BARTER: COMPACT DYNAMIC SPARSE TRAINING WITH STANDARD DENSE KERNELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic sparse training (DST) aims to reduce training costs while maintaining accuracy. Existing DST methods mask out parameters but keep matrix dimensions unchanged. Training speedups therefore depend on specialized support, such as sparse kernels. We introduce BARTER, a structured DST framework that reduces training cost using standard dense kernels by keeping training tensors physically compact while dynamically reallocating capacity across layers. Unlike existing works that require computing gradients for all parameters, BARTER instead uses statistics collected only from active units during ordinary backpropagation to prune and clone attention heads and feed-forward channels under separate fixed global budgets. At each topology update, it resizes the affected parameter tensors and remaps their optimizer states, so subsequent training steps operate directly on the compact model. Training and fine-tuning experiments with Transformer-based vision models on two image classification benchmarks at densities from 0.1 to 0.7 show competitive Top-1 accuracy relative to full-tensor DST baselines. On an NVIDIA L40, BARTER achieves speedups of up to $6.29\times$ in measured training segments and reduces peak allocated GPU memory by up to 81.3% relative to these implementations. An iPhone 13 Pro Max training benchmark additionally shows a $2.78\times$ speedup over the full-tensor DST baseline.

1 INTRODUCTION

Training neural networks requires substantial computation and memory, while deployment imposes limits on model size and inference cost. Pruning reduces deployment costs by removing redundant weights (Han et al., 2015) or structural units (Li et al., 2016), but training a dense model before pruning still incurs the full training cost. Dynamic sparse training (DST) (Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019; Evci et al., 2020) aims to reduce this training cost while maintaining accuracy. It trains a sparse network from initialization under a limited active-parameter budget. The initial topology may not make the best use of this budget, so DST periodically prunes and grows connections to revise its capacity allocation as it learns.

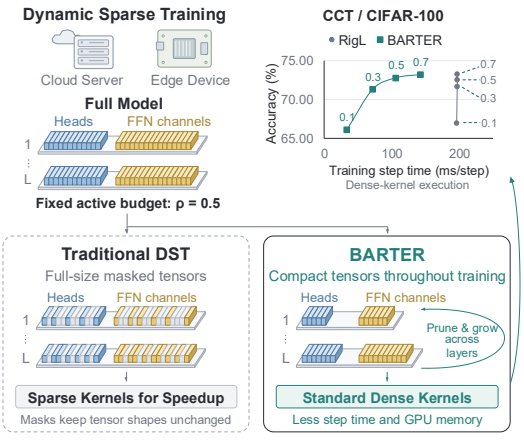

Figure 1: Full-size masked training versus compact dynamic training with BARTER. The plot compares CCT/CIFAR-100 accuracy and schedule-weighted training step-time estimates for RigL and BARTER under dense-kernel execution. Labels indicate density.

However, fewer active parameters do not necessarily mean cheaper training. Existing DST methods mask out parameters but keep matrix dimensions unchanged (Evci et al., 2020; Chen et al., 2021; Lasby et al., 2024). Standard dense kernels still process the full-size tensors. Sparse kernels can skip computations on inactive weights to accelerate training, but require implementations tailored to the sparsity format and execution backend (Nikdan et al., 2023; Hu et al., 2024; Mozaffari et al., 2025). This limits which platforms can benefit from these speedups.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

To achieve reductions in computation with standard dense kernels, a logical approach is to remove inactive structural units and train on a smaller tensor consisting only of active units. However, it is not possible to do so using existing approaches because they require using the gradients of inactive candidates to guide growth (Evci et al., 2020; Chen et al., 2021; Lasby et al., 2024); once these units are physically removed, ordinary backpropagation no longer provides those gradients.

Our Approach. We propose BARTER, a structured DST framework that prunes and grows structural units directly on compact tensors using standard dense kernels (Figure 1). Under fixed global Budgets, BARTER uses only Active-unit statistics to Reallocate capacity during Training. It maintains compact dense Execution by Resizing the affected tensors. In our Transformer implementation, attention heads and feed-forward network (FFN) channels have separate budgets, and training uses compact tensors from the first optimization step. At each update, BARTER uses these statistics to prune a unit from a donor layer and clone an active unit within the receiving layer. The clone starts from a feature the model has already learned, and the parent's outgoing weights are split between the two units. We resize the affected parameters and remap their optimizer states so training can continue on the compact model. The final model can be deployed without further pruning.

Our contributions are summarized as follows:

- We introduce BARTER, a structured DST framework that reallocates capacity under fixed global budgets while keeping training tensors physically compact. Unlike existing DST implementations that rely on specialized sparse kernels for acceleration, BARTER achieves training speedups using standard dense kernels and produces a compact model for deployment.
- We develop a reallocation mechanism that selects units for pruning and cloning. Unlike prior growth rules that rely on gradients of inactive candidates, it uses only active-unit statistics. Tensor resizing and optimizer-state remapping allow training to continue directly on the compact model after each structural update.
- We conduct training and fine-tuning experiments with two Transformer architectures on two image classification benchmarks, comparing against five state-of-the-art DST methods and a dense reference. Under standard dense-kernel execution, BARTER maintains competitive classification performance while achieving speedups of up to $6.29\times$ in measured GPU training segments and reducing peak allocated GPU memory by up to 81.3% relative to these DST methods. A smartphone training benchmark shows a $2.78\times$ speedup over RigL.

2 Related Work

Unstructured dynamic sparse training. Unstructured DST learns sparse connectivity alongside network weights under limited connection budgets (Bellec et al., 2017; Mocanu et al., 2018). SET (Mocanu et al., 2018) combines magnitude-based pruning with random growth, while RigL (Evci et al., 2020) selects new connections using gradient magnitudes. CHTs (Zhang et al., 2026) uses topology-based link prediction to guide regrowth. Beyond the selection of individual connections, dynamic sparse reparameterization redistributes a fixed nonzero parameter budget across layers (Mostafa & Wang, 2019), and in-time over-parameterization relates training performance to the exploration of sparse configurations over time (Liu et al., 2021). These methods adapt connectivity at the level of individual weights. BARTER changes the granularity of reallocation to complete attention heads and FFN channels, physically adjusting the corresponding tensor dimensions as capacity moves across layers.

Structured dynamic sparse training. Structured DST introduces constraints on the units or connectivity patterns being updated. S^2ViTE (Chen et al., 2021) prunes and grows attention heads and FFN channels; Chase (Yin et al., 2023) combines channel pruning with weight-level redistribution; and SRigL (Lasby et al., 2024) maintains constant fan-in connectivity with neuron ablation. Their growth rules use gradients of inactive candidate weights or units (Chen et al., 2021; Yin et al., 2023; Lasby et al., 2024), which ordinary backpropagation through a physically compact active model does not provide. BARTER supports growth by cloning an existing active unit within the receiving layer, with reallocation decisions based only on active-unit statistics. Transfers preserve separate

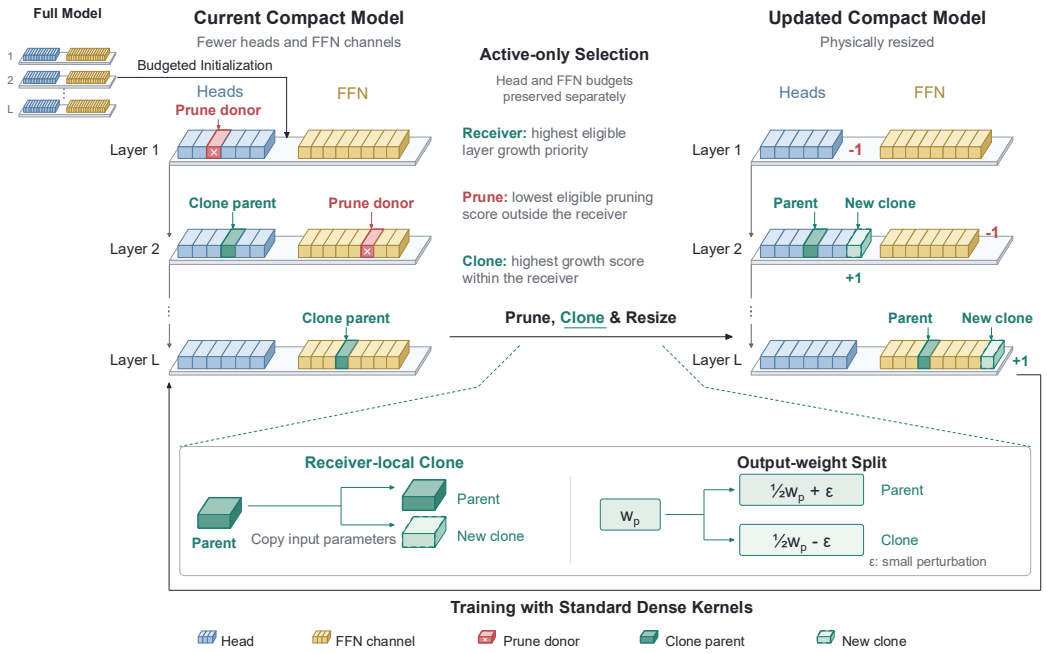

Figure 2: Overview of our approach, BARTER. Budgeted initialization produces a compact model with fewer attention heads and FFN channels while retaining all Transformer layers. During training, statistics from active units guide capacity reallocation across layers. For each unit type, the method selects the eligible receiving layer with the highest growth priority, prunes eligible units with the lowest pruning scores outside that layer, and adds the same number of units by cloning parents with the highest growth scores within it. These updates preserve the global head and FFN budgets separately. Each clone copies its parent's input parameters, while the parent's output weights are split between the parent and clone. The affected parameter tensors and optimizer states are resized accordingly, and training continues on the compact model with standard dense kernels.

global budgets for attention heads and FFN channels, while parameter resizing and optimizer-state remapping keep subsequent training compact.

