A Supplementary Material

This **Supplementary Material** is structured as follows. We provide a formulation of our algorithm in Section B. To investigate the effectiveness of different components of our SUBP, we conduct ablation studies and provide additional experimental results in Section C. In Section D, we provide deployment results on the x86 platform of Intel(R) Xeon(R) Platinum 8260L CPU @ 2.30GHz to further explore the performance of 1×N sparse on different platforms. Finally, in Section E, we discuss the societal impact of our method.

B Algorithm Formulation

Algorithm 1: Overview of the SUBP method.

Input: An L-layer CNN model with weights $\mathbf{W} = \{W^i | 1 \leq i \leq L\}$; block binary mask matrices $\mathbf{M} = \{M^i_{j,k} \in \{0,1\} | 1 \leq i \leq L, 0 \leq j \leq \frac{C_{i+1}}{N} - 1, 0 \leq k \leq C_i - 1\}$; indices of activate blocks with the top scores $\mathbf{T}$; indices of regrow blocks based on importance sampling $\mathbf{G}$; target prune rate p; initial regrowing factor δ_0; importance balance coefficient λ; sampling attention balance factor τ; training epochs T_{total}; start and end epoch in the pruning-regrowing stage t_s, t_e; training set $\mathcal{D}$;

Output: A sub-model satisfying the target prune rate p, its optimal weight values $\mathbf{W}^*$ and binary mask $\mathbf{M}^*$;

```
Randomly initialize the model weights W;
Initialize {M^i_{j,k} | ∀i, ∀j, ∀k} to 1 ;
Reformat W to B according to Section 3 ;
for each training epoch t ∈ [T_total] do
    Sample a mini-batch from D and update the model weights W ;
    if t_s < t ≤ t_e then
        Reset {M^i_{j,k} | ∀i, ∀j, ∀k} to 1 ;
        Compute the importance score S of block by Eq. 4 ;
        Get the indices of activate blocks with the top scores by Eq. 5 ;
        Prune the bottom-ranking block by set {M^i_{j,k} | k ∉ T^i_j} to 0;
        Compute the importance sampling probabilities by
            p^i_{j,k} = exp(S^i_{j,k} / τ) / Σ_{m∉T^i_j} exp(S^i_{j,m} / τ) ;
        Compute the regrowing factor by Eq. 6 ;
        Get the indices of regrow blocks based on importance sampling by
            G^i_j = Multinomial({p^i_{j,k} | k ∉ T^i_j}, δ_t C_i) without replacement ;
        Regrow the blocks by resetting {M^i_{j,k} | k ∈ G^i_j} to 1 ;
```

C Ablation Analysis

Table 4: Compare different design choices in the regrowing stages of the SUBP method. All the experiments are based on the TinyImageNet with ResNet18(1×32). The random baseline is 57.0%.

Regrowing factor					
Design choices	$\delta_0 = 0.1$	$\delta_0 = 0.2$	$\delta_0 = 0.3$	$\delta_0 = 0.4$	Full
Top-1	57.6%	58.4%	57.9%	58.0%	58.5%
Decay scheduler for block regrowing					
Design choices	Default	Constant	Linear decay	Cosine decay	
Top-1	58.3%	57.5%	58.4%	58.3%	

In Table 4, we investigate the effectiveness of different design choices in our block regrowing stage. All the experiments are based on the TinyImageNet with ResNet18(1×32). Compared to the random baseline with 57.0% top-1 accuracy, our SUBP achieves consistent improvement under the different settings.

We find that regrowing factor δ_0 significantly impacts the final quality of the model. Intuitively, a larger regrowing factor can provide a more extensive sampling space during training and retain the model's capacity to a greater extent. However, a sizeable regrowing factor may also cause drastic sub-model structure changes, affecting stability during training. As shown, the accuracy is improved by 0.8% as the δ_0 increases from 0.1 to 0.2. When δ_0 increases again, the model's accuracy drops until δ_0 is the full model size. This suggests that the relationship between the regrowing factor and the final quality of the model is varied, and selecting an appropriate regrowing factor in specific circumstances can improve the final quality.

We also investigate the decay scheduler for the block regrowing stage. We compare several decay schedulers, including default (Eq. 6), constant, linear, and cosine. The experiments show SUBP has good robustness to different decay schedulers, as default, linear, and cosine decay schedulers all show similar performance. With a decay scheduler, the sampling space can be gradually decreased, and the sub-model under training can converge stably.

D Deployment on x86 Platform

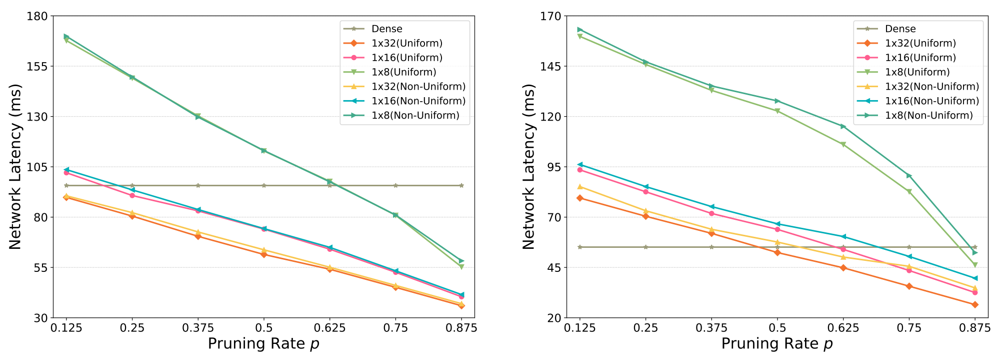

Figure 5: Network latency comparison between uniform 1×N sparse against non-uniform and dense model with varying N and prune rates. The experiment is conducted using ResNet18 and set the input shape as (4, 3, 224, 224) on the x86 platform of Intel(R) Xeon(R) Platinum 8260L CPU @ 2.30GHz with single thread (left) and two threads (right). Best viewd in colors.

In order to further explore the performance of 1×N sparse DNNs on different platforms, as shown in Fig. 5, we also conducted corresponding experiments on the x86 platform of Intel(R) Xeon(R) Platinum 8260L CPU @ 2.30GHz and obtain the similar results in general: 1) The gain of vanilla convolution in multithreading scenarios is much greater than that of 1×N sparse convolution. 2) The inference speed of uniform 1×N is slightly faster than that of non-uniform in the case of multithreading, indicating the importance of workload balance again. However, unlike the performance on the arm platform of Apple M1 Pro CPU @ 3.20GHz, the 1×N sparse DNNs are significantly accelerated when N is set to 16 and 32 on the Platinum 8260L CPU @ 2.30GHz. We can also notice that in most cases, N=32 achieves a fast inference speed.

E Societal Impact

Our method can reduce the computational overhead of training and inferencing stages while achieving satisfactory accuracy on modern CNN models. This can facilitate the application of CNN models on edge devices and is of high value for the community and society to realize Green AI. At the same time, our 1×N sparse DNNs are based on new sparse operators, which can promote the progress of related hardware and algorithms to a certain extent.