DOES ENERGY-AWARE SPARSITY ALLOCATION HELP DYNAMIC SPARSE TRAINING? LEVERAGE, EXECUTION BACK ENDS, AND STABLE REGROWTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Dynamic sparse training (DST) is motivated by lower training cost, and a natural refinement is to allocate sparsity by energy, pruning hardest where computation is most expensive. We test whether this helps, measuring GPU energy with hardware counters in randomized blocks on three execution back ends. An elementary identity bounds what any layer-wise allocation can save at a fixed number of weights: the saving is the drop in the average cost per weight when weights move to cheaper layers. This leverage is zero for masked dense kernels, as used by most DST code, and for the linear layers of Transformers, and it is large in CNNs, whose early layers cost 100–200$\times$ more per weight than their late layers. A score based on each layer's total energy captures it only on AlexNet; allocating by energy per weight captures about half of it. The measurements show why this leverage still does not pay off. Under cuSPARSE kernels, allocation effects are real and predicted within 2.8 points by per-layer costs, but energy is not monotone in density: a uniform network with twice the weights uses 44% less energy than one at the common density of 0.1 and is more accurate. Under channel-structured execution, sparse training finally costs less than dense training ($0.40\times$ on VGG-16), but at equal measured energy neither energy-aware allocation nor its reverse beats simply lowering the density of a uniform allocation. Energy-aware regrowth fares worse: it follows a replicator dynamic that collapses the network into one layer (-9.8 points on AlexNet) unless it is anchored, which we prove and confirm fixes it. On the hardware we measured, the execution back end and the density set the energy of sparse training, not the layer-wise allocation.

1 INTRODUCTION

Training energy has become a first-class concern in machine learning (Strubell et al., 2019; Schwartz et al., 2020; Patterson et al., 2021), and sparsity is one of the oldest tools for reducing it (LeCun et al., 1990; Han et al., 2015; Hoefler et al., 2021). Dynamic sparse training (DST) keeps a network sparse from initialization and rewires it during training (Mocanu et al., 2018; Evci et al., 2020), and its efficiency is usually reported as a fraction of the dense FLOPs. How much sparsity each layer receives is decided by the layer's shape, most often with the Erdős–Rényi-Kernel rule (Evci et al., 2020). For compressing networks for inference, it has long been known that FLOPs and parameter counts are poor proxies for energy and that allocating by an energy model does better (Yang et al., 2017; 2018; He et al., 2018). It is therefore natural to ask whether sparse *training* should allocate its sparsity by energy as well.

We study a concrete version of this idea, EA-DST, which ranks layers with a data-free energy score, prunes high-score layers more at initialization, and biases regrowth toward them (Section 3). A first benchmark of EA-DST over six architectures and ten DST baselines seemed to confirm the idea, with energy savings of up to 22% at high density. On closer inspection those savings came from a regrowth rule that silently changed the density of the network (Section 6.4), and they disappeared once the density was held fixed. That experience led to the question this paper answers: *when can a layer-wise allocation of sparsity lower the measured energy of training, and is it ever better than simply using fewer weights?*

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.

The answer depends on three things that are easy to overlook. The first is *leverage*: at a fixed number of weights, an allocation can only save energy by moving weights into layers that are cheap per weight, so the spread of the cost per weight across layers bounds what any allocation can do (Section 4). The second is *execution*: a saving exists only if the back end actually skips the pruned work, and how much it saves depends on the kernels, not on an analytical score (Sections 6.1–6.3). The third is *dynamics*: a regrowth rule that favors some layers changes the allocation over training, and we show that it does so in a predictable and harmful way (Section 6.4). Our contributions are:

- **A leverage identity** (Proposition 4.1) that tells, from one forward pass and before any training, how much energy any allocation can save on a skip-zero back end. It is zero for masked kernels and for Transformer linear layers, and it explains why the energy score helps only on AlexNet; allocating by energy per weight captures about half of the available leverage on the CNNs we analyze.
- **Measurements on three execution back ends** with hardware counters and randomized blocks. Masked dense kernels make every allocation cost the same, and more than dense training. cuSPARSE kernels make allocation effects real and predictable from per-layer costs, but their cost is not monotone in density, so a denser uniform network is cheaper and more accurate. Channel-structured execution is cheaper than dense training, and there, too, uniform allocation is at least as good as energy-aware allocation at equal measured energy.
- **A theory of score-weighted regrowth** (Proposition 6.1): it drifts in density or collapses the network into its top-score layers, following a replicator dynamic that training logs of two architectures reproduce quantitatively, and an anchored variant is provably stable.
- **A protocol and a corrected benchmark**: measured energy, logged per-layer density, blocked runs with a dense reference, and comparisons at equal energy. The code, logs, and scripts are released.

Our measurements come from one laptop GPU, two CNNs, and CIFAR-10 with short schedules, and we state the limits of each conclusion where it is drawn. The leverage identity and the regrowth analysis do not depend on this setting.

2 Related Work

Sparse training and layer-wise allocation. Dynamic sparse training keeps a network sparse from the start and adapts its connectivity by removing and regrowing weights (Mocanu et al., 2018; Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Jayakumar et al., 2020; Liu et al., 2021; Yuan et al., 2021; Zhang et al., 2025). How much sparsity each layer receives is usually decided by its shape: uniformly, by the Erdős–Rényi(-Kernel) rule (Mocanu et al., 2018; Evci et al., 2020), from saliency at initialization (Lee et al., 2019; Wang et al., 2020; Tanaka et al., 2020), from weight statistics (Lee et al., 2021), or by learned thresholds (Kusupati et al., 2020; Louizos et al., 2018); Liu et al. (2022a) show that with a well-chosen layer-wise allocation even randomly chosen connections train well. Sparse momentum (Dettmers & Zettlemoyer, 2019) already redistributes regrowth across layers, in proportion to momentum magnitude, which is an instance of the reweighted regrowth we analyze in Section 6.4. Structured variants trade flexibility for kernel efficiency: constant fan-in (Lasby et al., 2024), diagonal blocks (Tyagi et al., 2025), channels (Yin et al., 2023), and 2:4 patterns (Mishra et al., 2021). These methods allocate by shape or by training signals rather than by energy, and they typically report FLOPs rather than measured training energy.

Energy of training and energy-aware compression. Energy and carbon accounting (Strubell et al., 2019; Henderson et al., 2020; Anthony et al., 2020; Courty et al., 2024) and GPU power management (You et al., 2023) have made measured energy a standard reporting target. For inference, energy- and latency-aware pruning allocates channels under a modeled or measured budget (Yang et al., 2017; 2018; 2019; He et al., 2018; Shen et al., 2022), usually with accuracy feedback; for training, E2-Train skips data and computation (Wang et al., 2019) and energy-aware training has been proposed with gradient-based importance (Lazzaro et al., 2023; Yeom et al., 2021). It is also well known that sparsity does not translate into speed without kernels that exploit it (Gale et al., 2019; 2020; Hoefler et al., 2021). We connect these threads: we ask when a data-free energy estimate can guide the layer-wise allocation of sparse training, and we answer with a bound that holds before training, hardware energy counters on three execution back ends, and an analysis of the regrowth dynamics.

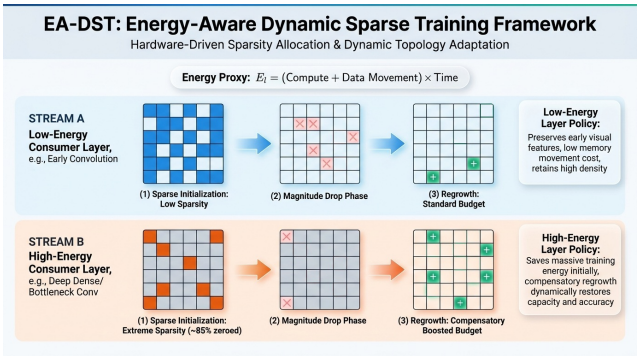

Figure 1: EA-DST. Layers with a higher energy score receive higher sparsity at initialization. During training, the drop phase removes low-magnitude weights and the regrowth budget is biased toward the high-score layers. Section 6.4 shows that this bias compounds over updates, so the high-score layers eventually become the densest layers.

3 Energy-Aware Allocation and Regrowth

EA-DST (Figure 1) changes two layer-wise decisions of DST at a fixed global density d: how dense each layer is at initialization, and how the regrowth budget is split across layers. Both are driven by a data-free score.

