

# RAM-NAS: Resource-aware Multiobjective Neural Architecture Search Method for Robot Vision Tasks

Shouren Mao\*, Minghao Qin\*, Wei Dong, Huajian Liu, Yongzhuo Gao†

\* Joint first authorship; † Corresponding author

**Abstract**—Neural architecture search (NAS) has shown great promise in automatically designing lightweight models. However, conventional approaches are insufficient in training the supernet and pay little attention to actual robot hardware resources. To meet such challenges, we propose RAM-NAS, a resource-aware multi-objective NAS method that focuses on improving the supernet pretrain and resource-awareness on robot hardware devices. We introduce the concept of subnets mutual distillation, which refers to mutually distilling all subnets sampled by the sandwich rule. Additionally, we utilize the Decoupled Knowledge Distillation (DKD) loss to enhance logits distillation performance. To expedite the search process with consideration for hardware resources, we used data from three types of robotic edge hardware to train Latency Surrogate predictors. These predictors facilitated the estimation of hardware inference latency during the search phase, enabling a unified multi-objective evolutionary search to balance model accuracy and latency trade-offs. Our discovered model family, RAM-NAS models, can achieve top-1 accuracy ranging from 76.7% to 81.4% on ImageNet. In addition, the resource-aware multi-objective NAS we employ significantly reduces the model’s inference latency on edge hardware for robots. We conducted experiments on downstream tasks to verify the scalability of our methods. The inference time for detection and segmentation is reduced on all three hardware types compared to MobileNetv3-based methods. Our work fills the gap in NAS for robot hardware resource-aware.

## I. INTRODUCTION

The application of deep learning techniques in robot environmental perception is a current focal point of research. However, their demanding resource requirement in terms of processing time challenges their sustained operation on edge devices embedded within robots [1]. Nevertheless, specific visual perception tasks are driven by deep learning, along with related tasks like object detection [2] and semantic SLAM [3]. Consequently, developing lightweight deep learning models becomes crucial for deploying them on edge computing devices embedded within robots. Model lightweighting strives to reduce model parameters while maintaining performance without significant degradation, thus enhancing computational efficiency. Research [4] suggests that numerous parameters within existing deep neural network models are unnecessary, indicating the potential for model lightweighting. This is particularly crucial for deploying deep learning models on robot platforms. NAS by limiting model complexity, uses reinforcement learning [5] or evolutionary algorithms [6] to automatically pick and adjust key parameters like network layers, width, and depth based on task performance, aiming for the lightweighting model

design. NAS stands out for its automation and innovation, reducing the need for manual design efforts by automatically finding the optimal neural network structure.

Recently, one-shot NAS [7] became popular due to low computation overhead and competitive performance. It does not require training thousands of individual models from scratch but only trains a single supernet that contains any architecture in the search space, weights are shared among subnets respectively. However, when a supernet shares weights across subnets, it might not guarantee adequate training for each specific subnet. This inconsistency in ranking the architecture could diminish the efficacy of the search process [8]. Retrain-free NAS methods [8], [9] propose simultaneously training multiple networks, slicing different-sized subnets directly from the supernet and synchronously training them during the supernet’s training process. In this paper, we present subnets mutual distillation and use DKD loss to enable the knowledge better transfer between subnets, without requiring an external teacher model. Through these strategies, the consistency in ranking individual subnets.

On the other hand, previous NAS [10], [11] mainly abstracts the search problem into a single objective optimization problem under given resource constraints, it demands multiple executions, consuming considerable computing resources and time in the pursuit of a suitable model. Compared to single objective optimization, multiobjective optimization not only allows for simultaneous optimization of multiple given objectives but also yields a Pareto-optimal frontier containing multiple optimal solutions [12]. These solutions exhibit different performances across multiple optimization objectives, thereby facilitating more convenient selection and deployment of different robot computational hardware configurations. Additionally, these methods [13], [14], predominantly consider the computing resources of mobile phones, and few take into account the more frequently used edge hardware of robots. However, as the NAS operating on the server side lacks direct latency aware of hardware, inspired by [12], [15], we propose surrogate predictors to assess the architecture’s latency. In this paper, we conduct a neural architecture search to be aware of the resources of actual robot edge hardware devices with surrogate-assisted.

In summary, to address the limitation of existing NAS methods, which often overlook robot edge device resources and necessitate multiple executions to identify models with varied computing power requirements, we introduce RAM-NAS. The key contributions are summarized below:

- We propose RAM-NAS, a Resource-aware Multiobjective Neural Architecture Search method. This method

All authors are with the State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, China.

allows for a single search to compute the Pareto frontier under various constraints, decision makers can deploy models under different resource constraints.

- We adopt a method where multiple subnets mutually distill each other, enabling joint parameter updates across these subnets. It uses DKD Loss to compute the loss function, ensuring more effective convergence of supernet and boosting the consistency in ranking individual subnets.
- We adopt reusable surrogate predictors to predict hardware latency, the predictor demonstrates high sample efficiency and exhibits strong consistency in predicting latency with actual performance.

## II. RELATED WORK

### A. Neural Architecture Search

Early NAS approaches were highly time-consuming. Recent work [7] has employed the One-Shot Weight Sharing strategy to reduce search costs. Its core idea involves training a supernet where all subnets share weights. However, these methods always require an additional retraining step after determining the optimal architecture. To address this issue, OFA [11] introduced a progressive shrinking algorithm that allows full-sized pre-training of the supernet before subnet sampling. However, optimizing all subnets simultaneously during the training of the supernet is not feasible. OFA samples only about 10–12% of the supernet during training, which is not scalable. Inspired by the concept of slim networks [16], some retrain-free NAS methods propose a novel solution of simultaneously training multiple networks, slicing different-sized subnets directly from the supernet and synchronously training them during the supernet’s training process. For instance, BigNAS [8] and AttentiveNAS [9] adopt a training strategy called the “sandwich rule” during supernet training. This strategy selects the largest subnet, the smallest subnet, and N-2 random subnets to perform forward-backward operations and aggregating gradients before updating model weights. However, it only distills other subnets using the largest subnet, failing to utilize the dark knowledge between subnets fully. This paper introduces the subnet Mutual Distillation strategy, enabling mutual distillation among subnets during the super network training phase and utilizing DKD Loss to compute their loss.