Sparse execution for training. MEST (Yuan et al., 2021) combines memory-economic sparse training with compiler-level optimizations for mobile execution. DSB (Jiang et al., 2022) organizes changing connectivity into shuffled blocks for efficient sparse computation, while DynaDiag (Tyagi et al., 2025) maintains diagonal sparsity and accelerates training with custom CUDA kernels. For large output spaces, HASTE (Ullah et al., 2026) pairs group-shared fixed fan-in patterns with custom CUDA kernels. Transformer pre-training has also benefited from optimized forward and backward kernels for 2:4 sparsity (Hu et al., 2024), while CHTs24 (Lyu et al., 2026) studies dynamic connectivity updates under transposable 2:4 constraints. These approaches tailor execution to particular sparsity patterns and backends. BARTER follows a different route: it reduces layer dimensions during training and executes the resulting compact tensors with standard dense kernels.

3 METHODOLOGY

BARTER adapts the allocation of attention heads and feed-forward channels across layers while training a physically compact model. Figure 2 provides an overview of the framework. We first define the compact model and its capacity budgets (Section 3.1), then describe how statistics from active units guide cross-layer reallocation (Section 3.2). We next present the prune-and-clone operation (Section 3.3) and explain how the affected parameters and optimizer states are resized so that training can continue with standard dense kernels (Section 3.4).

3.1 Compact Model and Capacity Budgets

Consider a model with L Transformer layers. We denote the numbers of active attention heads and feed-forward channels in layer ℓ by h_ℓ and c_ℓ, respectively. We store the parameters of these active units in compact dense tensors and train the model with standard dense kernels.

At initialization, the target density ρ specifies the fraction of the original model's prunable weights to retain. We denote the global budgets for attention heads and feed-forward channels by H and C, respectively:

$$H = \sum_{\ell=1}^{L} h_\ell, \qquad C = \sum_{\ell=1}^{L} c_\ell. \tag{1}$$

We choose integer values for these budgets based on ρ and the parameter costs of the two unit types, keeping the retained fractions of heads and channels close to ρ. We distribute each budget as evenly as possible across layers, assigning any remainder to randomly chosen layers. Within each layer of the original model, we randomly select the allocated number of units and copy their parameters into the initial compact model.

Training begins with this compact model. During training, `BARTER` uses statistics from active units to guide capacity reallocation across layers. Each transfer removes a unit from one layer and adds a unit of the same type to another, keeping the global budgets in Eq. (1) fixed.

3.2 Active-Only Capacity Reallocation

Active-unit scoring. We use activations and their gradients from ordinary training to decide where to remove and add capacity. For attention heads, we collect the head outputs before the output projection; for FFN channels, we collect the hidden activations after GELU.

For an active unit i in layer ℓ, let $a_{\ell,i,b,j}$ denote its activation for sample b at output coordinate j. Here, b indexes samples in the current minibatch. For an FFN channel, j runs over token positions; for an attention head, it runs over token positions and features within the head. Let $\mathcal{L}$ be the minibatch loss and $g_{\ell,i,b,j} = \partial\mathcal{L}/\partial a_{\ell,i,b,j}$ the gradient with respect to that activation.

We multiply each activation value by its corresponding loss gradient (Molchanov et al., 2016; Michel et al., 2019). We aggregate these products into two minibatch scores: $d_{\ell i}$ for pruning and $s_{\ell i}$ for growth. Because pruning removes all outputs of the unit, we compute $d_{\ell i}$ by summing the signed activation-gradient products over all output coordinates within each sample, taking the absolute value, and then averaging across samples. For $s_{\ell i}$, we take the absolute value of each product before averaging over coordinates and samples, so contributions of opposite signs do not cancel:

$$\begin{aligned} d_{\ell i} &= \mathrm{mean}_b \left| \sum_j a_{\ell,i,b,j} g_{\ell,i,b,j} \right|, \\ s_{\ell i} &= \mathrm{mean}_b \left[\mathrm{mean}_j \left| a_{\ell,i,b,j} g_{\ell,i,b,j} \right| \right]. \end{aligned} \tag{2}$$

We maintain running pruning and growth scores, $D_{\ell i}$ and $S_{\ell i}$, using exponential moving averages (EMAs) of $d_{\ell i}$ and $s_{\ell i}$ (Molchanov et al., 2019). The decay factor β determines how much of the previous running score is retained:

$$\begin{aligned} D_{\ell i} &\leftarrow \beta D_{\ell i} + (1-\beta) d_{\ell i}, \\ S_{\ell i} &\leftarrow \beta S_{\ell i} + (1-\beta) s_{\ell i}. \end{aligned} \tag{3}$$

Both EMAs are initialized to zero at the start of training and updated during each ordinary backward pass while score collection is enabled. At scheduled topology updates, these running scores guide the selections below.

Receiver selection. We apply the selection procedure separately to attention heads and FFN channels. For the type being transferred, let $\mathcal{U}_\ell$ be the set of active unit indices in layer ℓ. Its current unit count is $n_\ell = |\mathcal{U}_\ell|$: $n_\ell = h_\ell$ for attention heads and $n_\ell = c_\ell$ for FFN channels. We denote the count at compact initialization by $n_{\ell,0}$ and the count in the original model by $n_\ell^{\max}$.

A receiving layer must contain an active unit to clone and have room to grow, giving $0 < n_\ell < n_\ell^{\max}$. For each eligible layer, we multiply the mean growth score of its active units by the ratio of its initial to current unit count. This gives the corrected layer score R_ℓ. We select the eligible layer with the highest score as the receiver:

$$\begin{aligned} R_\ell &= \frac{n_{\ell,0}}{n_\ell} \frac{1}{n_\ell} \sum_{i \in \mathcal{U}_\ell} S_{\ell i}, \\ r &= \underset{\ell:\, 0 < n_\ell < n_\ell^{\max}}{\arg\max} R_\ell. \end{aligned} \tag{4}$$

The width correction lowers the priority of layers that have expanded relative to initialization and raises that of layers that have contracted.

Donor and clone-source selection. With the receiving layer r fixed, let $\mathcal{C}_r$ contain the eligible donor pairs (ℓ, i) outside layer r, where ℓ is the layer index and i is the unit index within that layer. Removing a unit must leave at least one FFN channel per layer, while attention layers may have zero heads. Recently cloned parents and their new copies are excluded from donor selection during a cooldown period.

We choose the eligible unit with the lowest running pruning score $D_{\ell i}$. Its layer and within-layer indices are denoted by ℓ_d and i_d, respectively. Within the receiving layer, we choose the active unit with the highest running growth score S_{ri} as the clone parent and denote its index by i_p:

$$\begin{aligned} (\ell_d, i_d) &= \underset{(\ell,i) \in \mathcal{C}_r}{\arg\min} D_{\ell i}, \\ i_p &= \underset{i \in \mathcal{U}_r}{\arg\max} S_{ri}. \end{aligned} \tag{5}$$

We skip a transfer if no receiving layer is eligible or the selected receiver has no eligible donor. Otherwise, layer ℓ_d releases unit i_d, and layer r gains a unit of the same type by cloning its own active unit i_p. Section 3.3 describes this update.

3.3 Prune-and-Clone Updates

Unit removal and cloning. Using the selections from Section 3.2, `BARTER` removes the donor unit (ℓ_d, i_d) and creates a unit of the same type in receiving layer r by cloning its selected parent i_p. We remove an attention head's query, key, and value parameter blocks, including their biases when present, together with the corresponding output-projection weights. For an FFN channel, we remove its input weights, bias, and output weights. The new unit, indexed by i_{new} within layer r, is initialized from its parent (Wu et al., 2019; 2020). We copy the parent's parameters for producing features and initialize the two units' output weights jointly through the split below.

Output-weight splitting. We divide the parent's output weights between the parent and the new unit to avoid duplicating its original output contribution (Chen et al., 2015). Let $W_{r,i}^{\text{out}}$ denote the weights that map unit i's features to the output of its module in layer r, and let $W_{r,i_p}^{\text{out,old}}$ denote the parent's output weights before cloning. Using a small Gaussian perturbation ε with the same shape as these weights, we set

$$\begin{aligned} W_{r,i_p}^{\text{out}} &\leftarrow \tfrac{1}{2} W_{r,i_p}^{\text{out,old}} + \varepsilon, \\ W_{r,i_{\text{new}}}^{\text{out}} &\leftarrow \tfrac{1}{2} W_{r,i_p}^{\text{out,old}} - \varepsilon. \end{aligned} \tag{6}$$

The opposite perturbations break the symmetry between the two units while keeping the sum of their output weights equal to the parent's original weights. When the parent and its copy produce identical features, their combined output contribution therefore equals the parent's contribution before the split.

Score updates after cloning. We remove the donor's scores and retain the scores of units unaffected by the update. Let D_{r,i_p}^{old} and S_{r,i_p}^{old} denote the parent's running pruning and growth scores before cloning. We split each stored score equally between the parent and its new copy, keeping

their combined value unchanged:

$$\begin{aligned} D_{r,i_p} &\leftarrow \tfrac{1}{2} D^{\text{old}}_{r,i_p}, \quad D_{r,i_{\text{new}}} \leftarrow \tfrac{1}{2} D^{\text{old}}_{r,i_p}, \\ S_{r,i_p} &\leftarrow \tfrac{1}{2} S^{\text{old}}_{r,i_p}, \quad S_{r,i_{\text{new}}} \leftarrow \tfrac{1}{2} S^{\text{old}}_{r,i_p}. \end{aligned} \tag{7}$$

Both units enter the donor-selection cooldown defined in Section 3.2. Subsequent training updates their scores from their own activations and gradients using Equation 3. Section 3.4 describes how the affected parameter tensors and optimizer states are resized for continued training.

3.4 Physical Resizing and Continued Training

Physical resizing. At each topology update, we rebuild the affected tensors from the retained and newly created parameter slices defined in Section 3.3. We use matching unit indices across the associated tensors. For an FFN with c_ℓ active channels and embedding dimension d, the input and output weight matrices have sizes $c_\ell \times d$ and $d \times c_\ell$, respectively. Attention projections are resized by complete head blocks. Both modules retain their input and output dimensions as their internal widths change. Removed units leave no zero-filled slots, and subsequent forward and backward passes operate directly on the resized tensors with standard dense kernels.