Energy score. For a convolution with C^l_{in} input and C^l_{out} output channels, g^l groups, a $K^l \times K^l$ kernel, and an $H^l \times W^l$ output, and for a linear layer with I^l inputs, O^l outputs, and T^l tokens per example, we count multiply–accumulates and data movements (weights and output activations) per example,

$$\Phi^l_{\text{conv}} = \tfrac{C^l_{\text{in}}}{g^l} C^l_{\text{out}} (K^l)^2 H^l W^l, \quad M^l_{\text{conv}} = \tfrac{C^l_{\text{in}}}{g^l} (K^l)^2 C^l_{\text{out}} + H^l W^l C^l_{\text{out}}; \quad \Phi^l_{\text{lin}} = I^l O^l T^l, \quad M^l_{\text{lin}} = I^l O^l + O^l T^l, \tag{1}$$

from one forward pass at the training resolution. With energies e^{op} per operation and e^{mov} per access, $E^l_{\text{step}} = \Phi^l e^{op} + M^l e^{mov}$ estimates the energy of a step, and EA-DST ranks layers by the energy–delay product (Gonzalez & Horowitz, 1996; Horowitz, 2014)

$$S^l = E^l_{\text{step}} \cdot \Phi^l / \Psi, \qquad s^l = S^l \Big/ \textstyle\sum_i S^i, \tag{2}$$

where Ψ is the throughput. Only shares enter the method, so Ψ, the batch size, and any factor common to all layers cancel; with $e^{op} = 0.01\,\mu$J and $e^{mov} = 0.1\,\mu$J (Holtmann, 2024; Delestrac et al., 2024), the ranking barely changes when e^{mov}/e^{op} varies from 1 to 1000 (Spearman $\rho \geq 0.76$; Appendix F). The score ranks measured per-layer energy about as well as FLOPs do (Appendix H).

Allocation. *EA+* sets the layer densities

$$d^l = \min\big(1,\, c\, d\,(1 - S^l_\alpha)\big), \qquad S^l_\alpha = \alpha s^l + (1-\alpha)/L, \qquad \alpha \in [0,1], \tag{3}$$

where c is found by bisection so that $\sum_l N^l d^l = d \sum_l N^l$ and layer sparsity is clamped at 0.99; $\alpha = 0$ is uniform allocation and $\alpha = 1$ is fully score-driven. *FLOP-share* is the same formula with FLOP shares, a control for whether the score knows more than FLOPs. We also report ERK (Evci et al., 2020) and the energy-per-weight allocation introduced in Section 4.

Regrowth. Every 100 iterations, each layer removes the fraction ζ_t of its active weights with the smallest magnitude (Frankle & Carbin, 2019). *Random* regrowth reactivates the same number of weights in the same layer. *EA-random* regrowth instead splits the total budget K_t across layers,

$$N^l_{\text{grow}} = K_t \frac{f^l N^l_{\text{rm}}}{\sum_i f^i N^i_{\text{rm}}}, \qquad f^l = \Big(1 + \beta\big(s^l - \tfrac{1}{L}\big) + \gamma \frac{\|\nabla W^l\|}{\sum_i \|\nabla W^i\|}\Big)_+, \tag{4}$$

so that high-score layers, which are pruned most at initialization, get more chances to recover useful connections. Eq. 4 conserves the global density exactly. The released implementation used for our legacy benchmark scaled each layer's growth probability by $1 + s^l - 1/L$ *without* this renormalization;

we call that rule *EA+/released*. **EA-DST-I** denotes EA+ initialization with random regrowth and **EA-DST** EA+ initialization with EA-random regrowth; defaults are $\alpha = 0.5$ and $\beta = \gamma = 0.5$. Pseudocode and all variants are in Appendix E.

4 How Much Can an Allocation Save?

Before training anything, it is worth asking how much a layer-wise allocation can change energy at all. Let layer l have N^l weights and density d^l, and fix the global budget $\sum_l N^l d^l = d \sum_l N^l$. Write the energy of a training step as $E = E_0 + \sum_l a_l(d^l)$, where E_0 collects the work that the allocation does not touch (the input pipeline, the optimizer, normalization of activations) and a_l is the cost of layer l on a given execution back end. Two cases bracket the back ends we measure. With masked dense kernels, $a_l(d^l) = A_l$ does not depend on the density. With a back end that skips pruned work, the arithmetic and weight traffic of a layer scale with its kept weights, $a_l(d^l) \approx d^l A_l$, where A_l is the cost of the dense layer.

Proposition 4.1 (Allocation leverage)**.** *Let $a_l(d^l) = d^l A_l$ and let $c_l = A_l / N^l$ be the cost per weight of layer l. For every allocation that meets the budget,*

$$\frac{\sum_l a_l(d^l)}{\sum_l a_l(d)} = \frac{\sum_l w_l\, c_l}{\sum_l \pi_l\, c_l}, \qquad w_l = \frac{N^l d^l}{d \sum_i N^i}, \quad \pi_l = \frac{N^l}{\sum_i N^i}. \tag{5}$$

(a) If c_l is the same for all layers, every allocation has the energy of uniform allocation. (b) Over allocations with $d^l \in [d_{\min}, 1]$, the ratio is smallest when the budget is filled in increasing order of c_l (a fractional knapsack). (c) With masked dense kernels ($a_l = A_l$), every allocation has the energy of uniform allocation.

The identity follows from $\sum_l d^l A_l = \sum_l N^l d^l c_l = dN \sum_l w_l c_l$ and $\sum_l d A_l = dN \sum_l \pi_l c_l$ with $N = \sum_l N^l$. It is elementary, but it separates two questions that are usually mixed together: whether a model offers any leverage at all, which depends only on how much the cost per weight varies across its layers, and whether a given allocation uses it, which depends on whether the allocation moves weights toward layers that are cheap per weight. We call $r = \sum_l w_l c_l / \sum_l \pi_l c_l$ the *relative cost* of an allocation; $r < 1$ is a saving.

Transformers have no leverage. In a Transformer with T tokens per example, every linear layer performs T multiply–accumulates per weight and reads each weight once per step. Up to the traffic of the output activations, which differs by a few percent between layers, the cost per weight is therefore the same for the query, key, value, output, and MLP projections of every block, and by part (a) no reallocation of unstructured sparsity among them can change the compute of a skip-zero back end; only the attention scores (which have no weights), the embeddings, and the output head break the symmetry. Our ViT shows it: its cost per weight varies by a factor of 1.4 across 38 layers, and no allocation can save more than 1% (Table 1). At the scale of language models the spread all but vanishes. For GPT-2 small at 1024 tokens the cost per weight varies by 1% across its 49 linear layers, and for a decoder with the shapes of LLaMA-7B at 2048 tokens by 0.15%, so no reallocation of a fixed number of weights can save more than 0.7% and 0.1% of their compute. Layer-wise allocation rules for language models, such as OWL (Yin et al., 2024), can therefore change accuracy but not the arithmetic of a fixed-budget sparse model.

Convolutional networks have a lot of it. A convolution uses each of its weights once per output position, so its cost per weight is roughly proportional to its output resolution, and the high-resolution early layers of VGG-16, ResNet-34, or MobileNetV2 cost 100–200$\times$ more per weight than their late layers. Table 1 computes the leverage of all six models from one forward pass, using the per-step energy estimate of the score, $A_l = E^l_{\text{step}}$ (Eq. 1). The best possible allocation would halve the compute of VGG-16 and ResNet-34 at a fixed number of weights ($r_{\min} = 0.50$ and 0.46).

What EA+ captures, and a direct alternative. EA+ ranks layers by their share of the energy–delay score, which grows with a layer's *total* cost, not with its cost per weight. The two orderings agree only when the costliest layers are also the most expensive per weight. They do on AlexNet, where EA+ reaches $r = 0.83$ at $\alpha = 1$; on VGG-16, ResNet-34, and MobileNetV2 they do not, and EA+ leaves r between 0.98 and 1.02 although these models offer the most leverage. The knapsack solution of part (b) is the opposite extreme: it empties the expensive layers, and Section 6.2 shows what that

Table 1: Allocation leverage at $d = 0.1$, from one forward pass at the CIFAR training resolution (no training). c_l: estimated energy per weight of layer l (E^l_{step}/N^l); max/min and CV (parameter-weighted coefficient of variation) describe its spread. The other columns give r (Proposition 4.1), the energy of the sparse computation relative to uniform allocation under a back end whose cost is proportional to the kept weights: EA+ with $\alpha = 0.5$ and 1, the energy-per-weight allocation with $\nu = 0.25$ and 0.5, and the best any allocation can do ($r_{\min}$, layer densities in $[0.01, 1]$). $r = 1$: no saving. *Language models, computed analytically from the layer shapes (every linear layer at T tokens).