### B. Resource Aware in NAS

Recent works have focused more on resource-aware neural architecture search aimed at maximizing predictive accuracy while minimizing resource demands such as latency, FLOPs, or memory usage. For instance, several algorithms [17] incorporate hardware-agnostic computational costs like FLOPs into the loss function to penalize the network’s resource consumption on target devices. However, FLOPs don’t always reflect actual latency [18]. Therefore, subsequent works use the actual hardware latency of models as the optimization target. There are two ways to obtain actual hardware latency: one involves acquiring real-time running metrics of the target hardware platform during the search process to guide the

search in an online manner, while the other relies on offline perception methods such as proxy models, lookup tables, and more. Online resource-aware method, such as [19], measures inference latency on target hardware by executing models on a mobile device. It integrates this measurement into the optimization objective during the search. However, it requires waiting for feedback on resource metrics during the search process, leading to substantial resource and time consumption. In contrast, feedback obtained from surrogate predictor [12] and lookup tables [10] based on offline perception is real-time and reusable. These predictors and tables can be constructed in advance and do not require fitting during the search process. Therefore, in this work, we fit a surrogate predictor by randomly sampling subnet structures to obtain a real-time and reusable resource proxy model. This accelerates the efficiency of neural architecture search.

## III. METHOD

RAM-NAS consists of four main components: Preliminary, Improving the Supernet Training Strategy, Latency Surrogate Predictor, and Multiobjective Evolution Search under Resource-aware. Fig 1 provides a comprehensive overview of our entire approach.

### A. Preliminary

**Problem Formulation.** The weight-sharing NAS can be represented as a bi-level optimization problem in the following manner:

$$\begin{aligned} \alpha^* &= \arg \max_{\alpha \in \mathcal{A}} \text{Acc}_{\text{val}}(\alpha, \mathbf{w}_\alpha^*), \\ \mathbf{w}_\alpha^* &= \arg \min_{\mathbf{w} \in \mathbf{W}} L_{\text{train}}(\alpha, \mathbf{w}_\alpha), \end{aligned} \quad (1)$$

$\mathcal{A}$  and  $\mathbf{W}$  denote the search space and weight of the supernet. Similarly,  $\alpha$  and  $\mathbf{w}$  denote the architecture and weight of subnets sampled from the supernet.  $\alpha^*$  denotes the optimal architecture in search space  $\mathcal{A}$  and  $\mathbf{w}_\alpha^*$  denotes optimal weight under the candidate architecture  $\alpha$ .  $L_{\text{train}}(\cdot)$  and  $\text{Acc}_{\text{val}}(\cdot)$  represent optimizing the weights of supernet while minimizing supernet loss on the training set and maximizing the accuracy on the validation set.

The aforementioned NAS method solely considers accuracy on the validation set, neglecting the need to account for inference delay during model deployment. Consequently, we formulate the NAS method as a multi-objective optimization problem. The formulation for this process is as follows:

$$F(\alpha) = (f_1(\alpha, \mathbf{w}_\alpha^*), \dots, f_m(\alpha, \mathbf{w}_\alpha^*)), \quad (2)$$

$F(\cdot)$  encompasses  $m$  competing objectives,  $f_i(\cdot)$  denotes the objective function, such as accuracy performance, FLOPs, and latency on specific hardware, etc. In this paper, we focus on two objectives: performance on validation set and latency on robot hardware.

**Search Space.** Consistent with the previous One-Shot NAS method [16], we need to transform the search space into a supernet, that covers all subnets with diverse configurations. To broaden the scope of the search and explore a wider range of possible solutions, we have chosen to adopt a

