Appendix for *EMQ: Evolving Training-free Proxies for Automated Mixed Precision Quantization*

Peijie Dong[1†] Lujun Li[2†] Zimian Wei[1] Xin Niu[1*] Zhiliang Tian[1] Hengyue Pan[1]
[1] National University of Defense Technology, [2] HKUST
[1]{dongpeijie, weizimian16, niuxin, tianzhiliang, hengyuepan}@nudt.edu.cn, [2]lilujunai@gmail.com

In this appendix, we provide additional details and information about EMQ, including related works, the MQ-Bench-101 dataset used for evaluation, computation graph details, and the evolving algorithm. We also provide more information on the routine, Weisfeiler-Lehman test, conflict awareness, naive invalid check, and operation sampling prioritization. Additionally, we analyze the iterations and population of the algorithm and present more ablation studies, including sensitivity analyses of batch size and seeds, as well as primitive operations. The appendix serves as a comprehensive reference for those interested in understanding and implementing the EMQ algorithm.

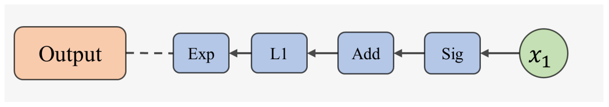
A. Additional Related Works

A.1. Mixed-Precision Quantization.

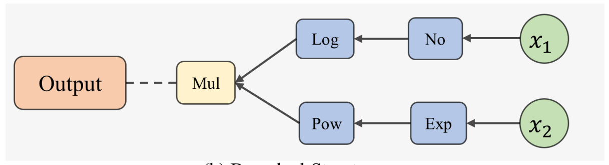
Quantization [24, 9, 27, 3] has been widely investigated as effective techniques [4, 14, 19, 13, 12, 15, 16] to accelerate the inference phase of neural networks by converting 32-bit floating-point weight/activation parameters into low-precision fixed-point values. However, the contribution of each layer to the overall performance is to varying extents, and mixed-precision quantization [35, 21, 37, 38, 6, 5, 2] has been proposed to achieve a better trade-off between accuracy and complexity by assigning different bit-precision to different layers. Existing mixed-precision quantization methods can be classified into four categories: reinforcement learning-based approaches [21, 35, 7], evolutionary algorithm-based approaches [36], one-shot approaches [37, 10, 8] (including differentiable search approaches), and zero-shot approaches [6, 5, 29, 22] (also known as heuristic-based methods). Reinforcement learning-based approaches, such as HAQ [35], use hardware feedback to search the bit-precision in discrete space. Evolutionary algorithm-based approaches, such as APQ [36], jointly search the pruning ratio, the bitwidth, and the architecture of the lightweight model from a hypernet.

However, these search-based methods require an extremely large amount of computational resources and are time-consuming due to the exponential search space. One-shot methods, such as DNAS [37] and Adabits [10], alleviate the searching problem greatly by constructing a supernet or hypernet where each layer consists of a linear combination or parallel blocks of outputs of different bit-precisions, respectively. Nevertheless, a differentiable search for mixed-precision quantization [37, 8] still needs a large amount of time due to the optimization of the large hypernet.

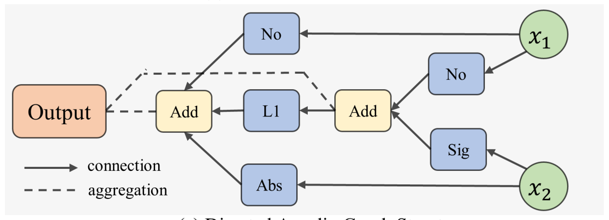
To address the issue of bit-precision selection, heuristic criterion-based methods utilize zero-cost quantization



(a) Sequential Structure



(b) Branched Structure



(c) Directed Acyclic Graph Structure

Figure 7. EMQ Search Space Structures: Three types of computation graph are shown for the EMQ search space. Dotted lines represent aggregation operations, while solid lines represent connections between nodes. (a) Sequential structure performs unary operations in a linear order. (b) Branched structure takes two branch inputs, applies various unary operations, and performs binary operations. (c) Directed Acyclic Graph (DAG) based structure takes two branch inputs and each node aggregates the statistics from all of its predecessors.

*Corresponding author, † equal contribution.

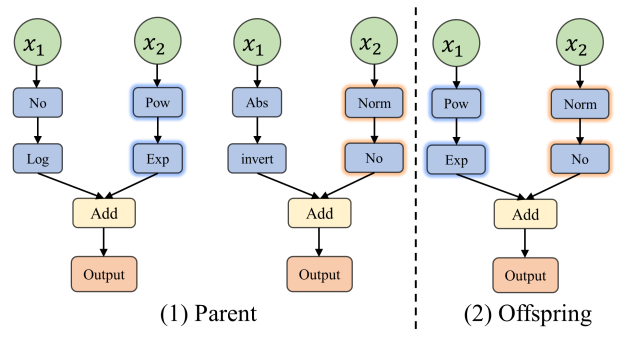


Figure 8. Crossover process for branched structures. A subtree crossover strategy is employed for the two parent structures, where a random subtree is selected from each parent and exchanged to create two new offspring structures. The glowing border indicates the selected branch.

proxies to rank the importance of layers. One approach is the Hessian-based quantization framework, which uses second-order information as the sensitivity metric. For instance, HAWQ [6] measures the sensitivity of each layer using the top Hessian eigenvalue and manually selects the bit-precision based on the relative sensitivity. HAWQ-V2 [5] proves that the average Hessian trace is a better sensitivity metric and proposes a Pareto frontier-based method for automatic bit-precision selection. Intrinsic zero-cost proxies are also developed to handle mixed-precision quantization. QE Score [29] evaluates the entropy of the last output feature map without training, representing the expressiveness. OMPQ [22] proposes an Orthogonality Metric (ORM) that incorporates function orthogonality into neural networks and uses it to find an optimal bit configuration without any searching iterations.