Optimizer-state remapping. We resize the optimizer states using the same unit indices as the parameters. For Adam, we retain the first- and second-moment estimates of unaffected parameter slices and remove those belonging to the pruned unit. Both estimates are set to zero for the parameters of the cloned parent and its new copy, while the Adam step counters remain unchanged.

Training schedule. Pruning decisions can benefit from training beyond initialization (Frankle et al., 2020). We therefore train the initial compact model and accumulate active-unit scores before the first capacity transfer. Let T denote the total number of training steps, and let t_{start} and t_{end} define the reallocation window. We set the total numbers of head and FFN-channel transfer attempts, K_H and K_C, in proportion to their initial active counts H and C, respectively. For each unit type, we spread these attempts approximately uniformly across the window. Training between updates lets the model adapt to its current structure and refresh the importance estimates used for the next decision (Chen et al., 2021). At each scheduled update, we attempt one transfer of the corresponding unit type after the optimizer step, keeping the change small to limit disruption to training (Yin et al., 2023).

We stop collecting scores for each unit type after its last scheduled update. After t_{end}, we fix the layer widths and continue optimizing the weights until step T, as sparse connectivity can stabilize earlier than the weights (Mostafa & Wang, 2019). Appendix C.5 examines how increasing the number of capacity transfers affects classification performance and incremental update cost.

4 Results and Discussion

We evaluate `BARTER` against dense and sparse baselines on two image classification benchmarks, comparing classification performance, training cost, and final model size.

Datasets and models. Our from-scratch experiments use a seven-layer CCT (Hassani et al., 2021) on CIFAR-100 (Krizhevsky & Hinton, 2009) and a nine-layer ViT (Dosovitskiy, 2020) on Tiny-ImageNet (Le et al., 2015). CCT uses 32×32 inputs, while ViT uses 64×64 inputs and the small-model configuration of Gani et al. (2022). We also fine-tune the Tiny-ImageNet-trained ViT on CIFAR-100 with images resized to 64×64. We reserve 10% of each official training set for validation. Final evaluation uses the CIFAR-100 test set and the labeled Tiny-ImageNet validation set.

Baselines. We compare `BARTER` with Dense, the unstructured DST methods SET (Mocanu et al., 2018), SNFS (Dettmers & Zettlemoyer, 2019), and RigL (Evci et al., 2020), and the structured DST methods SRigL (Lasby et al., 2024) and S^2ViTE (Chen et al., 2021).We use the official implementations of the DST baselines, adapted to our model configurations. Sparse models use target densities $\rho \in \{0.1, 0.3, 0.5, 0.7\}$. Initial capacity allocation and baseline settings are described in Appendices B.1 and B.3.

Table 1: Classification performance (accuracy/macro-F1) of models trained from scratch. Best sparse results per task and density in **bold and orange**; second-best in *underlined italics*. Mean ± std (%) over three seeds.

(a) CIFAR-100 (CCT)

Dense ($\rho = 1$): Acc $73.49_{\pm 0.10}$, F1 $73.29_{\pm 0.11}$

	$\rho = 0.1$		$\rho = 0.3$		$\rho = 0.5$		$\rho = 0.7$	
Method	Acc ↑	F1 ↑	Acc ↑	F1 ↑	Acc ↑	F1 ↑	Acc ↑	F1 ↑
SET	$63.21_{\pm 0.55}$	$62.80_{\pm 0.56}$	$70.26_{\pm 0.30}$	$70.08_{\pm 0.31}$	$72.25_{\pm 0.06}$	$72.04_{\pm 0.10}$	$73.21_{\pm 0.16}$	$72.98_{\pm 0.13}$
SNFS	$63.29_{\pm 0.36}$	$62.87_{\pm 0.35}$	$69.45_{\pm 0.14}$	$69.31_{\pm 0.14}$	$70.92_{\pm 0.53}$	$70.74_{\pm 0.51}$	$72.17_{\pm 0.17}$	$71.97_{\pm 0.12}$
RigL	**$67.01_{\pm 0.17}$**	**$66.67_{\pm 0.08}$**	**$71.74_{\pm 0.14}$**	**$71.57_{\pm 0.10}$**	*$72.62_{\pm 0.21}$*	*$72.40_{\pm 0.22}$*	**$73.29_{\pm 0.08}$**	**$73.09_{\pm 0.07}$**
SRigL	$60.88_{\pm 0.82}$	$60.37_{\pm 0.85}$	$64.66_{\pm 0.44}$	$64.34_{\pm 0.42}$	$69.94_{\pm 0.54}$	$69.79_{\pm 0.53}$	$72.40_{\pm 0.22}$	$72.17_{\pm 0.20}$
S^2ViTE	*$66.64_{\pm 0.57}$*	*$66.36_{\pm 0.60}$*	$70.94_{\pm 0.27}$	$70.79_{\pm 0.27}$	$72.49_{\pm 0.43}$	$72.27_{\pm 0.44}$	$73.10_{\pm 0.40}$	$72.88_{\pm 0.42}$
BARTER	$66.13_{\pm 0.40}$	$65.79_{\pm 0.44}$	*$71.38_{\pm 0.36}$*	*$71.21_{\pm 0.32}$*	**$72.85_{\pm 0.16}$**	**$72.64_{\pm 0.14}$**	*$73.26_{\pm 0.28}$*	*$73.05_{\pm 0.28}$*

(b) Tiny-ImageNet (ViT)

Dense ($\rho = 1$): Acc $52.21_{\pm 0.29}$, F1 $51.62_{\pm 0.33}$

	$\rho = 0.1$		$\rho = 0.3$		$\rho = 0.5$		$\rho = 0.7$	
Method	Acc ↑	F1 ↑	Acc ↑	F1 ↑	Acc ↑	F1 ↑	Acc ↑	F1 ↑
SET	$43.44_{\pm 0.03}$	$42.34_{\pm 0.05}$	$51.22_{\pm 0.18}$	$50.73_{\pm 0.18}$	*$52.39_{\pm 0.63}$*	*$51.92_{\pm 0.72}$*	$52.52_{\pm 0.38}$	$52.09_{\pm 0.36}$
SNFS	*$43.72_{\pm 0.13}$*	*$42.71_{\pm 0.18}$*	**$51.49_{\pm 0.30}$**	**$50.97_{\pm 0.33}$**	$52.16_{\pm 0.36}$	$51.74_{\pm 0.39}$	$52.12_{\pm 0.29}$	$51.72_{\pm 0.33}$
RigL	**$44.13_{\pm 0.52}$**	**$43.16_{\pm 0.51}$**	$51.00_{\pm 0.14}$	$50.51_{\pm 0.14}$	$51.76_{\pm 0.44}$	$51.32_{\pm 0.47}$	$51.58_{\pm 0.34}$	$51.10_{\pm 0.30}$
SRigL	$42.06_{\pm 0.69}$	$40.86_{\pm 0.78}$	$44.29_{\pm 0.05}$	$43.28_{\pm 0.07}$	$52.10_{\pm 0.34}$	$51.61_{\pm 0.32}$	**$53.01_{\pm 0.61}$**	*$52.59_{\pm 0.62}$*
S^2ViTE	$42.52_{\pm 0.83}$	$41.14_{\pm 0.91}$	$50.98_{\pm 0.32}$	$50.44_{\pm 0.30}$	$52.16_{\pm 0.30}$	$51.72_{\pm 0.32}$	$52.68_{\pm 0.26}$	$52.23_{\pm 0.24}$
BARTER	$42.56_{\pm 0.86}$	$41.14_{\pm 0.99}$	*$51.42_{\pm 0.39}$*	*$50.81_{\pm 0.35}$*	**$52.77_{\pm 0.23}$**	**$52.34_{\pm 0.29}$**	*$53.00_{\pm 0.28}$*	**$52.64_{\pm 0.23}$**

Training settings. In the fixed-step comparisons, methods share the data split, augmentation, and optimization settings within each task and seed. GPU training uses FP32 and Adam (Kingma & Ba, 2014) with decoupled weight decay (Loshchilov & Hutter, 2017). CCT is trained for 200 epochs. ViT training and standard fine-tuning each run for 100 epochs, with three shuffled passes through the training set per epoch. For BARTER, we set the score EMA decay to 0.99 and reallocate capacity between 10% and 40% of the training steps. Detailed training settings and checkpoint-selection rules are provided in Appendix B.2.

Evaluation. We select checkpoints by validation Top-1 accuracy and report Top-1 accuracy and macro-F1 as mean ± sample standard deviation across three random seeds. For training efficiency, we report measured training time for epoch-length segments, estimated per-step time, and peak allocated GPU memory. We also report estimated training FLOPs, counting each multiply-accumulate operation as two FLOPs, and the physical parameter counts of the exported models. Measurement and counting protocols are provided in Appendix D.

4.1 Classification Performance

Table 1 shows that BARTER ranks first or second among sparse methods in both Top-1 accuracy and macro-F1 at densities of 0.3, 0.5, and 0.7 on CIFAR-100 and Tiny-ImageNet. At $\rho = 0.5$, it achieves the **highest mean Top-1 accuracy and macro-F1** among sparse methods on both tasks, with Top-1 accuracies of 72.85% and 52.77%, respectively.

Even at $\rho = 0.1$, BARTER **physically removes approximately 90% of the original attention heads and FFN channels**, while the baselines in our experiments retain full-size training tensors. Despite this reduction, its Top-1 accuracy is only 0.88 and 1.57 percentage points below RigL, the best-performing sparse baseline at this density, on CIFAR-100 and Tiny-ImageNet, respectively.