		Cost per weight c_l		**EA+**		**Energy per weight**		
Model	L	max/min	CV	α=0.5	α=1	ν=0.25	ν=0.5	$r_{\min}$
AlexNet	8	17	0.95	0.91	0.83	0.89	0.80	0.56
VGG-16	14	128	2.02	0.99	0.98	0.83	0.73	0.50
ResNet-34	37	128	1.78	1.00	1.01	0.80	0.68	0.46
ConvNeXt	59	8	0.51	1.00	1.01	0.97	0.96	0.84
MobileNetV2	53	197	2.90	1.01	1.02	0.84	0.77	0.63
ViT	38	1	0.01	1.00	1.00	1.00	1.00	0.99
GPT-2 small, T=1024*	49	1.01	0.004	–	–	–	–	0.993
LLaMA-7B shape, T=2048*	225	1.00	0.001	–	–	–	–	0.999

costs in accuracy. We therefore also test a tempered version that targets cost per weight directly, the **energy-per-weight (EPW)** allocation

$$d^l = \min\big(1,\ c\,(c_l)^{-\nu}\big), \tag{6}$$

with c set by bisection to meet the budget and ν a single strength parameter. It is uniform at $\nu = 0$ and approaches the knapsack solution as $\nu \to \infty$; at $\nu = 0.5$ it reaches $r = 0.68$–0.80 on the four models with substantial leverage, about half of what is available. A negative ν does the opposite and favors the layers that are expensive per weight; we use it as a control. Proposition 4.1 says how much energy an allocation *can* save on an ideal skip-zero back end. It says nothing about accuracy, and real back ends are not ideal; the rest of the paper measures both.

5 Experimental Setup

Three execution back ends. The same sparse model can be executed in very different ways, and we measure three that differ only in how pruned weights are handled. *Masked dense* execution, used by most DST code, keeps every weight tensor dense and multiplies it by its mask after each optimizer step, either as a dense tensor or, as in the released implementation, through a PyTorch sparse tensor; all products are dense. The *skip-zero CSR* back end runs every convolution and fully connected layer through cuSPARSE: the forward product and the input gradient use CSR sparse–dense products, and the weight gradient is computed only at the non-zeros with a sampled dense–dense product; its outputs match masked dense execution to $2 \cdot 10^{-9}$ (Appendix C). The *channel-structured* back end prunes whole output channels (neurons in FC layers): in every step it gathers the active slice of each weight tensor and runs the ordinary cuDNN or cuBLAS kernel on it, so arithmetic, activations, and memory traffic shrink with the kept channels. Its topology changes by channel-level DST in the spirit of Chase (Yin et al., 2023): every 100 iterations each layer drops the weakest active channels (cosine-decayed fraction starting at 0.1) and regrows as many at random, with fresh incoming and zero outgoing weights, so the allocation stays at its target (Appendix D). Its forward pass matches the masked network to $2 \cdot 10^{-7}$. For the channel back end, a layer's weight density is the product of its own and its input's channel keep ratios, and an allocation sets the keep ratios with the same formulas as above, with the budget on the total number of kept weights.

Measurement protocol. All controlled runs use one NVIDIA RTX A500 laptop GPU. Energy is the GPU energy of the training epochs from the NVML hardware counter; evaluation is excluded, and the enforced power limit and temperature are logged every epoch. Each seed is a *block* that contains a dense reference run and every configuration of a comparison in random order, and every comparison is paired within blocks. The platform sets the power limit (35–45 W) and does not let it be fixed, and pairing removes the resulting block-to-block variation (Appendix B). No run is excluded without disclosure: a pilot and two runs slowed by other work on the machine were repeated, and all of them are in the released logs. Runs that only provide accuracy (the accuracy of uniform allocation at other

densities under masked execution and the anchor sweep) were partly run two at a time, and their energy is not used. Paired t-tests are Holm-corrected within each comparison.

Models and training. AlexNet (the CIFAR variant with batch normalization, 8 layers) and VGG-16 with batch normalization are trained on CIFAR-10 with SGD (Nesterov momentum 0.9, learning rate 0.1 divided by 10 at 50% and 75% of training, weight decay $5 \cdot 10^{-4}$, batch size 128) for 30 and 20 epochs; the channel-structured comparisons use the same schedule shape over 8 and 5 epochs, so that every block can hold all nine configurations, and their accuracies are correspondingly lower. We report test accuracy of the checkpoint selected on a held-out 10% validation split. Unstructured runs use magnitude removal and random regrowth every 100 iterations with a cosine-decayed drop fraction starting at 0.5. Nothing was tuned per method. A larger legacy benchmark (six architectures, ten DST baselines, 42 scenarios, whole-device CodeCarbon energy) is re-analyzed in Appendix A and summarized in Appendix A.1.

6 Results

We take the back ends in order of how much they let an allocation matter, then turn to accuracy at a fixed density and to regrowth.

6.1 Masked Dense Kernels: No Allocation Changes Energy

With masked dense execution the products do not depend on where the zeros are, so Proposition 4.1(c) predicts no effect, and the measurements agree. On AlexNet, uniform, ERK, FLOP-share, and EA+ allocations differ from each other by -8 to $+3$% in paired energy, within the block-to-block spread, and every sparse configuration costs *more* than dense training: 4.6–6.1% more when the mask is a dense tensor and 19–91% more when it is applied through PyTorch sparse tensors, as in the released DST code (Appendix B, Table 11). Sparse-tensor masking lowers the GPU's power draw but lengthens training by 1.3–2.4×, and energy follows the runtime (Figure 6c). This is the back end on which most DST results, including our own legacy benchmark, were obtained.

6.2 Skip-Zero CSR Kernels: Real Effects, Set by the Kernels

The allocation changes energy, as the leverage predicts. With every product executed on the non-zeros, EA+ lowers the energy per epoch of AlexNet at $d = 0.1$ monotonically in α, by 5.0, 11.3, 18.5, and 26.7% for $\alpha = 0.25$, 0.5, 0.75, and 1 in three blocks that contain every α (Holm-corrected $p \leq 0.006$; Table 2). On VGG-16, where EA+ has no leverage (Table 1), the savings stay below 2%. The effects are fully explained by per-layer costs: summing the energies of the layers measured one at a time predicts every AlexNet allocation effect within 2.8 points, without any fitted parameter (Appendix C).

But the kernels, not the energy score, decide what is cheap. The cost of a CSR layer is not monotone in its density. Timing every operation of the back end separately shows why (Appendix C): the sampled dense–dense product (SDDMM) that computes the weight gradient accounts for 97% of the energy of AlexNet's second convolution at density 0.1, and it becomes 13× cheaper between densities 0.2 and 0.3, where cuSPARSE evidently changes algorithm; the two sparse–dense products grow roughly in proportion to the density. The slow regime also depends on the memory layout of the dense operand, which our back end passes as a contiguous transposed copy: with a transposed view instead, the same product is 8–20× cheaper at densities 0.1 and 0.05. The popular density of 0.1 sits squarely in this regime. As a result, uniform networks with two and three times the weights ($d = 0.2$ and 0.3) use 44% and 54% less energy than uniform allocation at $d = 0.1$, consistently in all three blocks, and they are more accurate (82.81% and 83.22% against 81.18%); uniform allocation at $d = 0.05$ saves 30%, slightly more than EA+ at $\alpha = 1$ (26% in the same blocks). Measured against the best uniform network that uses no more energy, EA+ at $\alpha = 1$ is 2.66 points *less* accurate (95% CI $[-2.86, -2.43]$). The energy-optimal allocation computed from the per-layer costs saves 70.4% but loses 10.75 points, and even it costs 4.0× dense training. On this GPU, general-purpose sparse kernels make every configuration we ran 4–15× more expensive than dense training, and the allocation that looks energy-aware at $d = 0.1$ mostly moves the largest layer out of a kernel's slow regime. A better operand layout would lower all CSR costs, by an estimated 85% for that layer, but it

Table 2: Skip-zero CSR back end on CIFAR-10. Energy: GPU energy per training epoch relative to the dense run of the same block; ΔE: paired change against uniform allocation at $d = 0.1$ in the same blocks (mean $\pm$ std over 3 blocks; the third block of the uniform rows used one epoch per run, the others two); accuracy: test accuracy after 30 epochs with the same allocation and masked execution (5 seeds; 2 for uniform at $d \neq 0.1$; accuracy does not depend on the back end); G: accuracy minus that of the most accurate uniform network that uses no more energy (upper envelope of the uniform rows, interpolated in log-energy; 95% bootstrap interval over seeds; †: cheaper than every uniform row, compared with the cheapest). All ΔE of the AlexNet allocations have Holm-corrected $p \leq 0.006$. VGG-16 has no uniform curve.