The hand-crafted proxies used in previous works require expert knowledge and are often computationally inefficient [6, 5, 22]. These works suffer from major limitations. First, estimating the average Hessian trace using an implicit iterative approach based on the matrix-free Hutchinson algorithm [1] can lead to computational excesses and unstable iterative results for large-scale models. Second, the automatic bit-precision selection can only yield sub-optimal solutions, as the constraint space of the optimization problem is limited. For example, HAWQ-V2 [5] considers only one constraint on memory footprint when drawing the Pareto frontier of accuracy perturbation and model size, limiting the solutions to local optima in low-dimensional spaces. To overcome these challenges, we commence by benchmarking the existing zero-cost quantization approaches and aim to automate the process of designing zero-cost quantization proxies using techniques inspired by AutoML-zero [28]. Our objective is to automatically search for the most effective training-free quantization proxy, capable of achieving competitive results with hand-crafted solutions.

A.2. Zero-cost Proxies for NAS

Recently, research has been focused on zero-shot/zero-cost neural architecture search (NAS), which estimates the performance of network architectures using zero-cost proxies based on small batches of data. Zero-shot NAS outperforms early NAS since it can estimate model performance without the need for complete training and training of super-networks in a single NAS, and without the need for forward and backward propagation of neural networks, which makes the entire process cost negligible. Zero-shot NAS is classified into two types: architecture-level and parameter-level zero-shot NAS.

(1) Architecture-level zero-shot NAS evaluates the discriminative power of different architectures through inference. For example, NWOT [23] found that better-performing models can better distinguish the local Jacobian values of different images and proposed an indicator based on the correlation of input Jacobian for evaluating model performance. EPE-NAS [20] proposed an index based on the correlation of Jacobian with categories. ZenNAS [17] evaluates the candidate architectures with the gradient norm of the input image as a ranking score. MAE-DET [30] estimates the differential entropy of the last feature map to represent the network's expressiveness based on the maximum entropy theory [26].

(2) Parameter-level zero-cost NAS aims to evaluate and prune redundant parameters from neural networks. Several indicators have been proposed for this purpose, including GradNorm [25], Plain [25], SNIP [11], GraSP [34], Fisher [33], and Synflow [32]. These indicators evaluate the importance of each parameter in the network and rank them based on their values.

While both types aim to alleviate the computational burden of traditional NAS, parameter-level zero-shot NAS has gained more attention due to its similarity with existing MQ proxies. Zero-cost proxies operate at the parameter level and are useful in measuring the sensitivity of each layer in a neural network. Parameter-level zero-cost proxies offer a more fine-grained approach to evaluating the performance of different network architectures, which can be used to optimize the overall performance of the system. Thus, this approach is of great value to the development of efficient and effective neural architectures. Inspired by the existing MQ proxies, we adopt the zero-cost proxies in neural architecture search to measure the sensitivity of each layer by weighting the bit-width.

A.3. Revisit the Zero-cost Proxies for Mixed-precision Quantization

There are four types of input for the zero-cost quantization proxies, which are Hessian, activation, gradient, and weight. The notations are as follows: $\mathcal{L}$ denotes the loss function of a neural network, which measures the discrep-

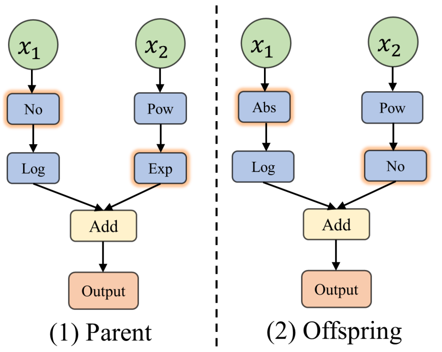


Figure 9. Mutation process for branched structures. Nodes are randomly selected and the corresponding operation would be mutated with random sampled unary operations. The glowing border indicates the selected node.

ancy between the predicted outputs and ground-truth labels. θ represents the parameter of a neural network, which is optimized through back-propagation with the aim of minimizing the loss function. H denotes the Hessian matrix, which describes the curvature of the loss function around a particular parameter configuration. F or z is the activation function, which transforms the input signal into the output signal of a neuron. i is used to denote the i-th layer of a neural network. $||\cdot||_F$ denotes the F-normalization, which normalizes a matrix by its Frobenius norm. Finally, Tr denotes the trace of a matrix, which is the sum of its diagonal elements.

(1) Hessian as Input. Hessian Matrix represents the second-order information. Evidence [6] shows that the eigenvalues of the Hessian can measure the sensitivity of a specific layer of a neural network. The highest Hessian Spectrum is proposed in HAWQ [6] as shown in Equ. 14, with which to decide the order of finetuning blocks.
where H is the Hessian matrix, $\lambda_i(H)$ denotes the ith eigenvalue of H, and n is the dimension of H. The curly braces $\{\cdot\}$ denote a set, and the subscript $i = 1$ indicates that the set starts with the first eigenvalue, while the superscript n indicates that the set ends with the n-th eigenvalue.

$$\text{spectrum}(H) = \max_i \{\lambda_i(H)\} \tag{14}$$

HAWQv2 [5] points out the drawbacks of HAWQ for it only focuses on the top eigenvalue but ignores the rest of the Hessian spectrum. Instead, HAWQv2 adopts the average hessian trace as the MQ proxy as shown in Equ. 15, where n is the number of Hessian matrices being averaged, $\text{tr}(H_i)$ denotes the trace of the ith Hessian matrix, and $\frac{1}{n}\sum_{i=1}^{n}$ represents the average over all n matrices.

$$\text{trace}(H) = \frac{1}{n}\sum_{i=1}^{n} \text{tr}(H_i) \tag{15}$$

(2) Activation as Input. Taking the activation as input, OMPQ [22] proposed the network orthogonality as the zero-cost proxy, which can correlate with the accuracy with different quantization configurations. If a neural network with N layers, the activation from different layers are $\{F\}_i^N$. Then the orthogonality metric is shown in Equ. 16.

$$\text{orm}(F) = \frac{||F_j^T F_i||_F^2}{||F_i^T F_i||_F^2 ||F_j^T F_j||_F^2} \tag{16}$$

QE Score (Quantization Entropy Score) [29] regard neural network as an information system, propose an entropy-driven zero-cost proxy to measure the expressiveness of bit configurations. The equation is shown in Equ. 17,

$$\text{qescore}(F) = \sum_{l=1}^{L} \log\left[\frac{C_l \sigma^2 \sigma_{\text{act}}^2}{\sigma_{\text{act}}^2}\right] + \log(\sigma_{\text{act}}^2) \tag{17}$$

where C_l represents the product of the kernel size K_l and the number of input channels C^{l-1} for layer l. σ^2 represents the variance of the quantization value used to represent the weights. σ_{act}^2 represents the variance of the activation. Fisher [33] proposes a method to quantify the importance of each activation channel in a neural network, which can be used to inform channel pruning. A metric `fisher` is defined as follows:

$$\text{fisher}(z) = \left(\frac{\partial \mathcal{L}}{\partial z} z\right)^2 \tag{18}$$

where $\mathcal{S}_z$ is the saliency activation z.