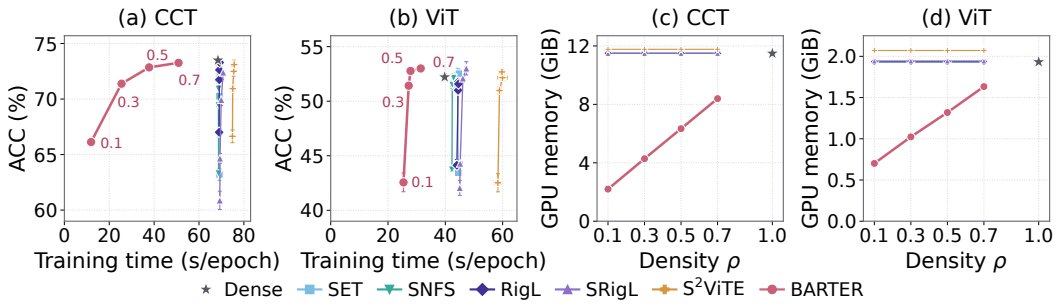

Figure 3: Accuracy, GPU training time per epoch, and peak allocated GPU memory for CCT on CIFAR-100 and ViT on Tiny-ImageNet. Colors distinguish methods. Each sparse-method curve connects four points corresponding to densities $\rho = 0.1, 0.3, 0.5, 0.7$. Dense is a single full-density reference ($\rho = 1$). Error bars show sample standard deviations.

4.2 Training Efficiency and Model Size

Figure 3 compares accuracy, GPU training time per epoch, and peak allocated GPU memory. **BARTER has the lowest mean training time and peak allocated memory among all compared methods at every tested density.** At $\rho = 0.5$, it achieves the highest mean ACC and macro-F1 among sparse methods on both tasks, while training each epoch $1.83\times$ faster than RigL on CCT and $1.60\times$ faster on ViT. The corresponding speedups over S^{2}ViTE are $2.00\times$ and $2.15\times$.

At $\rho = 0.1$, speedups over RigL reach $5.80\times$ and $1.73\times$, with peak allocated memory reduced by 80.9% and 63.8%, respectively. Schedule-weighted step-time estimates show the same efficiency advantage across all four densities (Appendix C.1).

Figure 4 compares estimated training FLOPs and exported parameter counts.

At $\rho = 0.1$, BARTER reduces estimated training FLOPs by 89.4% on CCT and 88.6% on ViT relative to Dense and all compared DST baselines. The exported models contain 0.480M and 0.373M parameters, compared with 3.783M and 2.762M for Dense, SET, SNFS, RigL, and SRigL. S^{2}ViTE reaches the same exported parameter counts as BARTER, but uses full-size tensors during training in our implementation. BARTER uses compact tensors during both training and deployment. Complete numerical results and measurement details appear in Appendices C.1 and D, respectively.

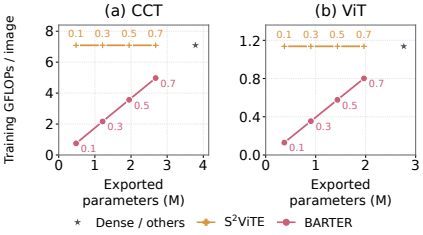

Figure 4: Estimated training GFLOPs per image versus exported parameter counts for (a) CCT and (b) ViT. Points on each colored curve correspond to densities $\rho = 0.1, 0.3, 0.5, 0.7$ from left to right. The star marks the overlapping results of Dense, SET, SNFS, RigL, and SRigL. S^{2}ViTE and BARTER have identical exported parameter counts at each density.

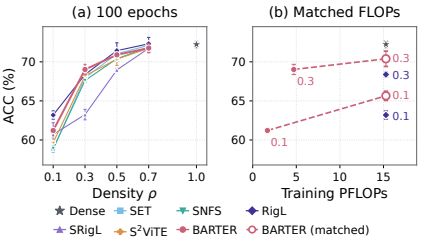

Figure 5: Fine-tuning on CIFAR-100. (a) Results after 100 epochs at sparse densities $\rho = 0.1, 0.3, 0.5, 0.7$. (b) BARTER's 100-epoch (filled circles) and compute-matched (hollow circles) results, compared with RigL at the same densities. Numeric labels indicate density; Dense is a full-density reference. Error bars show sample standard deviations.

4.3 Fine-Tuning Performance and Efficiency

Standard fine-tuning. We fine-tune the Tiny-ImageNet-pretrained ViT on CIFAR-100. Figure 5(a) compares the results after 100 epochs. **At $\rho = 0.3$, BARTER achieves the highest mean ACC and macro-F1 among sparse methods, at 69.02% and 68.81%, while using 69.2% fewer training FLOPs than Dense and all compared DST baselines.** At $\rho = 0.1$, the FLOP reduction reaches 89.0%, with ACC 1.98 percentage points below RigL.

Fine-tuning under a matched FLOP budget. We then use the saved compute to train longer at $\rho = 0.1$ and 0.3. Matching the baselines' 100-epoch budget of approximately 15.276 PFLOPs allows BARTER to train for 906.17 and 324.28 epochs in total, respectively. ACC rises from 61.21% to 65.65% at $\rho = 0.1$ and from 69.02% to 70.39% at $\rho = 0.3$; macro-F1 reaches 65.33% and 70.18%. **Under the matched compute budget, BARTER exceeds RigL, the strongest sparse baseline at both densities, by 2.46 and 2.01 percentage points in ACC.** Figure 5(b) shows this comparison. Complete results, training lengths, and cumulative FLOPs are in Appendix C.2.

4.4 On-Device Training

We benchmark CCT training on CIFAR-100 using an iPhone 13 Pro Max, comparing Dense, RigL, and BARTER. Both sparse methods use $\rho = 0.3$. Table 2 reports training time per optimizer step and the directly measured time for one epoch. **BARTER completes the measured epoch in 610.33 seconds, with speedups of 2.83× over Dense and 2.78× over RigL.** Its per-step speedups are $2.67\times$ and $2.73\times$, respectively. These results show that compact training tensors also reduce execution time on the phone. The runtime and timing protocol are detailed in Appendix D.3.

Table 2: CCT training on iPhone 13 Pro Max. RigL and BARTER use $\rho = 0.3$.

	Training time	
Method	ms/step	s/epoch
Dense	4814.77	1726.77
RigL	4927.10	1699.36
BARTER	**1804.57**	**610.33**

4.5 Ablation Studies

We examine capacity reallocation and its selection rules on CCT/CIFAR-100 at $\rho = 0.5$. Static trains the same initial compact architecture as BARTER but disables capacity reallocation. Random receiver randomizes the receiving layer and clone parent; Random donor randomizes the unit to prune; Random pair applies both randomizations (Appendix B.3).

Figure 6 shows that **BARTER achieves the highest mean ACC and macro-F1, exceeding Static by 1.05 percentage points in both metrics.** Compared with Random pair, the gains are 1.65 and 1.66 percentage points, respectively. Among the single-sided variants, Random donor incurs a larger mean performance drop than Random receiver. BARTER also has lower standard deviations in both metrics than all three randomized variants. Full numerical results are in Appendix C.3. We extend the comparison with Static to CCT and ViT across all four densities (Appendix C.4). BARTER achieves higher mean ACC and macro-F1 in every setting, with modest increases in measured training-segment time and similar peak allocated GPU memory.

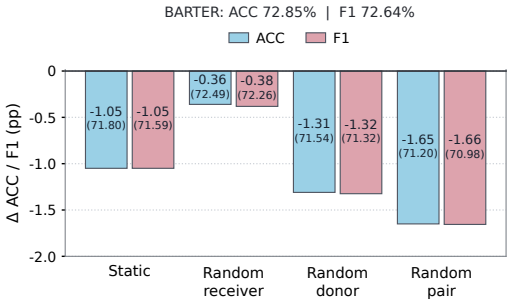

Figure 6: Ablations of capacity reallocation and selection on CCT/CIFAR-100 at $\rho = 0.5$. Bars show ACC and macro-F1 differences (variant minus BARTER) in percentage points, with variant scores (%) in parentheses. Values are means over three seeds per method.

5 Conclusion

We presented BARTER, a structured DST framework that reallocates capacity under fixed global budgets using active-unit statistics while maintaining compact dense execution. Experiments show competitive accuracy, lower peak allocated GPU memory, and speedups in GPU and smartphone training benchmarks. Matched-compute fine-tuning further improves accuracy at low densities, surpassing the strongest sparse baseline. The resulting models require no additional pruning for deployment.

Broader Impact and Future Work. BARTER supports resource-efficient training without specialized sparse kernels. Future work will explore more effective capacity allocation and extend compact training to environments with changing resource constraints.

AI USE STATEMENT

We used generative AI tools to help organize related literature during the early stages of this work and to assist with debugging during code development. We also used these tools to polish the manuscript for clarity and readability. The authors reviewed all AI-assisted content and take responsibility for the final manuscript and code.

ETHICS STATEMENT

Our experiments use the publicly available CIFAR-100 and Tiny-ImageNet benchmarks. This study does not involve recruiting human participants or collecting new personal data.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix A describe `BARTER` and its implementation. Appendix B specifies the datasets, model configurations, training settings, and baseline settings, while Appendix D details the efficiency measurement protocols. The supplementary material provides training and evaluation code, setup instructions, and a trained checkpoint for `BARTER` with ViT on Tiny-ImageNet at $\rho = 0.5$.

REFERENCES

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

T Chen, I Goodfellow, and J Shlens. Net2net: Accelerating learning via knowledge transfer. arxiv 2015. *arXiv preprint arXiv:1511.05641*, 2015.

Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in neural information processing systems*, 34:19974–19988, 2021.

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 113–123. IEEE, 2019.

Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. *Advances in neural information processing systems*, 33:18613–18624, 2020.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pp. 2943–2952. PMLR, 2020.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

Hanan Gani, Muzammal Naseer, and Mohammad Yaqub. How to train vision transformer on small-scale datasets? *arXiv preprint arXiv:2210.07240*, 2022.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

Yuezhou Hu, Kang Zhao, Weiyu Huang, Jianfei Chen, and Jun Zhu. Accelerating transformer pre-training with 2: 4 sparsity. *arXiv preprint arXiv:2404.01847*, 2024.

Peng Jiang, Lihan Hu, and Shihui Song. Exposing and exploiting fine-grained block structures for fast and accurate sparse training. *Advances in Neural Information Processing Systems*, 35: 38345–38357, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. In *International Conference on Learning Representations*, volume 2024, pp. 32982–33010, 2024.

Yann Le, Xuan Yang, et al. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL `http://arxiv.org/abs/1608.08710`.

Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*, pp. 6989–7000. PMLR, 2021.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Jiaqing Lyu, Ruijie Wang, Kangyou Bao, Yingtao Zhang, and Carlo Vittorio Cannistraci. Cannistraci-hebb training with n: M semi-structured sparsity for pre-training and re-training. In *The Third Conference on Parsimony and Learning (Proceedings Track)*, 2026.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 11256–11264. IEEE, 2019.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Mohammad Mozaffari, Amir Yazdanbakhsh, Zhao Zhang, and Maryam Mehri Dehnavi. Slope: Double-pruned sparse plus lazy low-rank adapter pretraining of llms. In *International Conference on Learning Representations*, volume 2025, pp. 20656–20685, 2025.

Mahdi Nikdan, Tommaso Pegolotti, Eugenia Iofinova, Eldar Kurtic, and Dan Alistarh. Sparseprop: Efficient sparse backpropagation for faster training of neural networks at the edge. In *International Conference on Machine Learning*, pp. 26215–26227. PMLR, 2023.

Abhishek Tyagi, Arjun Iyer, William H Renninger, Christopher Kanan, and Yuhao Zhu. Dynamic sparse training of diagonally sparse networks. *arXiv preprint arXiv:2506.11449*, 2025.

Nasib Ullah, Jinbin Zhang, Jean Lucien Randrianantenaina, Erik Schultheis, and Rohit Babbar. Haste: Hardware-aware dynamic sparse training for large output spaces. *arXiv preprint arXiv:2606.01117*, 2026.

Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. *Advances in neural information processing systems*, 32, 2019.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. *Advances in neural information processing systems*, 33: 22373–22383, 2020.

Lu Yin, Gen Li, Meng Fang, Li Shen, Tianjin Huang, Zhangyang Wang, Vlado Menkovski, Xiaolong Ma, Mykola Pechenizkiy, Shiwei Liu, et al. Dynamic sparsity is channel-level sparsity learner. *Advances in Neural Information Processing Systems*, 36:67993–68012, 2023.

Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34:20838–20850, 2021.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Yingtao Zhang, Diego Cerretti, Jialin Zhao, Wenjing Wu, Ziheng Liao, Umberto Michieli, and Carlo Vittorio Cannistraci. Brain network science modelling of sparse neural networks enables transformers and llms to perform as fully connected. *Advances in Neural Information Processing Systems*, 38:48241–48277, 2026.

Algorithm 1 Training with BARTER

Require: Initial model, density ρ, training data, and T optimizer steps
Construct the compact model with global budgets H, C
Initialize Adam, zero-valued score EMAs, and the update schedules
for $t = 1, \ldots, T$ **do**
    Run forward and backward passes on an augmented minibatch
    Update active-unit EMAs while score collection is enabled
    Update model parameters using the task's optimizer and learning-rate schedule
    for each unit type scheduled for reallocation at t **do**
        Select the highest-scoring eligible receiving layer r
        if r exists and an eligible donor exists outside r **then**
            Select the donor with minimum pruning score and the parent in r with maximum growth score
            Clone the parent in r, split its outgoing weights, and prune the donor
            Resize parameters and Adam states; update scores and cooldowns
        end if
    end for
end for
return the final model and the checkpoint selected by validation ACC

A Algorithm and Implementation Details

A.1 Complete Training Algorithm

Algorithm 1 summarizes training. Density ρ applies to the attention and FFN weight matrices; embeddings, normalization layers, and the classifier remain outside the pruning pool. Table 3 lists the integer head and channel budgets. We distribute each budget as evenly as possible across layers, assign remainders randomly, and randomly select the retained units within each layer. Their parameters form the initial compact model.

Selection and scheduling. We use the selection rules in Section 3.2, with EMA decay $\beta = 0.99$. A receiving layer must contain an active unit and remain below its full-density width. Donors come from other layers, with at least one FFN channel retained per layer. Reallocation takes place between $0.1T$ and $0.4T$, after optimizer updates. FFN proposals are evenly spaced over this window; head proposals use interval midpoints. Clone parents and their copies are excluded from donor selection for two typical FFN proposal intervals or one head interval. Score collection stops after each unit type's final scheduled proposal.

Reallocation budget. For the main experiments, we use a limited number of whole-unit transfers to evaluate whether capacity reallocation can improve classification performance in compact models. We scale the scheduled head and FFN-channel transfer counts with their active-unit budgets using reference ratios of $2/34$ and $60/1067$, respectively, corresponding to the CCT configuration at $\rho = 0.3$. Counts are rounded to the nearest integer, with at least one transfer scheduled for each feasible unit type. All scheduled transfers were completed in these runs. Table 3 reports the executed transfer counts alongside the global budgets and initial per-layer counts. ViT uses the same reference settings for training from scratch and fine-tuning.

Whole-unit transfers can change a substantial fraction of an individual layer's capacity. At $\rho = 0.1$, CCT and ViT initially contain only one or two heads per layer, so one head represents 50% or 100% of a layer's initial head count. FFN channels allow finer adjustments, with 20 to 142 channel transfers per run in the main-experiment settings. Appendix C.5 examines how increasing these transfer counts affects classification performance and estimated incremental topology-update cost.

A.2 Compact Model Implementation

Attention projections store only active head blocks, and FFN matrices store only active channel rows and columns. The residual width and per-head dimension remain fixed. Cloning copies the parent's incoming weights and biases, then splits its outgoing weights as in Eq. (6), using Gaussian perturbations with standard deviation $0.02 \max(\mathrm{RMS}(v), 10^{-8})$ for outgoing block v. The receiver is expanded before the donor is pruned; each completed transfer preserves the global head and

Table 3: Global capacity budgets and reference transfer counts for the main experiments. Initial per-layer counts are shown in parentheses. All scheduled head and FFN-channel transfers, K_H and K_C, were executed in these runs.

Model	ρ	Heads H	FFN channels C	K_H	K_C
CCT	0.1	11 (1–2)	371 (53)	1	21
	0.3	34 (4–5)	1,067 (152–153)	2	60
	0.5	56 (8)	1,795 (256–257)	3	101
	0.7	78 (11–12)	2,524 (360–361)	5	142
ViT	0.1	11 (1–2)	359 (39–40)	1	20
	0.3	33 (3–4)	1,033 (114–115)	2	58
	0.5	54 (6)	1,739 (193–194)	3	98
	0.7	76 (8–9)	2,413 (268–269)	4	136

Table 4: Full-density model configurations. Fine-tuning retains the ViT backbone.

Setting	CCT	ViT
Input resolution	32×32	64×64
Transformer layers	7	9
Embedding dimension	256	192
Heads per layer	16	12
Head dimension	16	16
FFN hidden dimension	512	384
Tokens	256	$64 + 1$
Pooling	Attention	Class token

channel budgets. Zero-head attention skips attention products but retains the output-projection bias and residual connection.

Adam states follow the parameter resizing: pruned slices are removed, unchanged slices are retained, and first- and second-moment slices for both the parent and clone are reset to zero. Optimizer step counters are preserved. The parent and clone each inherit half of the parent's previous pruning and growth EMAs. Persistent unit IDs track cooldowns across resizing. Training and evaluation use the resulting compact tensors directly.

B Experimental Details

B.1 Datasets and Model Configurations

For CIFAR-100, we use 45,000 images for training, 5,000 for validation, and the official 10,000-image test set. CCT uses a seeded random split; ViT fine-tuning uses a class-stratified split. For Tiny-ImageNet, we use a class-stratified 90,000/10,000 split of the official training set and evaluate on the official labeled validation set. Within each task and seed, all methods share the same split. Table 4 lists the architectures. CCT uses a 3×3 convolutional tokenizer followed by stride-2 max pooling; ViT uses 8×8 patches. Both use learned positional embeddings. Fine-tuning resizes CIFAR-100 images to 64×64 by bicubic interpolation and replaces ViT's classifier with a 100-class head.

We evaluate $\rho \in \{0.1, 0.3, 0.5, 0.7\}$, with Dense as a full-density reference. Within each task, density, and seed, S^2ViTE and `BARTER` share the initial structured allocation. This allocation also sets the active-parameter target for the weight-sparse baselines. For training from scratch, Static uses the same initial compact architecture as `BARTER`. Baseline and ablation settings are given in Appendix B.3.

Table 5: Training settings. Fine-tuning epochs refer to the initial phase; warmup is a fraction of optimizer steps.

Setting	CCT	ViT	ViT fine-tuning
Epochs	200	100	100
Batch size	128	256	256
Steps per epoch	351	1,054	527
Peak learning rate	3×10^{-4}	2×10^{-3}	10^{-3}
Minimum learning rate	0	10^{-6}	10^{-6}
Weight decay	0.05	0.05	0.05
Warmup fraction	0.05	0.10	0.10
Label smoothing	0.1	0.1	0.1
Gradient clipping norm	1.0	None	None
Attention dropout	0.1	0	0
Maximum stochastic depth	0.1	0.1	0.1
Mixup α	0.2	1.0	1.0
CutMix α	Disabled	1.0	1.0
Batch mixing probability	1.0	0.5	0.5

B.2 Training Settings

Optimization. Table 5 summarizes the training settings. All methods use label-smoothed cross-entropy and Adam with explicitly implemented decoupled weight decay, applied after the Adam update for CCT and before it for ViT training and fine-tuning. Decay includes biases and normalization parameters, and is restricted to active entries in masked implementations. Learning rates use linear warmup followed by cosine decay (Loshchilov & Hutter, 2016). All tasks use FP32 without automatic mixed precision; ViT training and fine-tuning also disable TF32.

One CCT epoch traverses the training set once. ViT training and fine-tuning use three shuffled passes per epoch, dropping the final incomplete minibatch; their 100-epoch schedules therefore correspond to approximately 300 data passes. Evaluation uses batch size 256. Embedding, FFN, and projection dropout are zero, while stochastic-depth rates increase linearly across layers.

Data augmentation. All tasks use random cropping with four-pixel padding and horizontal flipping. CCT additionally uses RandAugment (Cubuk et al., 2020) with two operations and magnitude 9, together with Mixup (Zhang et al., 2017). ViT training and fine-tuning use ImageNet AutoAugment (Cubuk et al., 2019) and random erasing with probability 0.25. When batch mixing is enabled, ViT chooses Mixup or CutMix (Yun et al., 2019) with equal probability. Fine-tuning retains the source model's Tiny-ImageNet normalization.