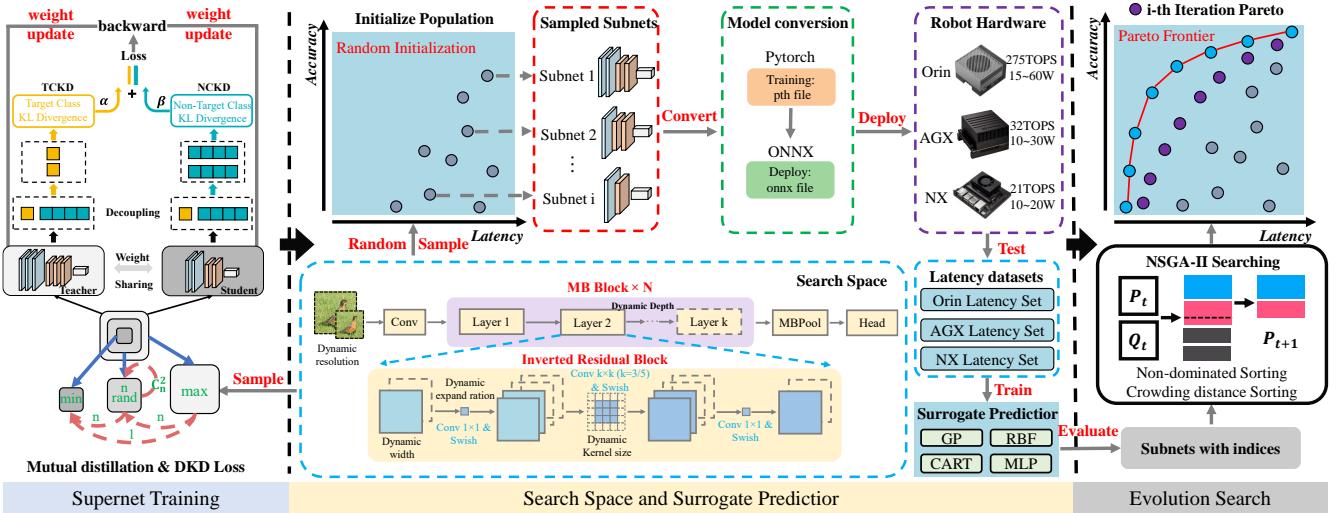


Fig. 1. RAM-NAS employs the Sandwich Rule [8] for subnet sampling, facilitating mutual distillation among diverse subnets and computing their DKD Loss. The supernet is then trained via backpropagation and weight updates. Following this, subnets are randomly sampled from the supernet and deployed on robot hardware for testing. This process generates Surrogate Predictor datasets containing inference latency, which are subsequently utilized to train the Surrogate Predictor. Finally, utilizing the NSGA-II multiobjective optimization algorithm, we define specific visual task accuracy and latency on target hardware as optimization objectives for the searched subnet. Through iterative processes, the Pareto frontier is derived, producing a set of Pareto-optimal solutions. This set includes multiple optimal solutions simultaneously, enabling the generation of precision-optimal models that cater to varying computational latency requirements.

TABLE I

MOBILENETV3-BASED SEARCH SPACE.

Stage	Block	Width	Depth	Kernel size	Stride	Expand ratio	SE
First Conv	Conv	{16, 24}	-	3	2	-	-
Stage1	MBCConv	{16, 24}	{1, 2}	{3, 5}	1	1	N
Stage2	MBCConv	{24, 32}	{3, 4, 5}	{3, 5}	2	{4, 5, 6}	N
Stage3	MBCConv	{32, 40}	{3, 4, 5, 6}	{3, 5}	2	{4, 5, 6}	Y
Stage4	MBCConv	{64, 72}	{3, 4, 5, 6}	{3, 5}	2	{4, 5, 6}	N
Stage5	MBCConv	{112, 120, 128}	{3, 4, 5, 6, 7, 8}	{3, 5}	1	{4, 5, 6}	Y
Stage6	MBCConv	{192, 200, 208, 216}	{3, 4, 5, 6, 7, 8}	{3, 5}	2	6	Y
Stage7	MBCConv	{216, 224}	{1,2}	{3, 5}	1	6	Y
MBPool	Conv	{1792, 1984}	-	1	1	6	-
Input Resolution		{192, 224, 276, 288}					

broader search space built upon MobileNetV3, building upon the previous work [9]. MobileNetV3 is comprised of stacked blocks incorporating Inverted residual blocks, encompassing the five crucial dimensions of the convolutional neural network, namely resolution, width, expansion ratio, depth, and kernel size. Additionally, we have incorporated squeeze-and-excitation modules into select blocks. MBCConv refers to an inverted residual block. SE denotes the squeeze and excites layer. width represents the channels per layer. Depth is the number of repeated MBCConv blocks. Expansion ratio and kernel size denote the expansion ratio for depth-wise convolution layer and filter size. A comprehensive overview of the search space can be found in Table I.

### B. Improving the Supernet Training Strategy

Our supernet is constructed from the maximum value in the search space. Following the previous One-Shot method [7], the subnets need to inherit the weight of the supernet to evaluate its performance to guide the optimal network selection during the search process. Therefore, comprehensive and high-quality training in supernet is very important

for NAS. We solve this problem from two aspects: Subnet Mutual Distillation and Decoupled Knowledge Distillation.

**Subnets Mutual Distillation.** Retrain-free NAS method [8] proposes a new solution for training multiple networks simultaneously, it can directly slice subnets of different sizes from the supernet and train them synchronously during the training process of the supernet. However, training multi-networks synchronously may cause gradient conflict of different subnets, resulting in the accuracy gap between single network and multi-networks [20]. Therefore, BigNAS uses in-place distillation and sandwich rules to achieve supernet training. However, there is an issue with the in-place distillation. It uses the max subnet to distill two random subnets and the min subnet. Theoretically, it cannot distillate the knowledge between random sampling networks and the minimal network. A naive solution is all subnets distill each other, which can provide different perspectives and knowledge compared to relying on the max subnet to distill others. To address this problem, we mutually distill all the networks extracted by the sandwich rule in the pre-training stage of the supernet. The workflow is illustrated in Fig 2.

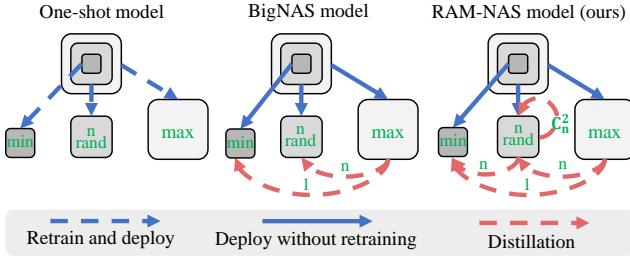


Fig. 2. Comparison with other existing workflows. We employ nested squares to represent models with shared weights, use squares of different sizes to denote subnets at various scales, and use max, N rand, and min to indicate sampled subnets. The numbers within red dotted lines signify the count of distillations for connected subnets.