(3) Gradient as Input. Various pruning-based techniques weight the gradient with activation or weight to measure the sensitivity of a layer. SNIP [11] computes a saliency metric at initialization using a single minibatch of data to approximate the change in loss when a specific parameter is removed. The equation is shown in 19,

$$\text{snip}(\theta) = \left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right| \tag{19}$$

where $\mathcal{L}$ is the loss function of a neural network with parameters θ, and $\odot$ is the Hadamard product. Synflow [32] proposes a modified version of the synaptic saliency scores that avoids layer collapse during parameter pruning. The synflow proxy is computed that is simply the product of all parameters in the network, so no data is needed to compute the metric. Synflow [32] is defined in Equ. 20,

$$\text{synflow}(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta \tag{20}$$

where $\frac{\partial \mathcal{R}}{\partial \theta}$ denotes the gradient of the synaptic flow loss.

(4) Weight as Input: The weight of a neural network is another important input to compute saliency metrics. Plain [25] proposed a method to estimate the importance of each weight in the network, where the score is determined by removing each weight and measuring the change in performance. SNIP [11] extended this idea by computing the saliency metric at initialization using a single minibatch of data to approximate the change in loss when a specific parameter is removed. The saliency score is defined as the absolute value of the product of the gradient and the weight, as shown in Eqn.19. Synflow [32] also employs the weight as input to compute the per-parameter saliency metric, as shown in Eqn.20. The synflow proxy is computed by taking the product of the gradient and weight, and it can avoid layer collapse during parameter pruning.

To conclude, the above studies have revealed that a variety of zero-cost quantization proxies can be constructed based upon the combination of four types of inputs (e.g. activation, gradient, weight, and hessian) and two types of operations (binary and unary). As shown in Fig. 10, we illustrate how naive proxies, including Fisher [18], Plain, SNIP [11], and Synflow [31], and Synflow, represents in our branched search space, where "G" denotes gradient, "W" denotes weight, "Z" denotes activation, and "V" denotes the virtual gradient proposed in Synflow [31]. This motivates the exploration of a larger search space of zero-cost quantization proxies in order to identify those that demonstrate improved performance.

B. MQ-Bench-101 Details

For Quantization-aware training, we model the effect of quantization using simulated quantization operations, which consist of a quantizer followed by a de-quantizer. In our implementation of post-training quantization, we utilize the ImageNet-1k dataset for training and evaluation. We set the bitwidth of activation to 8 and allow the bitwidth of weight to vary within the set $\{2, 3, 4\}$. We randomly sample 425 bit configurations and record their quantization accuracy to build the **MQ-Bench-101** (version 1.0). Moreover, we are actively training all bit configurations with a batch size of 64. All of the experiment of MQ-Bench-101 is computed by running each bit configuration on a single GPU (NVIDIA RTX 3090). Our chosen convolutional neural network is ResNet18, and we apply layer-wise quantization for weights. We perform weight rounding calibration with 20,000 iterations, where the temperature during calibration ranges from 20 to 2. We adopt asymmetry quantization and use mean squared error (MSE) as the scale method. The learning rate is set to 4e-4, and we use a calibration data size of 1,024. We would like to acknowledge OMPQ [22] for the implementation of our post-training quantization, which is based on their official repository.

To ensure the effectiveness of our implementation, we perform various experiments and evaluations. Our results show that our post-training quantization method achieves a significant reduction in memory usage without sacrificing accuracy compared to the original network. We believe our implementation can serve as a valuable reference for future researchers and practitioners working on post-training quantization for convolutional neural networks.

Table 9. The training hyper-parameter settings of post-training quantization in MQ-Bench-101.

Setting	Description
Precision	$\{2, 3, 4\}$
Quantization Scheme	asymmetric
Calibration Data	1,024
Weight-only Quantization	✓
Activation Quantization	8 bit
Layer-wise Quantization	✓
Infrastructure	NVIDIA RTX 3090
Dataset	ImageNet-1k
Network	ResNet18
Learning Rate	4e-4

The usage of API. We provide convenient APIs to access our MQ-Bench-101, which can be easily installed via "pip install -e ." in our EMQ repository. The code snippet of how to use MQ-Bench-101 is given below:

```python
from emq.api import EMQAPI as API
api = API('PTQ-GT.pkl', verbose=False)
# sample random index
rnd_idx = api.random_index()
# query by index
acc = api.query_by_idx(rnd_idx)
# query bit_cfg by index
bit_cfg = api.fix_bit_cfg(rnd_idx)
print(f'The index: {rnd_idx} bit_cfg: {bit_cfg},
    acc: {acc:.4f}')

# sample random bit_cfg
rnd_bit_cfg = api.random_bit_cfg()
# query by bit_cfg
acc = api.query_by_cfg(rnd_bit_cfg)
# query index by bit_cfg
idx = api.get_idx_by_cfg(rnd_bit_cfg)
print(f'The index: {idx} bit_cfg: {rnd_bit_cfg},
    acc: {acc:.4f}')
```

We will release the code and benchmark data file.

C. Computation Graph Details

Computation Graph Structures. The three data structures, Sequential Structure, Branched Structure, and Directed Acyclic Graph Structure, are used to represent the MQ proxies in our work. These proxies are built to search for the optimal neural architecture for a given task without requiring any training.