Allocation	d	**Energy**	ΔE (%)	**Accuracy (%)**	G (pts)
AlexNet (dense: 0.060 Wh/epoch, 83.60%)					
Uniform	0.05	9.7×	-30.2 ± 0.3	78.93 ± 0.19	–
Uniform	0.1	14.0×	–	81.18 ± 0.54	–
Uniform	0.2	7.9×	-43.6 ± 1.2	82.81 ± 0.11	–
Uniform	0.3	6.5×	-53.5 ± 1.0	83.22 ± 0.01	–
EA+ (α=0.25)	0.1	13.3×	-5.0 ± 0.5	81.16 ± 0.15	-2.06 [-2.2, -1.9]
EA+ (α=0.5)	0.1	12.4×	-11.3 ± 0.7	81.37 ± 0.30	-1.84 [-2.0, -1.6]
EA+ (α=0.75)	0.1	11.4×	-18.5 ± 1.8	80.85 ± 0.27	-2.36 [-2.6, -2.2]
EA+ (α=1)	0.1	10.3×	-26.7 ± 0.1	80.56 ± 0.27	-2.66 [-2.9, -2.4]
FLOP-share (α=0.5)	0.1	13.5×	-6.8 ± 1.3	80.97 ± 0.46	-2.24 [-2.6, -1.9]
ERK	0.1	10.7×	-25.7 ± 1.6	78.91 ± 0.22	-4.30 [-4.5, -4.1]
Energy optimum (Table 16)	0.1	4.0×	-70.4 ± 1.8	70.44 ± 0.77	-12.78 [-13.4, -12.1]†
VGG-16 (dense: 0.296 Wh/epoch, 90.72%)					
Uniform	0.1	14.4×	–	88.74 ± 0.28	–
EA+ (α=0.5)	0.1	14.3×	-0.7 ± 0.3	88.81 ± 0.06	–
EA+ (α=1)	0.1	14.1×	-1.8 ± 0.6	88.75 ± 0.18	–
FLOP-share (α=0.5)	0.1	14.3×	-0.2 ± 0.9	88.81 ± 0.10	–
ERK	0.1	15.3×	$+6.4 \pm 0.9$	90.12 ± 0.13	–

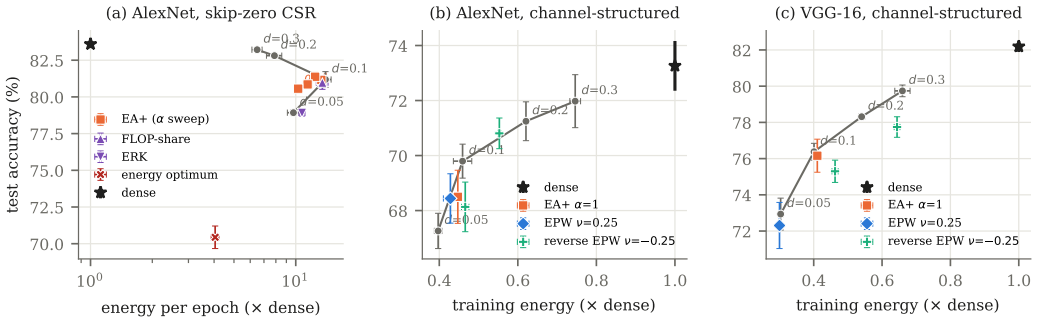

Figure 2: Measured energy against test accuracy on CIFAR-10. Gray: uniform allocation at global densities 0.05–0.3, which defines the reference curve; colored: allocations at $d = 0.1$ (reverse EPW also at 0.05); black star: dense training. Error bars: std over blocks and seeds. (a) Skip-zero CSR back end, AlexNet, energy per epoch relative to dense training (log scale); the uniform curve is not monotone because the kernels are slowest near $d = 0.1$. (b, c) Channel-structured back end, whole-training energy relative to dense training. Points above the gray curve would beat uniform allocation at equal energy.

would not change the lesson: what an allocation saves on this back end is decided by kernel behavior that no analytical score models.

6.3 Channel-Structured Execution: Real Savings, but Not From the Allocation

Channel-structured execution is the one back end on which sparse training is cheaper than dense training on our GPU. With uniform allocation at $d = 0.1$ it trains VGG-16 at 0.40× and AlexNet at 0.46× the energy of dense training, for 5.8 and 3.5 points of accuracy (Table 21 in Appendix D). The uniform curves in Figure 2b–c show how energy and accuracy trade off as the density changes, and they are the natural reference for any allocation.

No allocation we tried lies above these curves. EPW does what Proposition 4.1 says it should at a fixed number of weights: on VGG-16 it moves weights out of the high-resolution layers and cuts the energy from $0.40\times$ to $0.30\times$ dense. But it gives up accuracy about as fast as lowering the density does, and at equal energy it is no better than uniform allocation ($G = -0.63$ points, 95% CI $[-1.95, +0.74]$). On AlexNet, whose thin layers leave the GPU mostly idle, the same reallocation saves almost nothing ($G = -0.14$). EA+ at $\alpha = 1$ also sits on the curve on both models ($G = -0.37$ and -0.83). To check that the uniform curve is not simply hard to beat from one side, we also ran the reverse allocation ($\nu = -0.25$), which keeps the costly early layers wide and thins the cheap late layers. It is clearly worse on VGG-16, by 2.01 and 1.82 points at equal energy, and within noise or worse on AlexNet. For both CNNs, the uniform allocation at the right density is as good as any allocation we tested, energy-aware or not: the leverage is real, but spending it costs accuracy at least as fast as using fewer weights everywhere.

6.4 Energy-Aware Regrowth Is Unstable Unless Anchored

EA-random regrowth (Eq. 4) was meant to give heavily pruned, high-score layers a second chance. The following proposition shows what a multiplicative per-layer bias does over many topology updates.

Proposition 6.1 (Score-weighted regrowth)**.** *Let n_t^l be the active weights of layer l after update t and $x_t^l = n_t^l / \sum_i n_t^i$ its share. Let every update remove the fraction $\zeta_t \in (0,1)$ of each layer's active weights and regrow with fixed positive factors f^l, ignoring the cap at full density, and write $\bar{f}_t = \sum_l f^l x_t^l$. (a) Without renormalization, $\sum_l n_{t+1}^l = (1 - \zeta_t + \zeta_t \bar{f}_t) \sum_l n_t^l$: the global density drifts unless $\bar{f}_t = 1$ at every update. (b) With renormalization (Eq. 4), $x_{t+1}^l = x_t^l\big(1 - \zeta_t + \zeta_t f^l / \bar{f}_t\big)$. For $f^l > f^m$ the ratio x_t^l / x_t^m grows by at least $\exp(c \sum_t \zeta_t)$ with $c > 0$ depending on $f^l - f^m$ and $\max_i f^i$, so the weights concentrate in the layers with the largest f^l. (c) The shares are constant if and only if all active layers have the same factor, i.e., each layer regrows what it lost. (d) Adding an anchor toward a target allocation $n_\star^l$ with $\sum_l n_\star^l = \sum_l n_0^l$,*

$$N_{grow}^l = K_t \frac{f^l n_t^l}{\sum_i f^i n_t^i} + \eta\big(n_\star^l - n_t^l\big), \qquad \eta \in (0, 1], \tag{7}$$

conserves the global density, and if $\zeta_{\max}\delta < \eta$ with $\delta = \max_i f^i / \min_i f^i - 1$, every layer ends within $\limsup_t |n_t^l / n_\star^l - 1| \le \zeta_{\max}\delta / (\eta - \zeta_{\max}\delta)$ of its target.

Part (b) is the discrete replicator equation (Taylor & Jonker, 1978; Hofbauer & Sigmund, 1998) applied to layer shares; the proof is in Appendix B.2. It explains three things we observed. First, the released rule, which scales growth probabilities by $1 + s^l - 1/L$ without renormalizing, changes the global density: replaying its mask updates with the benchmark schedule takes a nominal density of 0.9 to 0.23 on AlexNet and moves every model toward a density of its own (Appendix B), which is where the regrowth "savings" of our earlier benchmark came from. Second, renormalizing does not help. On AlexNet at $d = 0.1$ the second convolution grows to a density of 0.80 while every other layer falls below 1.2%, and accuracy drops by 9.8 points (5 seeds); the logs follow part (b) quantitatively, with $\ln(d^{\text{conv2}}/d^l)$ linear in the cumulative drop fraction ($r \ge 0.995$ in every seed and layer; Figure 3). Third, the rate depends on how concentrated the score is, since $f^l - f^m = \beta(s^l - s^m)$. Iterating the recursion without training, from the layer sizes and score shares alone, predicts strong concentration on AlexNet, MobileNetV2, and VGG-16 and mild drift on ResNet-34, ConvNeXt, and ViT (Appendix B.3). On VGG-16 the prediction holds: the two highest-score convolutions reach a density of 0.181 (predicted 0.179), yet accuracy does not change (+0.27 points, 3 seeds), because no layer is starved within 20 epochs.