In the supernet training phase, we apply the sandwich rule to sample the minimum model, maximum model, and  $N$  random models between the minimum and maximum (with  $N$  set to 2 in our experiments, following the approach in BigNAS [8]). Each sampled subnet serves as a teacher for knowledge distillation to other subnets, enabling the calculation of their respective logits losses. The formulation for this process is as follows:

$$L_{\text{sum}} = L_{\text{max}} + \sum_{i=1}^4 L_{\text{sub}_i}, \quad (3)$$

$$L_{\text{sub}_i} = \frac{1}{k-i} \sum_{j=i+1}^k D_{\text{KL}}(p_i || p_j),$$

where  $L_{\text{max}}$  denotes the loss of ground truth with maximum subnet and  $L_{\text{sub}_i}$  denotes the  $i$ -th subnet with other subnets.  $D_{\text{KL}}(p_i || p_j)$  denotes the KL divergence between the prediction logits  $p_i$  and  $p_j$  of teachers and students, it takes into account the remaining  $k - i$  subnets as imitated teachers, calculates their KL divergence, and then computes the average.

Due to the weight-sharing strategy, multiple forward-backward propagation processes are necessary for each randomly sampled subnet to distill knowledge from one another. Every sampled subnets  $\alpha_i \subseteq \mathcal{A}$  and its weight  $\mathbf{w}_i \subseteq \mathbf{W}$ . It subsequently aggregates all gradients from the ground truth with the maximum subnet and all mutual distillers before updating the weights. The gradient aggregation process can be formulated as follows:

$$\mathbf{W} = \begin{cases} \mathbf{W} + \gamma \frac{\partial L_{\text{max}}}{\partial \mathbf{w}}, & \text{all weights} \\ \mathbf{W} + \gamma \frac{\partial L_{\text{sub}_i}}{\partial \mathbf{w}}, & \text{if } \mathbf{w} \in \mathbf{W}_{stu} \\ \mathbf{W}, & \text{if } \mathbf{w} \notin \mathbf{W}_{stu} \end{cases} \quad (4)$$

where  $\gamma$  denotes the learning rate and  $\mathbf{W}_{stu}$  denotes the weight belonging to the student model in distillation. During the gradient update phase, we solely update the gradient associated with the student model and keep the weights of other components fixed. Since both the randomly sampled subnets and min-max subnets are included in the gradient aggregates, mutual distillation can update the weights of all dimensions throughout the iteration.

**Decoupled Knowledge Distillation.** Previous research [21] has demonstrated that feature distillation consistently

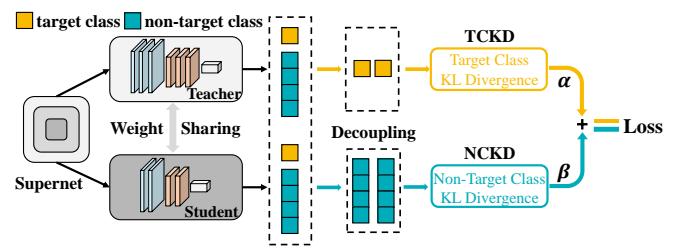


Fig. 3. Illustration of the DKD. It couples the importance of two parts about knowledge distillation: the loss of both the target class and the non-target class.

outperforms logits-based distillation across various tasks. However, the varying depths of subnets within our supernet pose a challenge in determining the state of the intermediate feature layer during mutual distillation. This makes it challenging to extract features from the feature extraction layers. To enhance the effectiveness of logits distillation, a method proposed in [22] is adopted. This method involves splitting the traditional KD loss into two components: target class knowledge distillation and non-target class knowledge distillation. Each component is then calculated separately and their losses are combined using a weighted sum. The workflow for this approach is depicted in Fig 3. The DKD method enables the attainment of greater flexibility and efficiency in knowledge distillation, while also addressing the issue of the non-target class being overshadowed by the target class in traditional distillation methods. The formula can be represented as follows:

$$L_{\text{DKD}} = \alpha \times L_{\text{TC}} + \beta \times L_{\text{NC}}, \quad (5)$$

$$L_{\text{TC}} = D_{\text{KL}}(\mathbf{c}^T || \mathbf{c}^S),$$

$$L_{\text{NC}} = D_{\text{KL}}(\hat{\mathbf{c}}^T || \hat{\mathbf{c}}^S),$$

where  $\alpha$  and  $\beta$  denotes the weight of target class loss  $L_{\text{TC}}$  and non-target class loss  $L_{\text{NC}}$ , respectively.  $\mathbf{c}^T$  and  $\mathbf{c}^S$  denote the probability distributions of the target class for the teacher and the student, calculated using the softmax function. Similarly,  $\hat{\mathbf{c}}^T$  and  $\hat{\mathbf{c}}^S$  represent the probability distributions of the non-target classes for the teacher and the student, also calculated using the softmax function.

### C. Latency Surrogate Predictor

After the supernet training is completed, a multiobjective evolutionary search is needed to obtain the Pareto frontier. To reduce the computational burden caused by evaluating subnets performance, we use surrogate predictors to assist neural architecture search and speed up the fitness evaluation process. We first random sample subnets  $\alpha_i$  from pre-trained supernet search space  $A$  and get its encoding format, then using corresponding weights to evaluate the accuracy  $f_i$  on the validation set. To ensure real-time execution of the searched subnets on the robot hardware platform, our objective is to minimize the inference delay on the hardware platform. Nevertheless, the intricate calculations involved in the search process necessitate its execution on the server platform. As a consequence, it is unable to consider the inference latency of the actual robot hardware. To address

this limitation, we employ a latency surrogate predictor to predict the actual inference latency on the target hardware. For the reusability of the latency surrogate predictor, we randomly sample the subnet  $\alpha_i$  and obtain the inference latency indicator  $l_i^n$  on the target platforms, where  $n$  is the  $n$ -th hardware platform. We randomly sample subnets  $\alpha_i$  and obtain the inference latency metric  $l_i^n$  on the target platform. The implementation of this part is shown in section IV, part A. Latency datasets  $(\alpha_i, l_i^n)$  are then constructed for training the latency surrogate predictor. Subsequently, the latency surrogate predictor is trained. This predictor can establish a mapping relation between latency and encoding format, and the predictor drives the evolutionary search. However, as indicated in [15], the efficacy of the surrogate predictor heavily relies on the accuracy of its predictions. Therefore, the fulfillment of the following two properties is crucial: (1) High-rank correlation between predicted performance and true performance. (2) Sample efficiency: Since obtaining each sample requires expensive subnet inference evaluation on hardware, it is necessary to minimize the number of samples required to train the surrogate predictor.