(1) The sequential structure is an efficient data structure that is commonly used to represent linear computations. It is implemented as a linked list of nodes, with each node representing an operation in the computation graph. This data structure is suitable for modeling simple sequential compu-

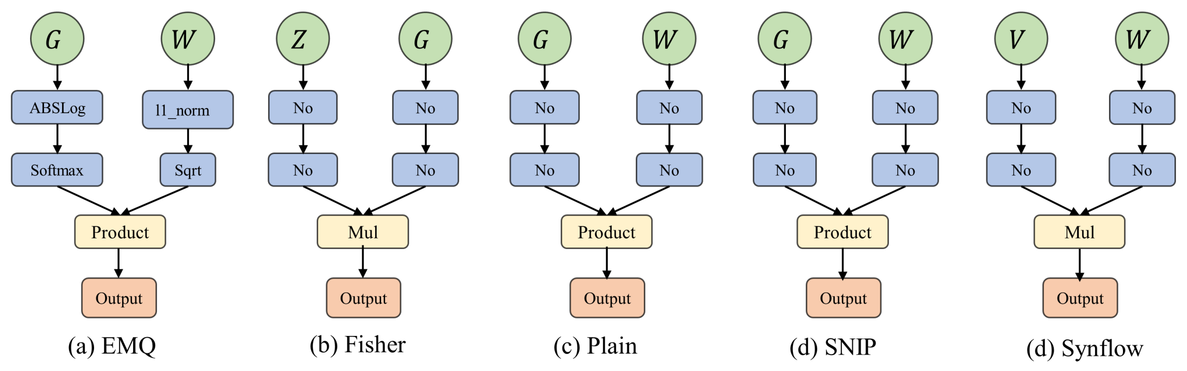


Figure 10. Branched structure of different naive proxies, including the searched EMQ, Fisher [18], Plain [25], SNIP [11], and Synflow [31]. The nodes represent unary operation or binary operation, and the color represents the type of operation performed in the branched structure. The searched EMQ proxy exhibits a more complex and diverse structure compared to other naive proxies, indicating its potential for achieving better compression performance.

tations. As illustrated in Fig. 7 (a), we only take one type of network statistics from gradient, activation, Hessian, and weight, as input. After that, we perform 4 operations sequentially and finally perform aggregation operation, a.k.a. *to_mean_scalar*, to produce the predicted score for the candidate proxy.

(2) The branched structure is a hierarchical data structure that can represent computations with multiple branches. It is implemented as a tree, with each node representing a unary operation in the computation graph and the branches representing the binary operations through the graph. This data structure is more powerful than the sequential structure and can model more complex computations. As demonstrated in Fig. 7(b), the branched structure takes two types of network statistics as input, which can be the same or different. Then, for each branch, we sequentially perform two unary operations to transform the network statistics. After that, we perform the binary operation to take the output of the two branches as input and perform *to_mean_scalar* aggregation function to get the output.

(3) The Directed Acyclic Graph (DAG) structure is a general-purpose data structure that can represent any computation, regardless of its complexity or structure. It is implemented as a directed graph with no directed cycles, which allows for modeling complex dependencies between nodes. This data structure provides the greatest flexibility in modeling complex computations. As shown in Fig. 7(c), we take the same input settings as the branched structure. For each node in the DAG, it would aggregate the information from all of its predecessors and here we adopt binary operation to perform the middle aggregation operation. The Fig. 7 illustrates the computation graph when there are just two middle nodes. We adopt three middle nodes as the default setting, which is more complex than the current computation graph. Finally, we aggregate all of the statistics from the middle nodes to produce the final output by to_mean_scalar.

All three data structures are utilized in constructing the MQ proxies for searching the optimal neural architecture in our work. Nonetheless, considering the superior tradeoff between validity and expressive capability, we primarily adopt the branched structure in this paper.

CrossOver for Structures. To perform crossover for the three computation graph structures, sequential structure, branched structure, and Directed Acyclic Graph structure, we use different strategies. Here we only depict how the branched structure performs crossover in Fig. 8. The probability of performing crossover is set to 50%.

(1) For sequential structure, we adopt a simple crossover strategy. Given two parent structures, we randomly select a crossover point and exchange the remaining nodes to create two new offspring structures.

(2) For the branched structure, we use a branch-switching crossover strategy. Given two parent structures, we randomly select a branch from each parent and exchange them to create two new offspring structures. This strategy allows for exchanging complex branches between parent structures and potentially producing more diverse offspring. As depicted in Fig. 8, the two branches of offspring are selected from one of the branches of the parent structure.

(3) For the Directed Acyclic Graph structure, we adopt a random graph-based crossover strategy. Given two parent structures, we randomly select a subset of nodes from each parent and exchange them to create two new offspring structures. This strategy allows for exchanging nodes with different levels of connectivity and potentially producing more diverse offspring.

Mutation for Computation Graph Structures. Mutation is an important operation in the evolutionary algorithm. For the Sequential and Branched structures, we perform mutation by modifying a randomly selected node, which can be either a layer or a connection between layers. Specifically, we randomly select a node and replace it with a new one sampled from the search space with the probability of 50%, as depicted in Fig. 9. In the Directed Acyclic Graph structure, we perform mutation by modifying the connections between nodes. We randomly select a node and add or delete its incoming or outgoing edges, or we change the type of an existing edge. This type of mutation allows for more complex modifications to the graph structure and enables the model to explore a larger search space.

D. Additional Details in Evolving Algorithm

D.1. Routine

In the context of mixed-precision quantization, proxies play a crucial role in efficiently exploring the search space of possible bit configurations for quantization. To evaluate the quality of different configurations, there are two main routines in the MQ framework: scoring the bit configurations as a whole and evaluating the layer-wise sensitivity separately.

Scoring the bit configurations involves computing a single scalar score for a given quantization configuration, which reflects its overall performance. This routine can be done efficiently without the need for training and evaluation since it only requires analyzing the model's structure and computing the gradient of each layer's output with respect to its input.

On the other hand, evaluating layer-wise sensitivity involves measuring the sensitivity of each layer in a model to quantization. This routine can be computationally expensive since it requires training and evaluating a separate set of quantized models for each configuration, which can be time-consuming for large models.

In this paper, we focus on tackling the former routine of scoring the bit configurations. We propose a novel approach that leverages a proxy to estimate the performance of a given quantization configuration without the need for training and evaluating separate models. Our approach relies on learning a function that maps a quantization configuration to a corresponding performance score. To train this function, we use a set of precomputed scores for a subset of quantization configurations and a gradient-based optimization method to fit the function to the available data. Our proposed method achieves competitive results on benchmark datasets while significantly reducing the computational cost of searching for optimal quantization configurations.