Fine-tuning protocol. For each seed, all methods start from the same validation-selected dense Tiny-ImageNet checkpoint, with a newly initialized classifier and optimizer. Sparse methods inherit pretrained weights at their selected positions. After the initial 100 epochs, `BARTER` at $\rho = 0.1$ and 0.3 continues from the endpoint checkpoint with its topology fixed and score collection disabled. Adam moments and step counters are retained, while a separate warmup/cosine schedule restarts over the remaining compute budget using the same learning-rate settings. Training stops at the baselines' cumulative forward-plus-backward MAC budget, allowing a final partial epoch. The counting protocol is given in Appendix D.2.

B.3 Baseline and Ablation Settings

Baseline implementations. We apply the official implementations of SET (Mocanu et al., 2018), SNFS (Dettmers & Zettlemoyer, 2019), RigL (Evci et al., 2020), SRigL (Lasby et al., 2024), and S^2ViTE (Chen et al., 2021) to the CCT and ViT configurations used in our experiments, following the common training recipes in Appendix B.2. We benchmark all methods using the GPU timing and memory protocol in Appendix D.1.

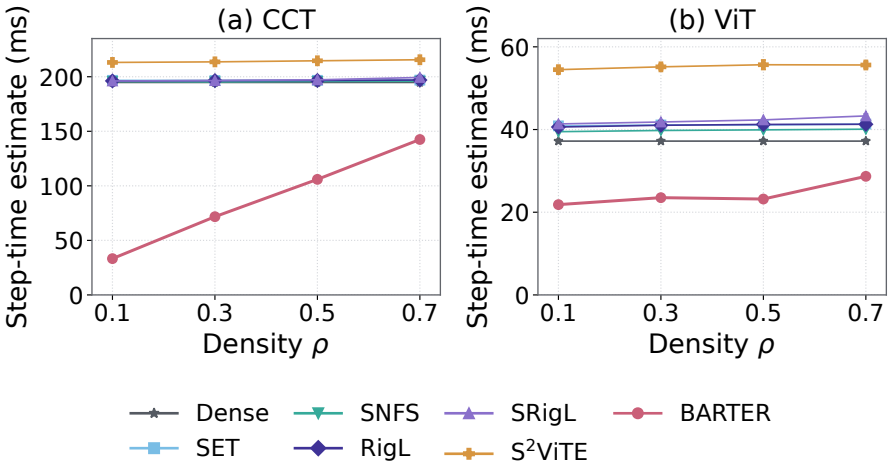

Figure 7: Schedule-weighted training step-time estimates for CCT on CIFAR-100 and ViT on Tiny-ImageNet. Error bars show sample standard deviations.

Static compact baseline. We evaluate Static on CCT/CIFAR-100 and ViT/Tiny-ImageNet at $\rho \in \{0.1, 0.3, 0.5, 0.7\}$. Static follows the same training recipe as BARTER but keeps its initial attention-head and FFN-channel allocation fixed throughout training. All retained weights remain trainable, and forward and backward computation use compact tensors. Static uses the same efficiency-measurement protocol as BARTER (Appendix D). Appendix C.4 reports the complete classification and efficiency results.

Selection ablations. We evaluate the three randomized variants on CCT/CIFAR-100 at $\rho = 0.5$, retaining the original model configuration, initial capacity budget, training recipe, and transfer-attempt schedule. Random receiver samples uniformly from layers with at least one active unit and room to grow, then samples a clone parent uniformly from the active units in that layer. Random donor samples uniformly from all eligible active units outside the receiving layer, subject to the same donor cooldown and minimum-width constraints as BARTER. Random pair applies both randomizations. The remaining selection operations use the original scoring rules; score tracking, cloning, and optimizer-state handling are unchanged.

C Additional Results

C.1 Complete Efficiency Results

Tables 6 and 8 report GPU training time per epoch, peak allocated memory, and schedule-weighted step-time estimates for BARTER, Dense, and the five DST baselines. Tables 7 and 9 give estimated training FLOPs and exported parameter counts. The measurement procedures are described in Appendix D; comparisons with Static are reported in Appendix C.4.

Training step time. Figure 7 shows that BARTER has the lowest mean step-time estimate on both tasks at every tested density. At $\rho = 0.5$, its estimates are 105.94 ms for CCT and 23.19 ms for ViT, compared with 196.59 ms and 41.23 ms for RigL. These correspond to $1.86\times$ and $1.78\times$ speedups, respectively.

C.2 Complete Fine-Tuning Results

Table 10 reports ACC, macro-F1, and cumulative training FLOPs for all methods after 100 epochs of fine-tuning on CIFAR-100. Results are means and sample standard deviations over seeds 1, 2, and 4. Training FLOPs include the accumulated forward and backward MAC counts, with two FLOPs per MAC; one PFLOP equals 10^{15} FLOPs.

Table 6: GPU training efficiency for CCT on CIFAR-100. Values are mean ± sample standard deviation. Dense is a full-density reference repeated across columns.

Method	$\rho = 0.1$	$\rho = 0.3$	$\rho = 0.5$	$\rho = 0.7$
Measured training time (s/epoch)				
Dense	68.45 ± 0.01	68.45 ± 0.01	68.45 ± 0.01	68.45 ± 0.01
SET	69.12 ± 0.04	69.05 ± 0.03	69.11 ± 0.02	69.18 ± 0.05
SNFS	68.73 ± 0.03	68.65 ± 0.03	68.72 ± 0.05	68.80 ± 0.07
RigL	68.97 ± 0.05	69.05 ± 0.10	69.09 ± 0.02	69.24 ± 0.05
SRigL	69.27 ± 0.05	69.36 ± 0.07	69.90 ± 0.02	70.78 ± 0.22
S^2ViTE	74.90 ± 0.15	75.03 ± 0.05	75.45 ± 0.17	75.70 ± 0.24
BARTER	11.90 ± 0.06	25.48 ± 0.02	37.78 ± 0.02	50.88 ± 0.11
Peak allocated GPU memory (GiB)				
Dense	11.489 ± 0.000	11.489 ± 0.000	11.489 ± 0.000	11.489 ± 0.000
SET	11.492 ± 0.000	11.492 ± 0.000	11.492 ± 0.000	11.492 ± 0.000
SNFS	11.492 ± 0.000	11.492 ± 0.000	11.492 ± 0.000	11.492 ± 0.000
RigL	11.493 ± 0.000	11.493 ± 0.000	11.493 ± 0.000	11.493 ± 0.000
SRigL	11.493 ± 0.000	11.493 ± 0.000	11.493 ± 0.000	11.493 ± 0.000
S^2ViTE	11.770 ± 0.000	11.770 ± 0.000	11.770 ± 0.000	11.770 ± 0.000
BARTER	2.197 ± 0.034	4.280 ± 0.011	6.335 ± 0.002	8.394 ± 0.031
Schedule-weighted training step estimate (ms/step)				
Dense	194.78 ± 0.06	194.78 ± 0.06	194.78 ± 0.06	194.78 ± 0.06
SET	196.71 ± 0.07	196.53 ± 0.17	196.71 ± 0.13	196.91 ± 0.10
SNFS	195.60 ± 0.08	195.27 ± 0.04	195.67 ± 0.19	195.84 ± 0.13
RigL	196.18 ± 0.17	196.47 ± 0.20	196.59 ± 0.17	196.98 ± 0.16
SRigL	196.12 ± 0.14	196.45 ± 0.19	197.19 ± 0.07	199.32 ± 0.42
S^2ViTE	213.12 ± 0.46	213.64 ± 0.08	214.63 ± 0.48	215.59 ± 0.56
BARTER	33.27 ± 0.21	71.71 ± 0.20	105.94 ± 0.05	142.48 ± 0.29

Table 7: Estimated training computation and exported model size for CCT on CIFAR-100. All sample standard deviations are zero at the reported precision. Dense is a full-density reference repeated across columns.

Method	$\rho = 0.1$	$\rho = 0.3$	$\rho = 0.5$	$\rho = 0.7$
Estimated training GFLOPs/image				
Dense	7.089840	7.089840	7.089840	7.089840
SET	7.089840	7.089840	7.089840	7.089840
SNFS	7.089840	7.089840	7.089840	7.089840
RigL	7.089840	7.089840	7.089840	7.089840
SRigL	7.089840	7.089840	7.089840	7.089840
S^2ViTE	7.089840	7.089840	7.089840	7.089840
BARTER	0.750408	2.165988	3.568980	4.972764
Exported parameters (M)				
Dense	3.7833	3.7833	3.7833	3.7833
SET	3.7833	3.7833	3.7833	3.7833
SNFS	3.7833	3.7833	3.7833	3.7833
RigL	3.7833	3.7833	3.7833	3.7833
SRigL	3.7833	3.7833	3.7833	3.7833
S^2ViTE	0.4802	1.2141	1.9480	2.6824
BARTER	0.4802	1.2141	1.9480	2.6824

Table 11 compares BARTER's 100-epoch and compute-matched runs with RigL at $\rho = 0.1$ and 0.3. The reference budget is 15.27612 PFLOPs, used by Dense and every compared DST baseline over 100 epochs. The two compute-matched runs use 15.27616 and 15.27618 PFLOPs over 906.17 and 324.28 total epochs, respectively. Relative to 100-epoch BARTER, ACC increases by 4.44 and

Table 8: GPU training efficiency for ViT on Tiny-ImageNet. Values are mean ± sample standard deviation. Dense is a full-density reference repeated across columns.