#### D. Multiobjective Evolutionary Search under Resource-aware

We employ the NSGA-II algorithm [23] as the standard multiobjective algorithm to explore the trade-off between model accuracy and hardware latency, two conflicting objectives. The pseudo-code for this method is provided in Algorithm 1. First, the initial population size  $P$  is obtained through random initialization, and then the dominance relationship of each individual in the population is calculated and the non-dominated solution is determined. Based on the non-dominance, the solution is divided into different frontier levels. At the same time, in order to prevent the solution distribution density from being too large and maintain the diversity of the Pareto frontier, sorting is performed by calculating the crowding degree, and finally, the solution is obtained based on the non-dominated sorting and crowding degree calculation results. NSGA-II uses a standard genetic algorithm, at the beginning of the genetic algorithm,  $P$  architectures are randomly selected as seeds. The top  $K$  architectures are selected as the parent generation, and the next generation is generated through crossover and mutation. For crossover, two randomly selected candidates and crossed to produce a new candidate in each generation. For mutation, the candidate mutates with probability  $P_m$  to produce a new architecture. The next generation is generated through crossover and mutation, which helps to retain the elite individuals in the Pareto frontier. As the algorithm iterates, a Pareto frontier is eventually formed and non-dominated solutions are output from the pool of evaluated architectures.

## IV. EXPERIMENT

In this section, we evaluate the performance of RAM-NAS on various robot vision tasks. In addition, we conducted a comparison of results achieved through FLOPs and resource-aware search, assessing the inference latency on the robotic

---

#### Algorithm 1 Multiobjective Evolutionary Search

---

**Input:** search space  $A$ , supernet weights  $W_A$ , population size  $P$ , number of generation iteration  $T$ , mutation probability  $p_m$ , top  $K$  architectures.  
**Output:** The searched architecture  $\alpha^*$  with  $w^*$  on the Pareto frontier.

```

1: Random sample  $P_1$  subnet architectures  $\{\alpha_1, \alpha_2, \dots, \alpha_P\}$  from  $A$  and its weights  $\{w_1, w_2, \dots, w_p\} \in W_A$ 
2: Latency predictor  $pred_{lat} \leftarrow fit(\alpha_i, l_i^n)$ 
3:  $n := P/2$ ,  $m := P/2$ 
4: while search step  $t \in (2, T)$  do
5:   Acc  $\leftarrow eval(\alpha_i, w_i)$ , Lat  $\leftarrow pred_{lat}(\alpha_i, w_i)$ 
6:    $P_t \leftarrow \text{NSGA-II}(\text{Acc}, \text{Lat}, \alpha_i, w_i)$ 
7:    $K \leftarrow \text{Update-Topk}(P_t)$ 
8:    $P_{t+1}^c \leftarrow \text{Crossover}(K, n)$ 
9:    $P_{t+1}^m \leftarrow \text{Mutation}(K, m, p_m)$ 
10:   $P_{t+1} \leftarrow P_{t+1}^c \cup P_{t+1}^m$ 
11: end while
12: return Pareto frontier architecture  $\alpha^*$  with  $w^*$ 
```

---

TABLE II

SUPERNET PRE-TRAIN AND DOWNSTREAM TASKS FINETUNE SETTINGS.

Tasks	Epochs	Warmup Epochs	Optimizer	Lr	Lr scheduler	Weight Decay	Dataset
Classification	240	10	SGD	0.8	cosine	1e-4	ImageNet-1K
Detection	20	1	SGD	0.2	cosine	1e-4	COCO
Segmentation	20	1	SGD	0.2	cosine	1e-4	Cityscapes

hardware platform. Finally, we conduct ablation experiments to verify our improved supernet training strategy and evaluate the correlation and sampling efficiency of surrogate predictors for latency.

#### A. Implementation Details

##### Supernet Training and Downstream Tasks Finetune.

The details of these training runs can be found in Table II. We conducted all training sessions on 8 GPUs (NVIDIA GeForce RTX 3090) using PyTorch 1.11. For image classification, a mini-batch size of 448 was employed, whereas for object detection, it was 144, and for semantic segmentation, it was 8. To better adapt our method to robot operational scenarios, we applied series data augmentation strategies, such as brightness adjustment, noise addition, and random mask. Unless explicitly stated otherwise, all other configurations were kept consistent with those outlined in BigNAS [8].

**Robotic Hardware Resource-aware.** In our latency surrogate predictor fitting and resource-aware experiments, we use three Nvidia-produced robotic hardware types: Nvidia Jetson AGX Orin, Xavier, and Nvidia Xavier NX. We use ONNX, an open format designed for model representation, to measure the latency of these models on three different types of hardware. It's important to mention that all models utilize onnxruntime-gpu for inference, with the ONNX component CUDAExecutionProvider and fp32 employed to accelerate the inference process.

**Latency Surrogate Predictor Fitting.** Following [12], we employ four distinct surrogate predictors for predic-

tion: Multi-Layer Perceptron (MLP), Classification And Regression Trees (CART), Radial Basis Function (RBF), and Gaussian Process (GP). First, we sampled 3000 subnets from the supernet, and their latencies were measured across three types of hardware. The dataset was split, with 80% used for training and 20% reserved for testing purposes. Subsequently, we choose the most effective predictor on the test dataset as our surrogate predictor.