Part (d) gives a stable variant. With $\eta = 0.5$, the anchored rule keeps every layer within 0.6% of its initial allocation on both models and restores the accuracy of energy-aware initialization (AlexNet: +0.20 points against uniform, $p_{\text{Holm}} = 0.78$). Weaker anchors let the allocation drift, but accuracy degrades gradually, with $\eta = 0.2$: +0.59 points, largest deviation 2%; $\eta = 0.05$: −0.39 points, largest deviation 130%; $\eta = 0.01$: −1.56 points, largest deviation 688% (Appendix B.3). But an anchored rule by construction leaves the allocation at its target, so any energy effect of energy-aware regrowth reduces to that of the target allocation, which Sections 6.2 and 6.3 examine.

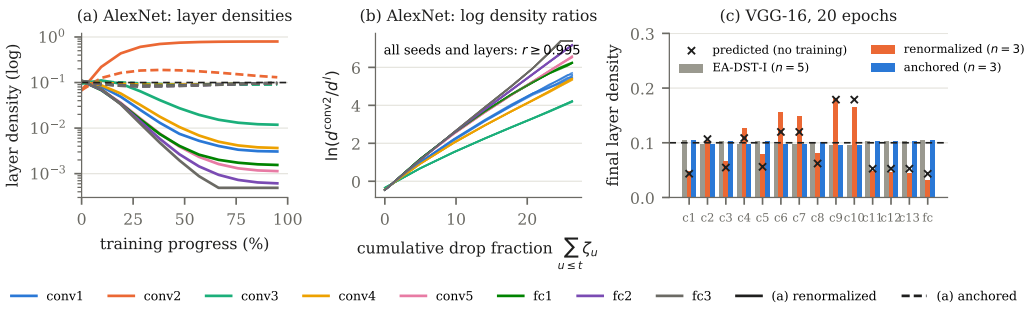

Figure 3: Energy-aware regrowth at $d = 0.1$. (a) AlexNet, per-layer density under the renormalized rule (solid) and the anchored rule of Eq. 7 (dashed), mean over seeds. (b) AlexNet, renormalized rule: $\ln(d^{\text{conv2}}/d^{l})$ is linear in the cumulative drop fraction, as Proposition 6.1(b) predicts; one line per seed and layer. (c) VGG-16, final per-layer density of EA-DST-I, the renormalized rule, and the anchored rule, with the prediction of the recursion computed without training (crosses).

7 Conclusion

Allocating sparsity by energy is an appealing idea, and our first benchmark seemed to confirm it. A careful look says otherwise. The leverage identity shows that an allocation can save energy only by moving weights into layers that are cheap per weight, which rules out masked kernels and Transformer linear layers from the start and explains why a score based on total layer energy helps only on AlexNet. On the back ends where the leverage is real, the kernels decide what is cheap, and moving weights toward cheap layers costs accuracy at least as fast as simply using fewer of them. The reverse allocation, which keeps the expensive layers wide, does no better, and clearly worse on VGG-16, so the uniform curve is not an artifact of looking in one direction. Energy-aware regrowth adds a failure of its own, a replicator dynamic that collapses the network, and anchoring it to a target allocation fixes the collapse at the price of reducing it to that allocation.

Recommendations. (1) Compute the leverage (Table 1) before running an energy-aware allocation; if the cost per weight barely varies across layers, no allocation can help. (2) Report measured energy together with the execution back end; FLOPs do not predict either the sign or the size of the effect. (3) Compare an allocation with uniform allocation at *equal measured energy*, i.e., against uniform allocation at other densities, not only at the same density. (4) Report the realized global and per-layer density at the end of training; a regrowth rule that reweights layers needs an anchor. (5) Run comparisons in randomized blocks with a dense reference and use paired tests.

Limitations. All controlled measurements come from one laptop GPU whose power limit we could not fix, and they cover GPU energy only. We train two CNNs on CIFAR-10 with short schedules (20–30 epochs, and only 5–8 for the channel-structured comparisons), so accuracies are below well-tuned recipes, and the equal-energy comparisons rest on three blocks and two to five seeds per point. Our channel-structured back end is a straightforward implementation: it gathers the active slices in every step and keeps the optimizer on the full tensors, and a production implementation would be faster. The CSR back end uses general-purpose cuSPARSE kernels; faster sparse kernels or accelerators would change the costs, and the leverage identity together with the additive model of Appendix C is how we suggest predicting the effect there. Our statements about Transformers rest on the identity and on offline analysis, not on training runs. The legacy benchmark was run with an earlier implementation and whole-device energy estimates; we use it only for accuracy comparisons and describe its caveats in Appendix A.

AI Use Statement

In this work, we used generative AI tools to assist with grammar correction, improving writing clarity, and debugging code. We have not used generative AI tools to generate research hypotheses, design experiments or produce results of this work. Generation of experimental results, evaluation, interpretation of findings, and all scientific claims were conducted by the authors. Other disclosure

categories are not applicable to this work. Additionally, we used generative AI tools to improve code readability and identify potential implementation issues during code development. All AI-assisted text and code suggestions were reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility Statement

The supplementary material contains the reference implementation with all allocation and regrowth rules (`ea_v6.py`), the skip-zero CSR back end (`csr_backend.py`), the channel-structured back end (`ea_chan.py`), the leverage analysis (`leverage.py`), the additive CSR model (`csr_additive.py`), the kernel benchmarks (`layer_bench.py`, `csr_kernels.py`, `csr_kernels2.py`), the regrowth simulation and replay (`sim_replicator.py`, `sim_drift.py`), the Pareto analysis (`pareto_v72.py`), the benchmark parser and statistics (`parse.py`, `gen.py`), the run queues, all raw run logs (per-epoch energy, power limit, temperature, and per-layer density), and the scripts that generate every table, figure, and in-text number of this paper as LaTeX macros. Models, datasets, splits, optimizer, and schedules are in Appendix I, and the energy measurement in Appendix J. **Energy of this study.** The legacy benchmark cells sum to about 270 kWh of whole-device energy (812 configuration means $\times$ three runs, excluding ablations, CPU and A100 runs, and failed or repeated runs), and the 324 logged controlled runs used 0.81 kWh of GPU energy.

References

Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv preprint arXiv:2007.03051*, 2020.

Benoit Courty, Victor Schmidt, Sasha Luccioni, Goyal-Kamal, MarionCoutarel, Boris Feld, Jérémy Lecourt, LiamConnell, Amine Saboni, Inimaz, supatomic, Mathilde Léval, Luis Blanche, Alexis Cruveiller, ouminasara, Franklin Zhao, Aditya Joshi, Alexis Bogroff, Hugues de Lavoreille, Niko Laskaris, Edoardo Abati, Douglas Blank, Ziyao Wang, Armin Catovic, Marc Alencon, Michał Stęchły, Christian Bauer, Lucas Otávio N. de Araújo, JPW, and MinervaBooks. mlco2/codecarbon: v2.4.1, May 2024. URL `https://doi.org/10.5281/zenodo.11171501`.

Paul Delestrac, Jonathan Miquel, Debjyoti Bhattacharjee, Diksha Moolchandani, Francky Catthoor, Lionel Torres, and David Novo. Analyzing GPU Energy Consumption in Data Movement and Storage . In *2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 143–151, Los Alamitos, CA, USA, July 2024. IEEE Computer Society. doi: 10.1109/ASAP61560.2024.00038. URL `https://doi.ieeecomputersociety.org/10.1109/ASAP61560.2024.00038`.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019. URL `https://arxiv.org/abs/1907.04840`.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2943–2952. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/evci20a.html`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *ArXiv*, abs/1902.09574, 2019. URL https://api.semanticscholar.org/CorpusID:67855585.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse GPU kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, 1996.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, second edition, 2008.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *European Conference on Computer Vision*, pp. 784–800, 2018.

Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248):1–43, 2020.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Josef Hofbauer and Karl Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, 1998.

Jonah Holtmann. Power consumption properties of modern gpus. Bachelor's thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2024. URL https://hpc.fau.de/files/2025/01/BA_Jonah_Holtmann.pdf.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014. doi: 10.1109/ISSCC.2014.6757323.

Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-K always sparse training. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5544–5555, 2020.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=kOBkxFRKTA.