D.2. Weisfeiler-lehman test

The Weisfeiler-Lehman (WL) test is a powerful method for graph isomorphism testing, which can also be used to perform the de-isomorphism process for the computation graph structures.

Table 10. The affection of different initialization seeds on the overall Spearman rank correlation. The rank consistency results are conducted on MQ-Bench-101.

Proxy	Seed				MEAN	STD
	0	1	2	3		
BParam	0.3353	0.6031	0.7082	0.5564	0.5508	0.1360
SNIP	0.3334	0.4603	0.2605	0.4850	0.3848	0.0920
Synflow	0.2841	0.3398	0.2991	0.3398	0.3157	0.0247
HAWQ	0.5821	0.5316	0.7233	0.5821	0.6048	0.0715
HAWQ-V2	0.7813	0.8243	0.6943	0.6903	0.7475	0.0573
OMPQ	0.3295	0.3406	0.3141	0.2585	0.3107	0.0316
QE	0.3280	0.3519	0.3185	0.4616	0.3650	0.0571
EMQ	0.7132	0.7841	0.8712	0.7998	0.7921	0.0561

To apply the WL test, we first initialize each node in the graph with a unique label. Then, for each iteration, we follow these steps: For each node, we collect the labels of its neighbors and sort them. We concatenate the sorted neighbor labels and the node's own label into a new string. We use a hash function to map the new string to a new label for the node. We update the label of each node with its new label. We repeat these steps for a fixed number of iterations, denoted by h. After h iterations, we obtain a new set of labels for all nodes in the graph, which can be used to perform the de-isomorphism process.

D.3. Conflict Awareness

Conflict awareness is a crucial aspect of optimizing search processes. In many cases, different operations that are part of the search space can conflict with each other, leading to unexpected or unstable behavior. In this section, we explore some common examples of conflicting operations and their potential impact on the search process. The potential conflict operation pairs are summarized as follows:

- "log" and "exp": These operations are inverses of each other, so applying them in succession may effectively cancel each other out. This can lead to numerical instability, especially when dealing with small or large values.
- "normalize" and "min_max_normalize": Both of these operations involve scaling the input data to lie within a certain range. However, they use different scaling strategies, which can cause conflicts when applied in succession. For example, normalizing data to have zero mean and unit variance (as in "normalize") may undo the effects of min-max normalization, which scales the data to lie within a specified range.
- "relu" and "sigmoid": These activation functions have different properties and are often used for different purposes. ReLU is commonly used for its simplicity and efficiency in deep neural networks, while sig-

Table 11. The unary operations and binary operations in the search space. “UOP” denotes the unary operations, and “BOP” denotes the binary operations. The type of input and output can be scalar or matrix. “no_op” denotes that we do not perform any operation, and allows for the sparsity of the computation graph. Not all operations below are available or mathematically sound. When there is an illegal operation, we adopt a try-catch mechanism to detect the invalidity and avoid the interruption of the search process.

OP ID	OP Name	Input Args	Output Args	Description
UOP00	`no_op`	–	–	–
UOP01	`element_wise_abs`	a / scalar,matrix	b / scalar,matrix	$x_b = \|x_a\|$
UOP02	`element_wise_tanh`	a / scalar,matrix	b / scalar,matrix	$x_b = \tanh(x_a)$
UOP03	`element_wise_pow`	a / scalar,matrix	b / scalar,matrix	$x_b = x_a^2$
UOP04	`element_wise_exp`	a / scalar,matrix	b / scalar,matrix	$x_b = e^{x_a}$
UOP05	`element_wise_log`	a / scalar,matrix	b / scalar,matrix	$x_b = \ln x_a$
UOP06	`element_wise_relu`	a / scalar,matrix	b / scalar,matrix	$x_b = \max(0, x_a)$
UOP07	`element_wise_leaky_relu`	a / scalar,matrix	b / scalar,matrix	$x_b = \max(0.1x_a, x_a)$
UOP08	`element_wise_swish`	a / scalar,matrix	b / scalar,matrix	$x_b = x_a \times \text{sigmoid}(x_a)$
UOP09	`element_wise_mish`	a / scalar,matrix	b / scalar,matrix	$x_b = x_a \times \text{tanh}(\ln 1 + \text{exp}(x_a))$
UOP10	`element_wise_invert`	a / scalar,matrix	b / scalar,matrix	$x_b = 1/x_a$
UOP11	`element_wise_normalized_sum`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{\sum x_a}{\text{numel}(x_a)+\epsilon}$
UOP12	`normalize`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{x_a - \text{mean}(x_a)}{\text{std}(x_a)}$
UOP13	`sigmoid`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{1}{1+e^{-x_a}}$
UOP14	`logsoftmax`	a / scalar,matrix	b / scalar,matrix	$x_b = \ln \frac{e^{x_a}}{\sum_{i=1}^{n} e^{s_i}}$
UOP15	`softmax`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{e^{x_a}}{\sum_{i=1}^{n} e^{s_i}}$
UOP16	`element_wise_sqrt`	a / scalar,matrix	b / scalar,matrix	$x_b = \sqrt{x_a}$
UOP17	`element_wise_revert`	a / scalar,matrix	b / scalar,matrix	$x_b = -x_a$
UOP18	`frobenius_norm`	a / scalar,matrix	b / scalar,matrix	$x_b = \sqrt{\sum_{i=1}^{n} s_i^2}$
UOP19	`element_wise_abslog`	a / scalar,matrix	b / scalar,matrix	$x_b = \|\ln x_a\|$
UOP20	`l1_norm`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{\sum_{i=1}^{n} \|s_i\|}{\text{numel}(x_a)}$
UOP21	`min_max_normalize`	a / scalar,matrix	b / scalar,matrix	$x_b = \frac{x_a - \min(x_a)}{\max(x_a) - \min(x_a)}$
UOP22	`to_mean_scalar`	a / scalar,matrix	b / scalar	$x_b = \frac{x_a}{n}$
UOP23	`to_std_scalar`	a / scalar,matrix	b / scalar	$x_b = \sqrt{\frac{\sum_{i=1}^{n}(s_i - \bar{s})^2}{n}}$
BOP01	`element_wise_sum`	a,b / scalar,matrixs	c / scalar,matrix	$x_c = x_a + x_b$
BOP02	`element_wise_difference`	a,b / scalar,matrixs	c / scalar,matrix	$x_c = x_a - x_b$
BOP03	`element_wise_product`	a,b / scalar,matrixs	c / scalar,matrix	$x_c = x_a \times x_b$
BOP04	`matrix_multiplication`	a,b / scalar,matrixs	c / scalar,matrix	$x_c = x_a @ x_b$

moid is often used in binary classification tasks. However, applying these functions in succession can lead to non-monotonic behavior, which can cause optimization problems.