**Multiobjective Evolution Search.** To initiate the process of evolutionary search, we select a population size of 5120 random architectures, denoted as  $P$ . Subsequently, the top 2560 architectures from this population are chosen as parents to generate the next generation through crossover and mutation. Additionally, to promote diversity within the population, a candidate may undergo mutation with a probability of  $P_m$  to generate a new architecture. In all of our experiments, we have set  $P_m$  to 0.2. The mutation and crossover operations are iteratively performed until the population size  $p$  is reached with a sufficient number of new candidates. The maximum number of iterations  $T$  is set to 30.

### B. Classification task

Classification is a basic task to prove the effectiveness of our supernet training strategy. To facilitate comparison with previous methods, we first set the optimization goal to top-1 accuracy and FLOPs on the validation data, demonstrating the benefits of our supernet pre-training strategy. The Supernet encompasses sizes ranging from 200 to 2000 MFLOPs, within which we explore architectures spanning from 200 to 1000 MFLOPs. Table III shows the results on ImageNet, establishing new accuracy vs. FLOPs trade-offs. Under constraints ranging from 200-1000 MFLOPs, our method surpassed all competitors. For small-sized models, our RAM-NAS-A0 achieves 76.7% accuracy under only 211 MFLOPs, which is 1.5% higher than MobileNetV3 1.0 $\times$  with fewer FLOPs. In the case of EfficientNet, our RAM-NAS-A3 achieves a Top-1 Accuracy of 79.1% with a significantly lower requirement of 460 MFLOPs, as compared to the 700 MFLOPs needed by EfficientNet-B1. In contrast to previous NAS methods, our approach yields superior outcomes. For instance, when compared to BigNAS-S, our method attains a Top-1 accuracy of 76.7% using only 211 MFLOPs, whereas BigNAS-S necessitates 242 MFLOPs to achieve a Top-1 accuracy of 76.5%.

Besides, we run RAM-NAS with resource-aware objectives: maximize Top-1 accuracy on ImageNet validation data and minimize latency on the hardware platforms. As illustrated in Fig 4, the FLOPs-based models correspond to RAM-NAS-A0 through RAM-NAS-A7. The Latency-based models are discovered using the latency surrogate predictor that is capable of recognizing hardware resources. To better align with the operational requirements in robotics, we have standardized the batch size for inference to 1. Compared to models that rely on the search for FLOPs as optimization targets, our resource-aware method can search for models with smaller inference latency and higher accuracy on each target hardware. Simultaneously, experimental results indi-

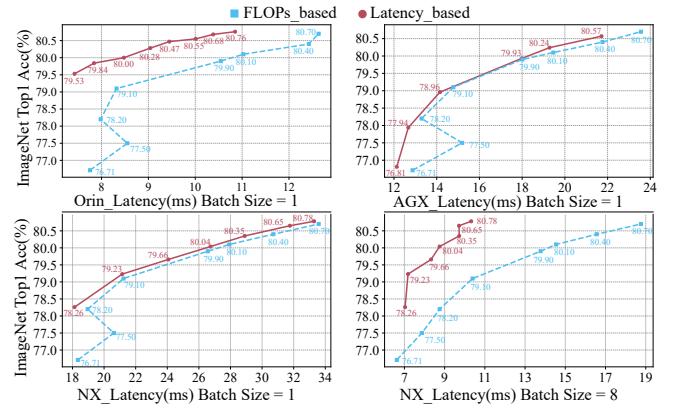


Fig. 4. Illustration of RAM-NAS with resource-aware objectives and FLOPs objectives on three robotic hardware.

cate a lack of precise mapping between the FLOPs of a model and the actual hardware inference delay. Surprisingly, models with larger FLOPs exhibit lower inference delays, challenging the conventional understanding. For example, in Fig 4, RAM-NAS-A1 has fewer FLOPs than RAM-NAS-A2, but its inference latency is higher across all hardware platforms when the batch size is 1. Meanwhile, the disparity is more pronounced on Orin, whereas the margin is narrower on AGX and NX, since the DataLoader may be time-consuming. Hence, we experimented using a batch size of 8 on the NX, which possesses the lowest computing power. Our findings indicate that, with an increase in batch size, the Latency-based search model exhibits significant improvement compared to the FLOPs-based model.

TABLE III  
COMPARISON WITH PRIOR MANUAL DESIGN AND NAS APPROACHES ON  
IMAGENET.

Group	Method	FLOPs(M)	Top-1(%)
200-400 (M)	<b>RAM-NAS-A0</b>	<b>211</b>	<b>76.7</b>
	MobileNetV3 1.0 $\times$ [24]	217	75.2
	OFA [11]	230	76.0
	FBNetV2-F4 [13]	238	76.0
	BigNAS-S [8]	242	76.5
	<b>RAM-NAS-A1</b>	<b>284</b>	<b>77.5</b>
	MNasNet-A1 [19]	312	75.2
	ProxylessNAS [25]	320	74.6
	FBNetV2-L1 [13]	325	77.2
	<b>RAM-NAS-A2</b>	<b>325</b>	<b>78.2</b>
400-600 (M)	FBNetV3 [14]	343	78.0
	MobileNetV3 1.25 $\times$ [24]	356	76.6
	EfficientNet-B0 [26]	390	77.1
	MNasNet-A3 [19]	403	76.7
	BigNAS-M [8]	418	78.9
	FBNetV2-L2 [13]	422	78.1
	<b>RAM-NAS-A3</b>	<b>460</b>	<b>79.1</b>
600-800 (M)	BigNAS-L [8]	586	79.5
	<b>RAM-NAS-A4</b>	<b>625</b>	<b>79.9</b>
	EfficientNet-B1 [26]	700	79.1
	<b>RAM-NAS-A5</b>	<b>704</b>	<b>80.1</b>
>800 (M)	<b>RAM-NAS-A6</b>	<b>808</b>	<b>80.4</b>
	<b>RAM-NAS-A7</b>	<b>976</b>	<b>80.7</b>
	EfficientNet-B2 [26]	1000	80.1

### C. Detection and Segmentation task

Moreover, we apply the searched RAM-NAS models to object detection and semantic segmentation tasks to verify the scalability of our methods. For fair comparison, we

TABLE IV

MEAN AVERAGE PRECISION(MAP)/MEAN INTERSECTION OVER UNION(MIOU) AND LATENCY ON COCO AND CITYSCAPES DATASET.