Method	$\rho = 0.1$	$\rho = 0.3$	$\rho = 0.5$	$\rho = 0.7$
Measured training time (s/epoch)				
Dense	39.78 ± 0.48	39.78 ± 0.48	39.78 ± 0.48	39.78 ± 0.48
SET	44.50 ± 0.92	44.32 ± 0.43	44.35 ± 0.34	44.76 ± 0.51
SNFS	42.31 ± 0.19	42.44 ± 0.40	42.64 ± 0.31	42.89 ± 0.26
RigL	44.03 ± 0.25	44.46 ± 0.29	44.46 ± 0.54	44.58 ± 0.45
SRigL	45.04 ± 0.39	45.19 ± 0.63	46.13 ± 0.49	47.32 ± 0.40
S^2ViTE	58.36 ± 1.28	58.90 ± 0.39	59.96 ± 1.63	59.77 ± 1.22
BARTER	25.40 ± 0.55	27.22 ± 0.29	27.86 ± 1.12	31.41 ± 0.07
Peak allocated GPU memory (GiB)				
Dense	1.932 ± 0.000	1.932 ± 0.000	1.932 ± 0.000	1.932 ± 0.000
SET	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000
SNFS	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000
RigL	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000	1.934 ± 0.000
SRigL	1.944 ± 0.000	1.944 ± 0.000	1.944 ± 0.000	1.944 ± 0.000
S^2ViTE	2.071 ± 0.000	2.071 ± 0.000	2.071 ± 0.000	2.071 ± 0.000
BARTER	0.701 ± 0.001	1.023 ± 0.003	1.319 ± 0.005	1.633 ± 0.001
Schedule-weighted training step estimate (ms/step)				
Dense	37.20 ± 0.45	37.20 ± 0.45	37.20 ± 0.45	37.20 ± 0.45
SET	40.88 ± 0.20	41.10 ± 0.40	41.12 ± 0.29	41.31 ± 0.41
SNFS	39.48 ± 0.20	39.77 ± 0.09	39.92 ± 0.02	40.08 ± 0.21
RigL	40.61 ± 0.36	41.03 ± 0.16	41.23 ± 0.16	41.27 ± 0.31
SRigL	41.35 ± 0.21	41.82 ± 0.42	42.33 ± 0.23	43.29 ± 0.48
S^2ViTE	54.48 ± 0.84	55.16 ± 0.71	55.67 ± 1.13	55.61 ± 0.83
BARTER	21.84 ± 0.57	23.52 ± 0.33	23.19 ± 0.18	28.68 ± 0.09

Table 9: Estimated training computation and exported model size for ViT on Tiny-ImageNet. All sample standard deviations are zero at the reported precision. Dense is a full-density reference repeated across columns.

Method	$\rho = 0.1$	$\rho = 0.3$	$\rho = 0.5$	$\rho = 0.7$
Estimated training GFLOPs/image				
Dense	1.137138	1.137138	1.137138	1.137138
SET	1.137138	1.137138	1.137138	1.137138
SNFS	1.137138	1.137138	1.137138	1.137138
RigL	1.137138	1.137138	1.137138	1.137138
SRigL	1.137138	1.137138	1.137138	1.137138
S^2ViTE	1.137138	1.137138	1.137138	1.137138
BARTER	0.129786	0.354006	0.577410	0.801624
Exported parameters (M)				
Dense	2.7619	2.7619	2.7619	2.7619
SET	2.7619	2.7619	2.7619	2.7619
SNFS	2.7619	2.7619	2.7619	2.7619
RigL	2.7619	2.7619	2.7619	2.7619
SRigL	2.7619	2.7619	2.7619	2.7619
S^2ViTE	0.3730	0.9039	1.4347	1.9656
BARTER	0.3730	0.9039	1.4347	1.9656

1.37 percentage points, and macro-F1 by 4.65 and 1.37 percentage points. At the matched budget, BARTER has higher mean ACC and macro-F1 than every compared sparse baseline at the same density.

Table 10: Fine-tuning results after 100 epochs. ACC and macro-F1 are percentages (mean ± sample standard deviation over three seeds). Compute is cumulative forward-plus-backward training PFLOPs.

Method	Density	ACC (%)	F1 (%)	PFLOPs
Dense	1.0	72.23 ± 0.42	72.01 ± 0.45	15.27612
SET	0.1	58.81 ± 0.42	58.26 ± 0.48	15.27612
SNFS	0.1	58.89 ± 0.26	58.25 ± 0.29	15.27612
RigL	0.1	63.19 ± 0.53	62.80 ± 0.56	15.27612
SRigL	0.1	60.70 ± 1.51	60.27 ± 1.55	15.27612
S^2ViTE	0.1	59.66 ± 0.74	59.01 ± 0.74	15.27612
BARTER	0.1	61.21 ± 0.12	60.68 ± 0.17	1.68580
SET	0.3	67.96 ± 0.22	67.71 ± 0.24	15.27612
SNFS	0.3	67.61 ± 0.21	67.36 ± 0.24	15.27612
RigL	0.3	68.38 ± 0.31	68.10 ± 0.34	15.27612
SRigL	0.3	63.25 ± 0.67	62.91 ± 0.69	15.27612
S^2ViTE	0.3	68.22 ± 0.28	68.02 ± 0.27	15.27612
BARTER	0.3	69.02 ± 0.66	68.81 ± 0.72	4.71074
SET	0.5	71.04 ± 0.60	70.82 ± 0.62	15.27612
SNFS	0.5	70.40 ± 0.33	70.11 ± 0.33	15.27612
RigL	0.5	71.43 ± 1.01	71.19 ± 1.05	15.27612
SRigL	0.5	68.96 ± 0.22	68.70 ± 0.26	15.27612
S^2ViTE	0.5	70.34 ± 0.80	70.14 ± 0.81	15.27612
BARTER	0.5	70.90 ± 0.18	70.69 ± 0.19	7.72472
SET	0.7	72.11 ± 0.42	71.89 ± 0.40	15.27612
SNFS	0.7	71.88 ± 0.42	71.69 ± 0.40	15.27612
RigL	0.7	72.31 ± 0.82	72.09 ± 0.85	15.27612
SRigL	0.7	71.73 ± 0.50	71.53 ± 0.49	15.27612
S^2ViTE	0.7	71.73 ± 0.49	71.52 ± 0.44	15.27612
BARTER	0.7	71.76 ± 0.60	71.58 ± 0.61	10.74966

Table 11: Fine-tuning with additional compute-matched training. RigL is the strongest sparse baseline at both densities. ACC and macro-F1 are percentages (mean ± sample standard deviation).

Method	Density	Epochs	PFLOPs	ACC (%)	F1 (%)
RigL	0.1	100.00	15.27612	63.19 ± 0.53	62.80 ± 0.56
BARTER	0.1	100.00	1.68580	61.21 ± 0.12	60.68 ± 0.17
BARTER	0.1	906.17	15.27616	65.65 ± 0.59	65.33 ± 0.62
RigL	0.3	100.00	15.27612	68.38 ± 0.31	68.10 ± 0.34
BARTER	0.3	100.00	4.71074	69.02 ± 0.66	68.81 ± 0.72
BARTER	0.3	324.28	15.27618	70.39 ± 0.98	70.18 ± 0.95

C.3 Complete Ablation Results

Table 12 reports the scores underlying Figure 6. The BARTER reference follows Table 1. Variant definitions are given in Appendix B.3.

C.4 Comparison with Static Compact Training

We compare BARTER with Static across all four densities on CCT/CIFAR-100 and ViT/Tiny-ImageNet. Both start from the same compact architecture and use the same training settings and efficiency-measurement protocol; Static keeps its capacity allocation fixed.

Classification performance. Table 13 shows that BARTER achieves higher mean ACC and macro-F1 at every density on both tasks. ACC gains range from 0.27 to 2.97 percentage points on CCT and from 0.34 to 0.63 on ViT. The BARTER results are reproduced from Table 1.

Table 12: Ablations on CCT/CIFAR-100 at $\rho = 0.5$. ACC and macro-F1 (%) are reported as mean $\pm$ sample standard deviation over three seeds per method. Best means are **bold and orange**.

Method	ACC	F1
Static	$71.80_{\pm 0.14}$	$71.59_{\pm 0.16}$
Random receiver	$72.49_{\pm 0.29}$	$72.26_{\pm 0.30}$
Random donor	$71.54_{\pm 0.75}$	$71.32_{\pm 0.75}$
Random pair	$71.20_{\pm 0.65}$	$70.98_{\pm 0.65}$
BARTER	$\mathbf{72.85}_{\pm 0.16}$	$\mathbf{72.64}_{\pm 0.14}$

Table 13: Static compact training versus BARTER on the two from-scratch tasks. ACC and macro-F1 (%) are mean $\pm$ sample standard deviation over three seeds per method. The higher mean in each comparison is **bold and orange**.

	Static		BARTER	
ρ	ACC	Macro-F1	ACC	Macro-F1
CCT/CIFAR-100				
0.1	$63.16_{\pm 0.27}$	$62.73_{\pm 0.25}$	$\mathbf{66.13}_{\pm 0.40}$	$\mathbf{65.79}_{\pm 0.44}$
0.3	$70.43_{\pm 0.13}$	$70.27_{\pm 0.15}$	$\mathbf{71.38}_{\pm 0.36}$	$\mathbf{71.21}_{\pm 0.32}$
0.5	$71.80_{\pm 0.14}$	$71.59_{\pm 0.16}$	$\mathbf{72.85}_{\pm 0.16}$	$\mathbf{72.64}_{\pm 0.14}$
0.7	$72.99_{\pm 0.46}$	$72.75_{\pm 0.48}$	$\mathbf{73.26}_{\pm 0.28}$	$\mathbf{73.05}_{\pm 0.28}$
ViT/Tiny-ImageNet				
0.1	$42.15_{\pm 0.81}$	$40.77_{\pm 0.93}$	$\mathbf{42.56}_{\pm 0.86}$	$\mathbf{41.14}_{\pm 0.99}$
0.3	$51.08_{\pm 0.43}$	$50.54_{\pm 0.42}$	$\mathbf{51.42}_{\pm 0.39}$	$\mathbf{50.81}_{\pm 0.35}$
0.5	$52.22_{\pm 0.67}$	$51.82_{\pm 0.69}$	$\mathbf{52.77}_{\pm 0.23}$	$\mathbf{52.34}_{\pm 0.29}$
0.7	$52.37_{\pm 0.28}$	$51.94_{\pm 0.31}$	$\mathbf{53.00}_{\pm 0.28}$	$\mathbf{52.64}_{\pm 0.23}$

Training efficiency. Table 14 reports the corresponding execution costs. BARTER achieves higher mean ACC and macro-F1 than Static at every tested density on both tasks, with modest increases in measured training-segment time and similar peak allocated GPU memory under the same measurement protocol.