- “log” and “softmax”: Both of these operations involve taking the logarithm of the input data. However, the softmax function also involves exponentiation and normalization, which can lead to numerical instability when combined with the logarithm function.
- “pow” and “sqrt”: pow raises a number to a power, while sqrt calculates the square root. Using them together may lead to unexpected results or loss of precision.
- “sigmoid” and “softmax”: these operations are commonly used in neural networks, but applying them together may lead to overfitting or unstable behavior.
- “frobenius_norm” and “revert”: frobenius_norm calculates the Frobenius norm of a matrix (i.e., the square root of the sum of the squared values). Applying revert to a matrix will negate all its values, including the norm.
- “invert” and “revert” operations are also in conflict with themselves. The invert operation involves dividing 1 by the tensor, which can cause numerical instability when the tensor contains values close to 0. The revert operation involves subtracting the tensor from 1, which can also cause numerical instability when the tensor contains values close to 1.

- "abs" and "relu": While these operations are not mathematically inverse, they have similar effects on the input data. Using them together in the same search space may lead to redundant or contradictory combinations.

- "to_mean_scalar" and "to_sum_scalar": In MQ proxy discovery, we do not care for the absolute value of the proxy score but the relative value. To aggregation operations, computing the mean of the value and the sum of it do not influence the ranking ability or correlation of a MQ proxy.

- others: In general, there are conflicts between activation functions, such as "relu", "leaky_relu", "swish" and "mish", and two consecutive activation functions do not need to appear in a computational graph.

Generally, conflict awareness plays a non-trivial role in optimizing search processes. It helps to mitigate numerical instability, improve performance, and ensure the efficiency and effectiveness of the search process. Our analysis of common examples of conflicting operations emphasizes the need to consider the relationships between different operations and their potential impact on the overall search process. By identifying potentially conflicting operation pairs, such as those we have discussed, the EMQ search process can avoid generating invalid combinations of operations and instead focus on discovering high-quality configurations that are both diverse and effective. Ultimately, conflict awareness is a critical component of any search process that aims to produce accurate and reliable results.

D.4. Naive Invalid Check

In the search for an optimal mixed-precision quantization scheme, the validity of the generated proxies is crucial. The Naive Invalid Check technique is a simple yet effective way to reduce the number of invalid proxies generated during the search process. This technique involves checking if the estimated score of a proxy belongs to a set of invalid scores, which includes $\{-1, 1, 0, NaN, and\ Inf\}$. These scores indicate that the proxy is indistinguishable and unreliable, and should be rejected at an early stage.

An estimated score of a MQ proxy in EMQ search space can be invalid due to several reasons. For example, a score of -1 arise from a shape mismatch issue or a user-defined exception, while a score of 1 may indicate a numerical insensitivity. A score of 0 may indicate a numerical instability or the result of an invalid operation. NaN (Not a Number) may arise due to a variety of reasons such as division by zero, square root of a negative number, or logarithm of a non-positive number. Finally, a score of infinity may arise from an overflow in arithmetic operations or the result of invalid mathematical operations. By rejecting these invalid proxies early, the search space is reduced, and the optimization process becomes more efficient. Furthermore, the rejection of invalid proxies reduces the computational cost of evaluating the fitness of the generated proxies, which can be quite expensive. Despite its simplicity, the Naive Invalid Check technique has been shown to be effective in identifying invalid proxies, as the set of invalid scores used in the technique covers a wide range of possible invalid proxy configurations.

D.5. Operation Sampling Prioritization

To mitigate the large number of invalid candidates that result from random operation sampling during the search for MQ proxies, we propose Operation Sampling Prioritization (OSP), which assigns different probabilities to various operations. Specifically, we assign a higher probability to the no op operation for unary operations to promote sparsity in the search space and prevent an excess of operations. For binary operations, we assign a higher probability to the element-wise add operation, as it is the most common operation and unlikely to cause shape-mismatch problems. We also assign probabilities to other binary operations based on their likelihood to cause shape-mismatch problems. Concretely,

- For unary operations: We assign a probability of 0.2 to the "no_op" operation to promote sparsity in the search space and prevent excessive operations. The remaining operations are assigned an equal probability of 0.1.

- For binary operations: We assign a probability of 0.6 to the "sum" operation, which is the most common and least likely to cause shape-mismatch problems. We assign a probability of 0.3 to the "subtract" operation. The "product" and "matrix_multiplication" operations, which are more likely to cause shape-mismatch problems, are assigned a lower probability of 0.05.

D.6. Iterations and Population

In this section, we discuss how we determined the appropriate iteration and population size for the evolutionary algorithm. Both iteration and population size are important parameters that affect the performance of evolutionary algorithms. To determine the iteration, we used a stopping criterion that takes the best Spearman rank coefficient of human-designed MQ proxies [31, 11, 22, 29] as the desired level of correlation. For population size, we found that it mainly affects initialization performance. Thanks to the proposed diversity-prompting selection mechanism, we can maintain population diversity with a small population size of 20. As shown in Fig. 11, the Spearman coefficient for the initialization generation with a population size of 200 is over 0.5, surpassing the one with a population size of 20 by a large margin (10%). However, the population size only

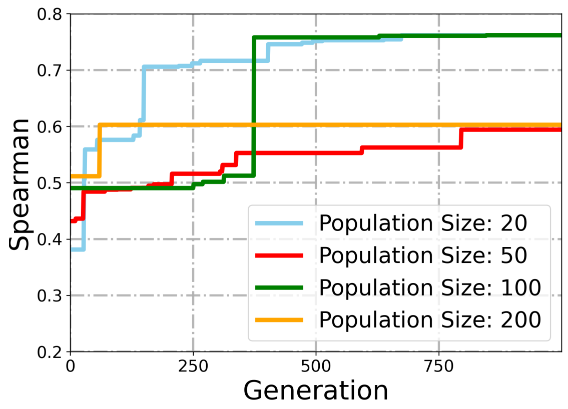


Figure 11. Effect of population size on initialization Spearman correlation. As the population size increases, the initialization Spearman correlation improves.

influences initialization performance and does not affect the final performance.