Dario Lazzaro, Antonio Emanuele Cinà, Maura Pintor, Ambra Demontis, Battista Biggio, Fabio Roli, and Marcello Pelillo. Minimizing energy consumption of deep learning models by energy-aware training. In Gian Luca Foresti, Andrea Fusiello, and Edwin Hancock (eds.), *Image Analysis and Processing – ICIAP 2023*, pp. 515–526, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-43153-1.

Ya Le and Xuan Yang. Tiny ImageNet visual recognition challenge. *CS 231N Course Report, Stanford University*, 2015.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, pp. 598–605, 1990.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=H6ATjJ0TKdf.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-Shot Network Pruning Based on Connection Sensitivity. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. In *Advances in Neural Information Processing Systems*, 2021.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=VBZJ_3tz-t.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11976–11986, June 2022b.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*, 2018.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

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, June 2018.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4646–4655, 2019.

David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M. Alvarez. Structural pruning via latency-saliency knapsack. In *Advances in Neural Information Processing Systems*, 2022.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*, 2020.

Peter D. Taylor and Leo B. Jonker. Evolutionarily stable strategies and game dynamics. *Mathematical Biosciences*, 40(1–2):145–156, 1978.

Abhishek Tyagi, Arjun Iyer, William H Renninger, Christopher Kanan, and Yuhao Zhu. Dynamic sparse training of diagonally sparse networks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=bAUVnNc0Ky.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkgsACVKPH.

Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-Train: Training state-of-the-art CNNs with over 80% energy savings. In *Advances in Neural Information Processing Systems*, 2019.

Haichuan Yang, Yuhao Zhu, and Ji Liu. ECC: Platform-independent energy-constrained deep neural network compression via a bilinear regression model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11206–11215, 2019.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning . In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6071–6079, Los Alamitos, CA, USA, July 2017. IEEE Computer Society. doi: 10.1109/CVPR.2017.643. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.643.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Seul-Ki Yeom, Kyung-Hwan Shim, and Jee-Hyun Hwang. Toward compact deep neural networks via energy-aware pruning. *arXiv preprint arXiv:2103.10858*, 2021. URL https://arxiv.org/abs/2103.10858.

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

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In *International Conference on Machine Learning*, 2024.

Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 119–139, 2023.

Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, Siyue Wang, Minghai Qin, Bin Ren, Yanzhi Wang, Sijia Liu, and Xue Lin. Mest: Accurate and fast memory-economic sparse training framework on the edge. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20838–20850. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/ae3f4c649fb55c2ee3ef4d1abdb79ce5-Paper.pdf.

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=OM0Qkq9xtY.

APPENDIX CONTENTS

A Benchmark: Protocol, Paired Results, Statistics, and Data Audit **15**
A.1 Summary . . . 15
B Replay of the Released Mask Updates and Controlled-Study Details **20**
B.1 Accuracy at a Fixed Density . . . 22
B.2 Proof of Proposition 6.1 . . . 23
B.3 Share Dynamics on the Training Logs . . . 23
C Skip-Zero CSR Back End **25**
D Channel-Structured Back End **27**
E Method Details **28**
F Energy Proxy: Metrics, Robustness, and Design Choices **30**
F.1 Robustness of the Score and of the EA+ Allocation . . . 30
F.2 Proxy-Accuracy Metrics . . . 30
F.3 Invariances of the Score Shares . . . 31
F.4 Backward-Pass Cost . . . 31
F.5 Hardware Coefficients . . . 31
F.6 Allocation Safeguards . . . 31
F.7 Energy and Layer Importance . . . 32
G A Stylized Allocation Model (Motivation Only) **32**
H Rank Agreement of the Score with Measurements **32**
I Configuration Setup **32**
J Energy Measurement **34**
K Sensitivity and Ablation: Full Results **34**
L Layers' Density **34**
M Accuracy and Energy Consumption **35**
N Extreme Sparsity **37**
O Various Processor Architectures **37**

A BENCHMARK: PROTOCOL, PAIRED RESULTS, STATISTICS, AND DATA AUDIT

A.1 SUMMARY

Our earlier benchmark trained six architectures (AlexNet, VGG-16, ResNet-34, ConvNeXt, MobileNetV2, ViT-B/16) on CIFAR-10/100 and Tiny-ImageNet with ten DST baselines and measured whole-device energy with CodeCarbon, all with masked execution (Appendix A). Three conclusions survive the checks above. At $d = 0.1$ the unstructured random-growth methods save 10–13% energy relative to dense training in the median, at a cost of 2–4 points, and at $d \geq 0.5$ nearly all of them use more energy than dense training. Energy-aware initialization is statistically indistinguishable from every baseline in accuracy, also when scenarios are first aggregated per model. And it does not transfer to RigL, MEST, or CHT. The energy advantages that the benchmark attributed to energy-aware regrowth came from the density drift of Section 6.4. While re-parsing the result tables we also found four RigL rows that a typesetting error had hidden and three cells that cannot be re-derived; all statistics include the correction.

Benchmark setup. We train AlexNet (Krizhevsky et al., 2012), VGG-16 with batch normalization (Simonyan & Zisserman, 2015), ResNet-34 (He et al., 2016), ConvNeXt (Liu et al., 2022b), MobileNetV2 (Sandler et al., 2018), and ViT-B/16 (Dosovitskiy et al., 2021) from scratch on CIFAR-10/100 (Krizhevsky, 2009) at densities 0.1, 0.5, and 0.9, on Tiny-ImageNet (Le & Yang, 2015) (AlexNet, VGG-16, ViT) at 0.1, and at 0.01 for four CIFAR scenarios. All methods share model, epochs (20–100; Table 26), optimizer (SGD with Nesterov momentum 0.9, weight decay $5 \cdot 10^{-4}$, step decay), batch size 128, and topology-update period (100 iterations). The reported accuracy is **test accuracy** of the checkpoint selected on a held-out 10% validation split. Energy is the CPU+GPU+RAM energy of the training epochs measured with CodeCarbon (Courty et al., 2024) on an Intel i7-13700H

Method	Nominal FLOPs	Measured energy
Dense	$1.0\times$	$1.00\times$ (60.03 Wh)
SNIP	$\approx 0.1\times$	$0.92\times$
ERK (SET)	$\approx 0.1\times$	$0.92\times$
RigL	$\approx 0.1\times$	$0.97\times$
EA-DST-I	$\approx 0.1\times$	$0.92\times$

Table 3: Benchmark (original implementation): nominal training FLOPs vs. measured training energy, ResNet-34 on CIFAR-100, density 0.1 (Supplementary Table S2). These whole-device CodeCarbon estimates were attributed to a CSR execution path that is not in the released code; the controlled CSR back end of Section 6.2 costs 10–$15\times$ dense energy on the same GPU model, and the controlled masked-dense harness measures a GPU-energy *penalty* at this density, so we do not rely on these numbers.

with an NVIDIA RTX A500 (Appendix J). ViT-B/16 receives 32×32 (CIFAR) or 64×64 (Tiny-ImageNet) images, i.e., 4 or 16 patch tokens plus a class token; it therefore behaves more like a deep MLP than a standard ViT, and we qualify every ViT result accordingly. Budgets are short and ViT and ConvNeXt are trained with SGD, so absolute accuracies are below well-tuned recipes, and dense ResNet-34 and ViT are under-trained (sparse runs sometimes exceed them); conclusions are specific to this regime.

Naming. In the benchmark, every energy-aware-regrowth run used the *released* rule (growth probability scaled by $1 + s^l - 1/L$, no renormalization; Section 3), not the renormalized EA-random rule of the reference implementation. We therefore label that configuration **EA+/released** throughout this appendix and in the supplementary tables ("EA+/EArandom"); **EA-DST-I** (EA+ initialization, random regrowth) is identical in both implementations up to the score details of Section efsec:proxy.

Evaluation protocol. A *scenario* is a (model, dataset, density) triple (42 in total); we analyse the 39 scenarios at $d \geq 0.1$, because the three $d = 0.01$ scenarios contain values that cannot be re-derived without per-run logs (Appendix A). Runs at chance accuracy are excluded. Per baseline, we report the median paired difference with a 95% bootstrap CI and a Holm-corrected two-sided Wilcoxon test; wins and ties use tolerances of 1% and 0.5 points.

Source of every number. All benchmark statistics in Section 5 are computed from the per-configuration result tables of Appendix M (Tables 31–32 and Supplementary Tables S1–S5). A script parses these tables into one machine-readable file, computes every statistic, and emits the tables of this section; it is included in the supplementary material. Each benchmark cell is the mean of three runs; per-run values of the benchmark were not retained, so all benchmark tests treat scenarios, not seeds, as the unit of replication. Per-run values, including energy per epoch and realized density, are available for the controlled study (Appendix B).

