun, Jianzhe Lin, Jie Zhao, and An Zou. Order of compression: A systematic and optimal sequence to combinationally compress cnn, 2024.
- [19] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *International conference on machine learning*, pages 25656–25667. PMLR, 2022.
- [20] Manoj Alwani, Yang Wang, and Vashisht Madhavan. Decore: Deep compression with reinforcement learning. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12339–12349, 2022.
- [21] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.
- [22] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.
- [23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [24] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357, July 2021.
- [25] Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.
- [26] Yidan Zhang, Guangjin Wang, Taibo Yang, Tianfeng Pang, Zhenan He, and Jiancheng Lv. Compression of deep neural networks: bridging the gap between conventional-based pruning and evolutionary approach. *Neural Computing and Applications*, 34(19):16493–16514, 2022.
- [27] Yerlan Idelbayev and Miguel A Carreira-Perpinán. More general and effective model compression via an additive combination of compressions. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21*, pages 233–248. Springer, 2021.
- [28] Bofeng Jiang, Jun Chen, and Yong Liu. Single-shot pruning and quantization for hardware-friendly neural network acceleration. *Engineering Applications of Artificial Intelligence*, 126:106816, 2023.
- [29] Zongyue Wang, Shaohui Lin, Jiao Xie, and Yangbin Lin. Pruning blocks for cnn compression and acceleration via online ensemble distillation. *IEEE Access*, 7:175703–175716, 2019.
- [30] Ximeng Sun, Rameswar Panda, Chun-Fu Richard Chen, Naigang Wang, Bowen Pan, Aude Oliva, Rogerio Feris, and Kate Saenko. Improved techniques for quantizing deep networks with adaptive bit-widths. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 957–967, 2024.
- [31] Guoliang Yang, Shuaiying Yu, Hao Yang, Ziling Nie, and Jixiang Wang. Hmc: Hybrid model compression method based on layer sensitivity grouping. *Plos one*, 18(10):e0292517, 2023.
- [32] Nota Inc. Netspresso model compressor model zoo. Accessed: 2025-05-12.
- [33] Luis Balderas, Miguel Lastra, and José M. Benítez. Optimizing convolutional neural network architectures. *Mathematics*, 12(19), 2024.
- [34] Yanjing Li, Sheng Xu, Mingbao Lin, Xianbin Cao, Chuanjian Liu, Xiao Sun, and Baochang Zhang. Bi-vit: Pushing the limit of vision transformer quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 3243–3251, 2024.
- [35] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in Neural Information Processing Systems*, 34:19974–19988, 2021.
- [36] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *ICLR*, 2022.
- [37] Yanjing Li, Sheng Xu, Baochang Zhang, Xianbin Cao, Peng Gao, and Guodong Guo. Q-vit: Accurate and fully quantized low-bit vision transformer. *Advances in neural information processing systems*, 35:34451–34463, 2022.
- [38] Xinjian Wu, Fanhu Zeng, Xiudong Wang, and Xinghao Chen. Ppt: Token pruning and pooling for efficient vision transformers. *arXiv preprint arXiv:2310.01812*, 2023.
- [39] Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12145–12154, June 2022.
- [40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.