E. More Ablation Study

In this section, we will perform ablation studies to analyze the sensitivity of the searched EMQ proxy to different settings. This will help us understand the impact of these parameters on the performance of the search process and identify the optimal settings for achieving high-quality results.

E.1. Sensitivity Analysis of Batch Size

We present the sensitivity analysis of the searched EMQ proxy to batch size. We observed that the searched EMQ proxy is completely data-free, and when different batch sizes are used with the same seed, the Spearman rank correlation remains the same. This is intuitive because the searched EMQ takes the gradient of the Synaptic flow loss [31] and the weight, both of which are unrelated to the input. We also noted that when we set "shuffle=False" in the dataloader, the Spearman's rank correlation remains unaffected by the batch size. However, shuffling the mini-batch of data during evaluation can be influenced by the seed, which in turn affects the mini-batch of data. The Fig. 12 investigates the impact of batch size on the Spearman correlation of the searched EMQ proxy. The results reveal that increasing the batch size leads to an improvement in the average Spearman correlation. Moreover, the variance of the Spearman correlation over seven seeds decreases as the batch size increases. Notably, when the batch size is extremely small, the Spearman correlation exhibits a surprisingly good performance. Based on the above observation, we select 64 as the batch size to strike a balance between computational complexity and correlation performance. Specifically, selecting a batch size that is too small may improve correlation performance but will increase computational overhead due to frequent parameter updates, while choosing a batch size that is too large can lead to slower convergence and worse correlation performance. However, under extreme computational constraints, a batch size of 1 can be selected.

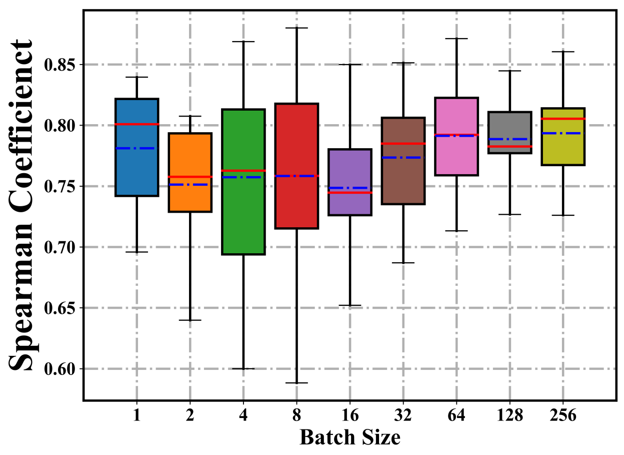


Figure 12. Effect on Spearman rank coefficient of the searched EMQ with respect to batch size. The experiments are done on the MQ-Bench-101 over 7 seeds for 50 bit configurations

E.2. Sensitivity Analysis of Seeds

To assess the influence of the random seed and batch on the searched MQ proxies, we conduct experiments using different seeds and batches of data. This investigation aids in comprehending the extent to which variations in search outcomes can be ascribed to the random seed and batch data. By performing a sensitivity analysis of the seed, we can ensure the robustness of the searched MQ proxy and minimize its susceptibility to the influence of a specific random seed. The seed mainly influences the candidate bit configuration chosen during the evaluating the EMQ proxy. When the seed is fixed, the sampled bit configuration is also fixed, resulting in the same outcomes. As indicated in Tab. 10, we conduct each experiment four times using different seeds and compute the mean and standard deviation of the Spearman rank correlation. Our searched EMQ proxy exhibit the best correlation on MQ-Bench-101, with a similar variance as other unsophisticated proxies. Notably, we observe that when scoring bit configurations as a whole, HAWQ [6] and HAWQ-V2 [5] also achieve competent performance when compared with their counterparts.

E.3. Performance with other metrics

Tab. 12 presents the results of the ranking consistency analysis using Kendall's Tau and Pearson correlation coefficients. The experiments were run five times with different seeds, and the ranking correlation was computed based on 50 bit configurations. The rankings were compared between

Table 12. The ranking consistency with Kendall's Tau and Pearson. Each experiments run 5 times and the ranking correlation is computed based on 50 bit configurations.

	Kendall Tau		Pearson	
	MEAN	STD	MEAN	STD
QE [29]	0.2831	0.0493	0.4131	0.0808
OMPQ [22]	0.1766	0.0249	0.1823	0.0312
HAWQv2 [5]	0.5550	0.0478	0.7213	0.0486
HAWQ [6]	0.4722	0.0006	0.6795	0.0028
Synflow [31]	0.2356	0.0768	0.3633	0.1130
SNIP [11]	0.2724	0.0175	0.2313	0.0227
Bparam	0.3133	0.0893	0.4763	0.1318
EMQ(Ours)	0.6030	0.0373	0.8084	0.0315

the different runs to evaluate the consistency of the results.

Kendall's Tau and Pearson are two widely used correlation coefficients in data analysis. Kendall's Tau is a non-parametric measure of the association between two variables, which means that it does not assume any particular distribution of the data. It measures the similarity of the rankings between two variables, and it ranges between -1 (perfect negative correlation) and 1 (perfect positive correlation). Pearson correlation coefficient, on the other hand, assumes a linear relationship between two variables and measures the strength of this relationship. It ranges between -1 (perfect negative correlation) and 1 (perfect positive correlation).

The results of Kendall's Tau and Pearson correlation coefficients furnish us with a more extensive comprehension of the effectiveness of the hand-crafted MQ proxy and our designed EMQ proxy. It is noteworthy that our EMQ proxy demonstrates the highest correlation in both Kendall's Tau and Pearson. It is also notable that the HAWQ [6] and HAWQv2 [5] exhibit proficient performance.

F. Primitive Operations

The primitive operations used in the search space of EMQ can be classified into two categories: unary and binary operations.