Compute and model size. Across both tasks and all tested densities, BARTER achieves higher mean ACC and macro-F1 than Static with the same exported parameter counts and nearly identical estimated training FLOPs per image (Table 15).

C.5 Increasing the Number of Capacity Transfers

We further examine whether additional capacity transfers improve classification performance on ViT/Tiny-ImageNet at $\rho = 0.5$. The $1\times$ reference setting uses three head transfers and 98 FFN-channel transfers, matching the main experiments. We increase both counts by factors of two, four, and six while retaining the same initial capacity allocation, global head and FFN-channel budgets, and 100-epoch training recipe. All settings follow the same reallocation rules within the window from $0.1T$ to $0.4T$, with one unit transferred at each scheduled update.

Classification performance. Table 16 shows higher mean ACC and macro-F1 at each larger tested transfer count. Increasing the counts from $1\times$ to $6\times$ raises mean ACC from 52.77% to 53.49% and mean macro-F1 from 52.34% to 53.11%. These gains of 0.72 and 0.77 percentage points come from additional reallocation under the same active-unit budgets.

Update cost. Table 17 reports schedule-weighted estimates of incremental topology-update cost, following Appendix D.1. From $1\times$ to $6\times$, the amortized increment increases from 0.019 to 0.117 ms per optimizer step, corresponding to cumulative estimates of 1.989 and 12.321 seconds over

Table 14: GPU training efficiency of Static and BARTER. Training time is measured over an epoch-length segment; step time is a schedule-weighted estimate. Values are mean ± sample standard deviation. Both methods use the same hardware, batch size, segment length, and measurement protocol.

ρ	Method	Training time (s/segment)	Step time (ms/step)	Peak memory (GiB)
		CCT/CIFAR-100		
0.1	Static	$11.53_{\pm 0.03}$	$32.69_{\pm 0.05}$	$2.222_{\pm 0.000}$
0.1	BARTER	$11.90_{\pm 0.06}$	$33.27_{\pm 0.21}$	$2.197_{\pm 0.034}$
0.3	Static	$25.18_{\pm 0.03}$	$71.49_{\pm 0.02}$	$4.263_{\pm 0.001}$
0.3	BARTER	$25.48_{\pm 0.02}$	$71.71_{\pm 0.20}$	$4.280_{\pm 0.011}$
0.5	Static	$36.90_{\pm 0.08}$	$104.92_{\pm 0.29}$	$6.296_{\pm 0.000}$
0.5	BARTER	$37.78_{\pm 0.02}$	$105.94_{\pm 0.05}$	$6.335_{\pm 0.002}$
0.7	Static	$49.70_{\pm 0.06}$	$141.23_{\pm 0.23}$	$8.353_{\pm 0.031}$
0.7	BARTER	$50.88_{\pm 0.11}$	$142.48_{\pm 0.29}$	$8.394_{\pm 0.031}$
		ViT/Tiny-ImageNet		
0.1	Static	$24.49_{\pm 0.58}$	$21.72_{\pm 0.07}$	$0.720_{\pm 0.000}$
0.1	BARTER	$25.40_{\pm 0.55}$	$21.84_{\pm 0.57}$	$0.701_{\pm 0.001}$
0.3	Static	$25.65_{\pm 0.80}$	$22.73_{\pm 0.75}$	$0.989_{\pm 0.000}$
0.3	BARTER	$27.22_{\pm 0.29}$	$23.52_{\pm 0.33}$	$1.023_{\pm 0.003}$
0.5	Static	$26.91_{\pm 0.30}$	$22.38_{\pm 0.25}$	$1.252_{\pm 0.000}$
0.5	BARTER	$27.86_{\pm 1.12}$	$23.19_{\pm 0.18}$	$1.319_{\pm 0.005}$
0.7	Static	$30.53_{\pm 0.15}$	$28.38_{\pm 0.38}$	$1.541_{\pm 0.000}$
0.7	BARTER	$31.41_{\pm 0.07}$	$28.68_{\pm 0.09}$	$1.633_{\pm 0.001}$

Table 15: Estimated training compute and exported model size. Training GFLOPs per image are rounded to three decimals; parameter counts use four decimals. Reported standard deviations are zero at these precisions. Static parameter counts use `last.pt`; BARTER counts use the validation-selected `best.pt`.

	Training GFLOPs/image		Parameters (M)	
ρ	Static	BARTER	Static	BARTER
		CCT/CIFAR-100		
0.1	0.750	0.750	0.4802	0.4802
0.3	2.166	2.166	1.2141	1.2141
0.5	3.569	3.569	1.9480	1.9480
0.7	4.973	4.973	2.6824	2.6824
		ViT/Tiny-ImageNet		
0.1	0.130	0.130	0.3730	0.3730
0.3	0.354	0.354	0.9039	0.9039
0.5	0.577	0.577	1.4347	1.4347
0.7	0.802	0.802	1.9656	1.9656

the 100-epoch schedule. The accuracy gains at $6\times$ therefore accompany an estimated additional topology-update cost of 10.332 seconds relative to the reference setting.

Table 16: Effect of increasing capacity-transfer counts on ViT/Tiny-ImageNet at $\rho = 0.5$. ACC and macro-F1 are mean $\pm$ sample standard deviation over seeds 1, 2, and 4. Executed transfer counts and classification scores are taken from the same validation-selected checkpoints. Best means are **bold and orange**.

Multiplier	Head transfers	FFN transfers	ACC (%)	Macro-F1 (%)
1× (reference)	3	98	$52.77_{\pm 0.23}$	$52.34_{\pm 0.29}$
2×	6	196	$53.15_{\pm 0.89}$	$52.69_{\pm 0.94}$
4×	12	392	$53.30_{\pm 0.60}$	$52.92_{\pm 0.66}$
6×	18	588	$\mathbf{53.49}_{\pm 0.48}$	$\mathbf{53.11}_{\pm 0.59}$

Table 17: Estimated incremental topology-update cost for the settings in Table 16. Values are mean $\pm$ sample standard deviation across three seeds. Per-step costs are averaged over all optimizer steps; cumulative costs cover the 100-epoch schedule.

	Incremental topology-update cost	
Multiplier	ms/step	s/schedule
1× (reference)	$0.019_{\pm 0.001}$	$1.989_{\pm 0.069}$
2×	$0.038_{\pm 0.001}$	$4.051_{\pm 0.080}$
4×	$0.077_{\pm 0.003}$	$8.067_{\pm 0.362}$
6×	$0.117_{\pm 0.005}$	$12.321_{\pm 0.494}$

D Efficiency Measurement Protocol

D.1 GPU Timing and Memory

We benchmark the validation-selected checkpoints using their stored tensor shapes. For each method and density, we replay one epoch of its training schedule: 351 steps with batch size 128 for CCT, and 1,054 steps with batch size 256 for ViT. These segments process 44,928 and 269,824 images, respectively; a ViT epoch follows the three-pass convention used in training. The replays use the schedules at epoch 50 for CCT and epoch 25 for ViT.

After warmup, we synchronize the GPU and time the full training segment, synchronizing again before stopping the timer. The interval includes data loading, device transfer, augmentation, forward and backward passes, optimizer steps, score collection, and scheduled topology updates. We reset the peak-memory counter after warmup and record the maximum GPU memory allocated during the same interval.

We estimate average step time separately. Ordinary training steps are grouped by their score-collection settings, and collection or topology-update windows are timed as separate segments. We weight these timings by their frequency in the training schedule; an update window uses the nearest measured window of the same type. Measurements are aggregated within each seed before reporting the mean and sample standard deviation across seeds. Speedups are ratios of the reported mean times.

Incremental topology-update cost. For the study in Appendix C.5, we benchmark topology-update windows and ordinary training steps using GPU-resident batches. We subtract the cost of an equal number of ordinary steps with matching score-collection settings from each update window and weight the differences over the training schedule. The ordinary-step reference includes the corresponding score collection and EMA updates. We report the resulting increment averaged over all 105,400 optimizer steps in milliseconds per step and accumulated over the 100-epoch schedule in seconds.

D.2 Training FLOPs and Exported Parameter Counts

Both compute measures count only convolution and matrix-multiplication MACs, with two FLOPs per MAC.

Training from scratch. Let M_{fwd} denote forward MACs per image, calculated from the tensor shapes executed during training. We approximate the backward cost as twice the forward cost and report estimated training FLOPs per image as

$$\widehat{F}_{\text{train}} = 6M_{\text{fwd}}. \tag{8}$$

Fine-tuning. We derive forward and backward MAC counts separately from operator shapes observed during training, reusing counts for identical shapes and execution settings. We accumulate these counts over all completed training steps to obtain M_{train} and report cumulative training FLOPs as $F_{\text{total}} = 2M_{\text{train}}$. The matched-compute experiments use this cumulative budget.

Exported parameter counts. We report all parameters physically stored in the exported models, in millions. Parameter counts use validation-selected checkpoints; Static's counts in Table 15 use its final checkpoint. S^2ViTE removes inactive units during export, whereas Static and `BARTER` already store compact tensors. Dense, SET, SNFS, RigL, and SRigL retain full-size tensors in the evaluated exports.

D.3 On-Device Measurement

We run a native Swift implementation of CCT on an iPhone 13 Pro Max with iOS 18.7.8, using MPSGraph on the Metal GPU in a Release build. Training uses FP32 and an effective batch size of 128, formed by accumulating gradients over 32 microbatches of four images each. For each method, we report one step-timing run and one epoch-timing run.

Step time is the mean over 100 optimizer steps after 30 warmup steps. It includes synchronized device execution, gradient accumulation, and the optimizer update; CPU data preparation is outside this interval. Epoch time is measured separately over 351 optimizer steps, covering 44,928 CIFAR-100 images, and includes CPU data preparation. Initial graph compilation and prewarming precede both measurements. `BARTER`'s score collection and EMA updates are included in the measured training time.