Statistical tests. For each baseline, the paired differences of a configuration over scenarios are summarized by their median with a 95% percentile bootstrap confidence interval ($10{,}000$ resamples of scenarios) and tested with a two-sided Wilcoxon signed-rank test; p-values are Holm-corrected over the ten baselines, separately for each configuration and metric (Table 7). We make no claim that relies on significance across tables; Tables 5, 6, and 7 are alternative views of the same comparison, and the paper's conclusions are negative, so family-wise error across them would only make them more conservative.

Pre-specified configurations. Summarizing a family of variants by its best member per scenario would bias the statistics toward that family. We therefore compare only fixed configurations: EA-DST-I (EA+/random) and, for the density audit, EA+/released (EA+/EArandom in the result tables). The remaining variants appear only as ablations (Appendix K) and in the plug-in analysis (Table 9).

Scenarios, failures, exclusions. A scenario is a (dataset, model, density) triple with dense results for the same (dataset, model); there are 42. Runs with test accuracy $\leq 10\%$ did not converge and are excluded from all comparisons: SNIP on ConvNeXt, MobileNetV2, and ViT on CIFAR-10/100, on all Tiny-ImageNet models, and on all CIFAR-10 models at $d = 0.01$; uniform on MobileNetV2 at $d = 0.1$ (CIFAR-10/100) and on ViT at $d = 0.5, 0.9$ (CIFAR-10); CHT and CHTs on MobileNetV2 (CIFAR-100, $d = 0.1$). The uniform MobileNetV2 failure did not reproduce in the controlled harness (two seeds: 73.6% and 72.6% test accuracy on CIFAR-10 at $d = 0.1$), so these exclusions reflect

individual benchmark runs rather than intrinsic failures of the methods. The EA+/released cell for ViT on CIFAR-100 at $d = 0.1$ (3261.56 Wh against 256–301 Wh for all other methods) is excluded as a corrupted log.

Table 4: EA+/released minus EA-DST-I in the benchmark, paired over scenarios (median, range, and two-sided Wilcoxon p). The realized density of EA+/released differs from the nominal d (Table 10), so these differences are density effects.

d	n	ΔE (%): med. [min, max], p	ΔA (pts): med. [min, max], p
0.1	13	$+1.00\,[-3.35, +12.59]$, 0.19	$-1.02\,[-7.90, +1.30]$, 0.057
0.5	11	$-4.71\,[-14.24, -0.02]$, 0.001	$-1.24\,[-5.94, +0.90]$, 0.042
0.9	11	$-11.54\,[-22.44, +0.22]$, 0.005	$-0.82\,[-3.85, +0.40]$, 0.007

Outcome definitions. With ΔE the relative energy difference and ΔA the test-accuracy difference of configuration a over baseline b: *win* if ($\Delta E < -\tau_E$ and $\Delta A \geq -\tau_A$) or ($\Delta A > \tau_A$ and $\Delta E \leq \tau_E$); *loss* symmetrically; *tie* if $|\Delta E| \leq \tau_E$ and $|\Delta A| \leq \tau_A$; otherwise *trade-off*; $\tau_E = 1\%$, $\tau_A = 0.5$ points. Given the block-to-block variation observed in the controlled harness (Table 11), energy wins below a few percent should not be read as practically relevant.

Data audit. While parsing the tables we checked them for internal consistency. (i) Four RigL baseline rows of Supplementary Table S4 had been hidden by a typesetting error; they are restored, and all RigL statistics include them. (ii) The ablation rows $\alpha = 0$ and $\alpha = 0.5$ coincide with uniform/random and EA+/random, as expected by definition. (iii) On ViT, EA-uniform+ and uniform have identical accuracies in five scenarios, because EA-uniform+ changes only layers named `classifier`, which ViT does not have. (iv) In the $d = 0.01$ table (Table 32), the test accuracies of EA+/EArandom and EA+/random differ by exactly 3.00, 3.00, and 2.00 points, and three methods share 28.26 Wh on VGG-16. Without per-run logs these values cannot be re-derived, so the main analysis (Table 5) excludes the three $d = 0.01$ scenarios. With them included (Table 7), the accuracy differences of EA-DST-I to RigL and SRigL become nominally significant and its energy difference to SNIP does not; all other conclusions are unchanged. (v) All aggregate numbers are regenerated from the parsed tables by the released script.

Table 5: Benchmark: EA-DST-I vs. each baseline, paired over the scenarios at $d \geq 0.1$ in which both converged. W/T/L/X: wins/ties/losses/trade-offs; ΔE (%) and ΔA (points): median paired difference [95% bootstrap CI]; p: Holm-corrected two-sided Wilcoxon signed-rank test over scenarios ("–": $n < 6$). The version including $d = 0.01$ is Table 7.

Baseline	n	W/T/L/X	ΔE [CI]	p_E	ΔA [CI]	p_A
Uniform	35	16/13/4/2	$-0.21\,[-0.7, -0.1]$	0.003	$+0.04\,[-0.3, +0.8]$	0.451
ERK (SET)	39	16/10/10/3	$+0.05\,[-0.1, +0.2]$	0.863	$+0.21\,[-0.2, +1.2]$	1.000
SNIP	18	8/4/3/3	$-0.39\,[-0.9, +0.0]$	0.044	$+0.01\,[-0.9, +0.8]$	1.000
RigL	39	24/5/7/3	$-0.50\,[-1.0, -0.4]$	0.011	$+0.78\,[+0.1, +1.8]$	0.072
SRigL	18	18/0/0/0	$-16.27\,[-28.8, -10.1]$	<0.001	$+0.35\,[+0.0, +0.7]$	0.059
CHT	5	1/3/0/1	$-0.27\,[-1.9, +0.3]$	–	$+0.33\,[-1.7, +2.1]$	–
CHTs	20	8/3/5/4	$-0.69\,[-1.6, -0.1]$	0.017	$-0.42\,[-0.8, +0.5]$	1.000
CHTss	30	22/2/2/4	$-5.16\,[-7.2, -3.1]$	<0.001	$+0.20\,[-0.2, +1.1]$	0.409
MEST	39	15/7/9/8	$-0.72\,[-1.4, -0.1]$	0.011	$+0.09\,[-0.5, +0.4]$	1.000
DynaDiag	39	11/3/12/13	$+0.76\,[-0.6, +2.7]$	0.102	$+0.64\,[-0.1, +2.6]$	0.409

Energy-aware initialization (Table 5, Figure 4c). After Holm correction, EA-DST-I is statistically indistinguishable from every baseline in accuracy; the largest median differences, $+0.78$ points against RigL and $+0.35$ against SRigL, are not significant ($p = 0.07, 0.06$). Its energy is consistently lower than that of uniform, SNIP, RigL, CHTs, and MEST, but by only 0.2–0.7% in the median, an order of magnitude below the block-to-block variation of the controlled harness, so these differences have no practical relevance. The larger differences to CHTss (-5.2%) and SRigL (-16%) reflect the cost of their growth and constraint procedures, not the allocation. This is the outcome predicted by Table 22. Scenarios of the same model are not independent; aggregating first per model (six models) and testing with a sign test, no difference to any baseline, in energy or accuracy, is significant after Holm correction (Table 6). Excluding the under-trained ResNet-34 and ViT scenarios leaves the accuracy conclusion unchanged; the energy difference to uniform remains (-0.28%, $p = 0.004$), and those to SNIP, RigL, and MEST lose significance (Table 6).

Table 6: Robustness of the benchmark comparison of EA-DST-I ($d \geq 0.1$). Left: clustered by model; the median paired difference is computed per model and tested with a two-sided sign test over models ("$\Delta E < 0$": models in which EA-DST-I uses less energy; "$\Delta A > 0$": models in which it is more accurate), Holm-corrected over baselines; with six models the smallest attainable p is 0.031. Right: scenario-level test after excluding the under-trained ResNet-34 and ViT scenarios (median difference, Holm-corrected Wilcoxon p).