- The unary operations available in the search space include "log", "abslog", "abs", "pow", "exp", "normalize", "relu", "swish", "mish", "leaky_relu", "tanh", "invert", "frobenius_norm", "normalized_sum", "l1_norm", "softmax", "sigmoid", "logsoftmax", "sqrt", "revert", "min_max_normalize", "to_mean_scalar", "to_std_scalar", and "no_op." These operations can be applied to a single input tensor, which may have any number of dimensions.

- The binary operations available in the search space are "sum", "subtract", "multiply", and "dot." These operations can be applied to two input tensors, which may have any number of dimensions, as long as they are compatible for the specified operation.

The possible types of statistics in the computation graph can be scalar, vector o r graph. However, we observe that the computation between the matrix and vector is always mismatched and can not function well. Most of the middle statistics in the computation graph are matrix type, which would cause severe shape mismatch problems and decrease the search process of the EMQ. Empirically, we the operations that can produce the type of vector, for example, "heaviside", "dot", "std", etc.

References

[1] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011. 2

[2] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020. 1

[3] Ting-Wu Chin, I Pierce, Jen Chuang, Vikas Chandra, and Diana Marculescu. One weight bitwidth to rule them all. In *European Conference on Computer Vision*, pages 85–103. Springer, 2020. 1

[4] Peijie Dong, Lujun Li, and Zimian Wei. Diswot: Student architecture search for distillation without training. In *CVPR*, 2023. 1

[5] Zhen Dong, Zhewei Yao, Yaohui Cai, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *arXiv preprint arXiv:1911.03852*, 2019. 1, 2, 3, 9, 10

[6] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302, 2019. 1, 2, 3, 9, 10

[7] Ahmed T. Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. Releq : A reinforcement learning approach for automatic deep quantization of neural networks. *IEEE Micro*, 40:37–45, 2020. 1

[8] Hai Victor Habi, Roy H. Jennings, and Arnon Netzer. Hmq: Hardware friendly mixed precision quantization block for cnns. *ArXiv*, abs/2007.09952, 2020. 1

[9] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018. 1

[10] Qing Jin, Linjie Yang, and Zhenyu A. Liao. Adabits: Neural network quantization with adaptive bit-widths. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2143–2153, 2019. 1

[11] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity. *ArXiv*, abs/1810.02340, 2018. 2, 3, 4, 5, 8, 10

[12] Lujun Li. Self-regulated feature learning via teacher-free feature distillation. In *ECCV*, 2022. 1

[13] Lujun Li, Peijie Dong, Zimian Wei, and Ya Yang. Automated knowledge distillation via monte carlo tree search. In *ICCV*, 2023. 1

[14] Lujun Li and Zhe Jin. Shadow knowledge distillation: Bridging offline and online knowledge transfer. In *NeuIPS*, 2022. 1

[15] Lujun Li, Liang Shiuan-Ni, Ya Yang, and Zhe Jin. Boosting online feature transfer via separable feature fusion. In *IJCNN*, 2022. 1

[16] Lujun Li, Liang Shiuan-Ni, Ya Yang, and Zhe Jin. Teacher-free distillation via regularizing intermediate representation. In *IJCNN*, 2022. 1

[17] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance image recognition. *ICCV*, 2021. 2

[18] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. *arXiv preprint arXiv:2101.09048*, 2021. 4, 5

[19] Xiaolong Liu, Lujun Li, Chao Li, and Anbang Yao. Norm: Knowledge distillation via n-to-one representation matching. In *ICLR*, 2023. 1

[20] Vasco Lopes, Saeid Alirezazadeh, and Luís A. Alexandre. Epe-nas: Efficient performance estimation without training for neural architecture search. In *ICANN*, 2021. 2

[21] Qian Lou, Feng Guo, Lantao Liu, Minje Kim, and Lei Jiang. Autoq: Automated kernel-wise neural network quantization. *arXiv preprint arXiv:1902.05690*, 2019. 1

[22] Yuexiao Ma, Taisong Jin, Xiawu Zheng, Yan Wang, Huixia Li, Guannan Jiang, Wei Zhang, and Rongrong Ji. Ompq: Orthogonal mixed precision quantization. *ArXiv*, abs/2109.07865, 2021. 1, 2, 3, 4, 8, 10

[23] Joseph Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647*, 2020. 2

[24] Nelson Morgan et al. Experimental determination of precision requirements for back-propagation training of artificial neural networks. In *Proc. Second Int'l. Conf. Microelectronics for Neural Networks*, pages 9–16. Citeseer, 1991. 1

[25] Michael C. Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NIPS*, 1988. 2, 3, 5

[26] Kenneth H Norwich. *Information, sensation, and perception*. Academic Press San Diego, 1993. 2

[27] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 580–595, 2018. 1

[28] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, 2020. 2

[29] Zhenhong Sun, Ce Ge, Junyan Wang, Ming Lin, Hesen Chen, Hao Li, and Xiuyu Sun. Entropy-driven mixed-precision quantization for deep network design on iot devices. In *Advances in Neural Information Processing Systems*, 2022. 1, 2, 3, 8, 10

[30] Zhenhong Sun, Ming Lin, Xiuyu Sun, Zhiyu Tan, Hao Li, and Rong Jin. Mae-det: Revisiting maximum entropy principle in zero-shot nas for efficient object detection. In *International Conference on Machine Learning*, 2021. 2

[31] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020. 4, 5, 8, 9, 10

[32] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *ArXiv*, abs/2006.05467, 2020. 2, 3, 4

[33] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *ArXiv*, abs/1801.05787, 2018. 2, 3

[34] Chaoqi Wang, ChaoQi Wang, Guodong Zhang, and Roger Baker Grosse. Picking winning tickets before training by preserving gradient flow. *ArXiv*, abs/2002.07376, 2020. 2

[35] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019. 1

[36] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, and Song Han. Apq: Joint search for nerwork architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 1

[37] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018. 1

[38] Haibao Yu, Qi Han, Jianbo Li, Jianping Shi, Guangliang Cheng, and Bin Fan. Search what you want: Barrier panelty nas for mixed precision quantization. In *European Conference on Computer Vision*, pages 1–16. Springer, 2020. 1