Models	mAP/mIOU(%)	Latency on Orin(ms)
MobileNetV3-SSD	20.2	20.57
RAM-NAS-SSD (A4)	23.6	24.73
RAM-NAS-SSD (A7)	24.5	26.98
RAM-NAS-SSD (80.00)	23.3	23.43
RAM-NAS-SSD (80.76)	26.2	27.48
MobileNetV3-LRASPP	64.11	53.53
RAM-NAS-LRASPP (A4)	67.41	95.30
RAM-NAS-LRASPP (A7)	68.31	124.42
RAM-NAS-LRASPP (79.84)	66.76	82.95
RAM-NAS-LRASPP (80.00)	68.70	98.43

replaced the original backbone with the model obtained through our search process and the weights were fine-tuned specifically for each task. For object detection, we followed the MobileNetV3 [24] architecture and used the RAM-NAS search-derived backbone directly as the feature extractor for SSDLite. In our experiments, the initial layer of SSDLite was connected to stage 4 in search space, and the second layer of SSDLite was connected to the top of the stage 7. For semantic segmentation, the layer of LRASPP was connected to stage1, 2 and 7.

As shown in Table IV, we present the mAP/mIOU scores for each method on their respective datasets and provide the inference latency on Orin. The A4 and A7 represent the flops searched model, while 79.84, 80.00, 80.76 correspond to structures identified through the original latency surrogate model. In particular, RAM-NAS-SSD (80.76) achieves 26.2 mAP, providing +1.7 mAP gain over RAM-NAS-SSD (A7), and increases latency by 0.5 ms, RAM-NAS-LRASPP (80.00) achieves 68.70 mIOU, providing +0.39 mIOU gain over RAM-NAS-LRASPP (A7) and 1.27× speedup on Orin.

#### D. Ablation Study

**Improving the Supernet Training Strategy.** During the supernet pre-training phase, in order to figure out which component is working, we further performed the following ablation experiments. We evaluated their performance based on two dimensions: min subnet and max subnet accuracy on Imagnet, as they determine the lower and upper bounds of the supernet, as reported in Table V. In Baseline, we employ in-place distillation and naive distillation loss [27] in BigNAS. In EXP1, we use subnets mutual distillation and naive distillation loss in our subnets. This approach results in a Top1 improvement of 0.4% for the min subnet and 0.18% for the max subnet. It aggregates knowledge and information across networks to maximize the lower bound of the network, enabling each network to acquire additional knowledge from other networks. In EXP2, we use naive distillation and Decoupled Knowledge Distillation, the  $\alpha$  is set as 1.0, and  $\beta$  is set as 0.5. It yields a 0.28% boost and 0.17% over baseline. Compared to using the default BigNAS settings [8], utilizing SMD and DKD separately notably enhances the performance of the min subnet and max subnet. Combining both further enhances the performance of the min subnet and max subnet. Our supernet training

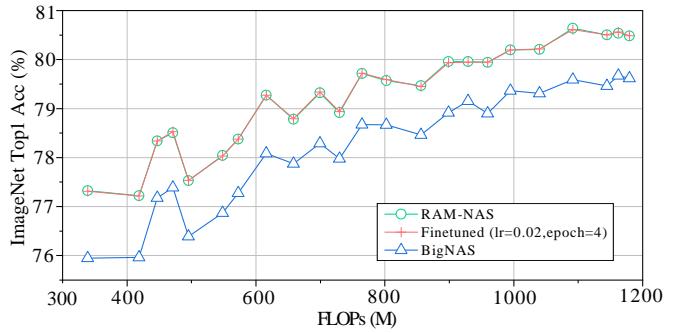


Fig. 5. Illustration of the consistency between subnet directly searched by BigNAS [8], our method, and finetuned performance. strategy surpasses the baseline 0.44% and 0.36% with the same structure.

TABLE V  
ABLATION STUDY RESULTS ABOUT THE IMPROVED SUPERNET TRAINING STRATEGY ON IMAGENET. SMD: SUBNETS MUTUAL DISTILLATION. DKD: DECOUPLED KNOWLEDGE DISTILLATION.

	Methods		min subnet		max subnet	
	SMD	DKD	Top1(%)	Top5(%)	Top1(%)	Top5(%)
Baseline			76.24	92.76	81.03	95.10
EXP1	✓		76.64 <sup>(+0.40)</sup>	92.96 <sup>(+0.20)</sup>	81.21 <sup>(+0.18)</sup>	95.21 <sup>(+0.11)</sup>
EXP2		✓	76.52 <sup>(+0.28)</sup>	92.89 <sup>(+0.13)</sup>	81.20 <sup>(+0.17)</sup>	95.18 <sup>(+0.08)</sup>
EXP3	✓	✓	76.68 <sup>(+0.44)</sup>	92.95 <sup>(+0.19)</sup>	81.39 <sup>(+0.36)</sup>	95.26 <sup>(+0.16)</sup>

Furthermore, to demonstrate the effectiveness of our proposed supernet training strategy in enhancing the consistency between the directly searched subnets and the performance after fine-tuning, we randomly sampled subnets with varying FLOPs. To attain optimal subnet performance, the RAM-NAS sampled subnets undergo fine-tuning with a learning rate of 0.02 over 4 epochs to achieve their actual performance. At the same time, the random seeds are fixed and the supernet trained by BigNAS is randomly sampled. The results are as shown in Fig 5. The figure illustrates that subnets obtained through our supernet training strategy exhibit comparable or even superior performance compared to finetuned results across different FLOPs, with an RMSE of just 0.067 about Finetuned and RAM-NAS. Meanwhile, RAM-NAS outperforms BigNAS in subnets of varying FLOPs. This result demonstrates the effectiveness of our approach in benefiting subnets at all levels within the supernet. It enhances the generalization of subnets and helps discover more robust values for weight-sharing supernet parameters.

**Performance of the Surrogate Predictors.** We use Spearman correlation coefficient  $\rho$  and Kendall correlation coefficient  $\tau$  to quantitatively evaluate the ranking correlation of surrogate predictors. These indicators have a value range of [-1, 1]. A value closer to 1 indicates a higher prediction performance. In addition to comparing correlation coefficients, sample efficiency is also used as an evaluation metric for Surrogate Predictors. We replicate the previous experiment and systematically decrease the number of samples used to train the Predictors. Then, we evaluated the correlation coefficients and sample efficiency across three distinct hardware setups and generated a dot-line plot illustrating the variances,

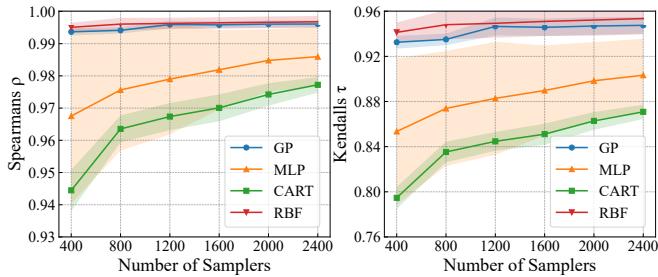


Fig. 6. Illustration of the latency predictors ranking correlation and sampling efficiency.

as depicted in Fig 6. The findings indicate that GP and RBF outperform other methods in predicting latency consistency and sampling efficiency. Even with a small sampling size, they consistently maintain a high ranking. However, RBF exhibits a superior mean compared to GP. Therefore, our method can achieve excellent performance on new hardware with only a small number of samples for both sampling and training.

## V. CONCLUSIONS

In this paper, we introduced RAM-NAS, a resource-aware multiobjective evolutionary NAS method that focuses on improving the supernet pre-train and robot resource-aware. We propose a subnet mutual distillation to substitute the in-place distillation and utilize DKD Loss to achieve more flexible and efficient knowledge distillation. We also develop a surrogate predictor to achieving high-rank correlation and sample efficiency. Our approach is the first to incorporate robotic hardware resources into neural architecture search, achieving a trade-off of accuracy and latency through multiobjective optimization. Based on the RAM-NAS method, there is considerable potential for further research: (1) Exploring more complex neural architectures. (2) Conducting latency testing with a greater number of sensor inputs and higher resolutions. (3) Adapting to other types of robotics hardware platforms, such as x86, ARM, and NPU.

## REFERENCES

- [1] Z. Zhao, K. Wang, N. Ling, and G. Xing, “Edgeml: An automl framework for real-time deep learning on the edge,” in *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, 2021, pp. 133–144.
- [2] Z. Xu, X. Hong, T. Chen, Z. Yang, and Y. Shi, “Scale-aware squeeze-and-excitation for lightweight object detection,” *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 49–56, 2022.
- [3] J. Zhang, M. Henein, R. Mahony, and V. Ila, “Vdo-slam: a visual dynamic object-aware slam system,” *arXiv preprint arXiv:2005.11052*, 2020.
- [4] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” *Advances in neural information processing systems*, vol. 28, 2015.
- [5] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.
- [6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.
- [7] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, “Single path one-shot neural architecture search with uniform sampling,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*. Springer, 2020, pp. 544–560.
- [8] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le, “Bignas: Scaling up neural architecture search with big single-stage models,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*. Springer, 2020, pp. 702–717.
- [9] D. Wang, M. Li, C. Gong, and V. Chandra, “Attentivenas: Improving neural architecture search via attentive sampling,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 6418–6427.
- [10] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10734–10742.
- [11] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” *arXiv preprint arXiv:1908.09791*, 2019.
- [12] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, “Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 2020, pp. 35–51.
- [13] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen *et al.*, “Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 12965–12974.
- [14] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda *et al.*, “Fbnetv3: Joint architecture-recipe search using predictor pretraining,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16276–16285.
- [15] Z. Lu, R. Cheng, S. Huang, H. Zhang, C. Qiu, and F. Yang, “Surrogate-assisted multiobjective neural architecture search for real-time semantic segmentation,” *IEEE Transactions on Artificial Intelligence*, 2022.
- [16] J. Yu and T. S. Huang, “Universally slimmable networks and improved training techniques,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.
- [17] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang, “Densely connected search space for more flexible neural architecture search,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10628–10637.
- [18] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan, “Mobileone: An improved one millisecond mobile backbone,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7907–7917.
- [19] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.
- [20] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020.
- [21] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *arXiv preprint arXiv:1412.6550*, 2014.
- [22] B. Zhao, Q. Cui, R. Song, Y. Qiu, and J. Liang, “Decoupled knowledge distillation,” in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 2022, pp. 11953–11962.
- [23] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [24] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.
- [25] H. Cai, L. Zhu, and S. Han, “Proxylessnas: Direct neural architecture search on target task and hardware,” *arXiv preprint arXiv:1812.00332*, 2018.
- [26] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.
- [27] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.