	Clustered by model					Without ResNet-34 and ViT				
Baseline	models	$\Delta E < 0$	p_E	$\Delta A > 0$	p_A	n	ΔE (%)	p_E	ΔA	p_A
Uniform	6	6/6	0.312	4/6	1.000	24	−0.28	0.004	−0.02	0.712
ERK (SET)	6	3/6	1.000	4/6	1.000	26	+0.07	0.960	−0.24	0.806
SNIP	3	2/3	1.000	2/3	1.000	12	−0.46	0.065	−0.30	0.806
RigL	6	5/6	1.000	6/6	0.312	26	−0.46	0.263	+0.56	0.131
SRigL	6	6/6	0.312	5/6	1.000	12	−25.35	0.004	+0.22	0.647
CHT	2	2/2	1.000	1/2	1.000	2	–	–	–	–
CHTs	6	5/6	1.000	2/6	1.000	11	−0.71	0.049	−0.40	0.806
CHTss	6	6/6	0.312	4/6	1.000	23	−4.68	<0.001	+0.28	0.561
MEST	6	4/6	1.000	3/6	1.000	26	−0.51	0.363	−0.30	0.561
DynaDiag	6	2/6	1.000	4/6	1.000	26	+1.45	0.018	+1.08	0.078

Table 7: Paired comparisons with each baseline over all 42 scenarios (including $d = 0.01$) in which both ran successfully. EA+/released uses the released, non-renormalized regrowth rule, so its differences include the density drift of Table 10 (tolerances 1% energy, 0.5 points test accuracy). n: scenarios; W/T/L/X: wins/ties/losses/trade-offs; ΔE: median paired energy difference (%, negative = less energy) with 95% bootstrap confidence interval over scenarios; ΔA: median paired test-accuracy difference (points) with 95% CI; p: two-sided Wilcoxon signed-rank test over scenarios, Holm-corrected within each configuration and metric.

	EA-DST-I (EA+ init, random regrowth)						**EA+/released** (realized density ≠ d)					
Baseline	n	W/T/L/X	ΔE [CI]	p_E	ΔA [CI]	p_A	n	W/T/L/X	ΔE [CI]	p_E	ΔA [CI]	p_A
Uniform	38	19/13/4/2	−0.23 [-0.6, -0.1]	0.001	+0.29 [-0.2, +0.9]	0.145	34	8/6/5/15	−1.49 [-5.1, -0.1]	0.008	−0.46 [-1.3, -0.1]	0.159
ERK (SET)	42	18/10/11/3	+0.01 [-0.2, +0.2]	0.536	+0.33 [-0.2, +1.3]	0.950	38	9/6/7/16	−1.28 [-3.5, -0.4]	0.003	−0.50 [-1.5, -0.1]	0.159
SNIP	18	8/4/3/3	−0.39 [-0.9, +0.0]	0.059	+0.01 [-0.9, +0.8]	1.000	15	3/3/3/6	−0.97 [-11.5, +0.2]	0.400	−0.69 [-1.3, +0.0]	0.287
RigL	42	27/5/7/3	−0.49 [-0.8, -0.3]	0.010	+0.79 [+0.2, +2.3]	0.018	38	14/2/4/18	−2.71 [-5.1, -0.9]	<0.001	−0.32 [-0.9, +0.8]	1.000
SRigL	21	2/0/0/0	−16.25 [-23.8, -9.9]	<0.001	+0.47 [+0.2, +2.0]	0.009	20	11/1/0/8	−17.00 [-28.4, -11.3]	<0.001	−0.17 [-0.9, +0.4]	1.000
CHT	8	4/3/0/1	−0.43 [-1.9, -0.1]	0.117	+1.26 [-0.2, +3.5]	0.391	7	1/1/0/5	−1.40 [-7.6, +0.1]	0.400	−1.12 [-2.8, +0.7]	1.000
CHTs	23	11/3/5/4	−0.66 [-1.5, -0.3]	0.009	−0.36 [-0.6, +1.6]	1.000	22	2/4/3/13	−3.61 [-7.8, -0.5]	0.008	−1.53 [-2.8, -0.3]	0.007
CHTss	33	25/2/2/4	−4.69 [-7.0, -3.1]	<0.001	+0.28 [-0.0, +1.4]	0.180	29	11/0/1/17	−7.43 [-9.5, -3.4]	<0.001	−0.82 [-1.3, +0.0]	1.000
MEST	42	17/7/10/8	−0.79 [-1.4, -0.5]	0.004	+0.08 [-0.4, +0.3]	1.000	38	7/2/7/22	−2.68 [-5.3, -1.1]	<0.001	−1.11 [-2.8, -0.6]	0.001
DynaDiag	42	12/3/12/15	+1.01 [-0.4, +2.5]	0.117	+0.76 [+0.1, +3.4]	0.127	38	8/1/2/27	−1.67 [-3.3, +1.1]	0.400	−0.01 [-2.4, +2.4]	1.000
EA+/released vs. EA-DST-I			–				38	4/5/9/20	−1.28 [-3.7, -0.0]	0.039	−1.09 [-2.4, -0.4]	<0.001

Versus dense training (Figure 4a–b). At density 0.1 the unstructured random-growth methods save a median 10–13% energy relative to dense training at a median cost of 2–4 points (EA-DST-I: 12.6%, −2.4 points; uniform: 11.9%, −1.8; ERK: 11.8%, −3.0; MEST: 12.0%, −1.8); DynaDiag saves 14.9% at −11.1 points. At densities 0.5 and 0.9, every method except EA+/released uses *more* energy than dense training in the median (typically −22% at 0.9). EA+/released alone shows a positive median saving at 0.5 (3.9%) and the smallest excess at 0.9 (−6.1%), but this pattern is what a loss of density during training would produce (Section 6.4), so we do not claim it as an energy advantage.

Per model (Figure 5). At $d = 0.1$, EA-DST-I, uniform, ERK, SNIP, RigL, and MEST save energy relative to dense training on every model–dataset pair (EA-DST-I: 3–20%), and within a column their savings differ by at most 5 points. The median accuracy cost is 2.5 points and depends far more on the model than on the allocation (up to 27 points for ConvNeXt and MobileNetV2 with RigL, SRigL, or CHTss; dense ResNet-34 and ViT are under-trained). EA-DST-I is never an outlier: it trails the most accurate of uniform, ERK, and MEST by at most 3.2 points and exceeds it by at most 2 points (ViT, CIFAR-10). The α ablation (Appendix K) is flat for ResNet-34; its ViT differences cannot stem from an allocation that deviates from uniform by less than 0.001.

Energy-aware initialization inside other DST methods (Table 9). Adding EA+ initialization to RigL, CHTs, CHTss, or MEST changes energy by at most 0.11% and accuracy by at most 0.4 points in the median; 71% of all 173 paired energy differences are below 1%. With SRigL and DynaDiag it raises energy (median +2.3% each), because it overrides their structured allocations.

Energy-aware regrowth in the benchmark. Relative to EA-DST-I, the benchmark's EA+/released uses a

Table 9: EA+/method − method (benchmark, $d \geq 0.1$; medians).

Base	n	ΔE (%)	ΔA	W/T/L/X
RigL	39	−0.11	+0.05	10/16/10/3
SRigL	15	+2.29	+0.00	2/3/8/2
CHTs	18	+0.07	−0.01	5/5/8/0
CHTss	24	−0.03	−0.40	1/12/10/1

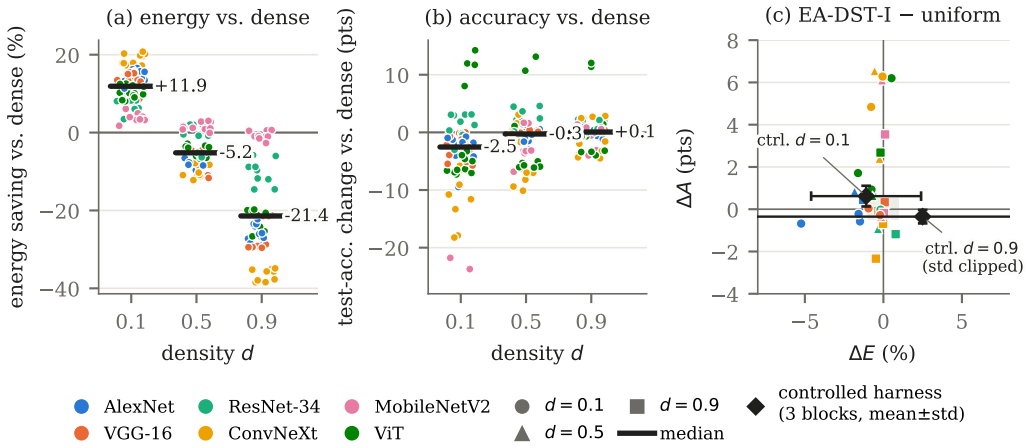

Figure 4: Benchmark evidence across all six models ($d \geq 0.1$). (a) Energy saving and (b) test-accuracy change vs. dense training of every converged run of EA-DST-I, uniform, ERK, RigL, and MEST (bars: medians). (c) EA-DST-I minus uniform per benchmark scenario (dots; shaded: tie region) and in the controlled harness (diamonds, mean ± std; the $d = 0.9$ energy std is clipped). Benchmark energies are whole-device estimates; the harness measures a GPU-energy penalty at $d = 0.1$ (Section 5).

(a) energy saving vs. dense (%), $d = 0.1$ (blue: saves energy; color scale clipped at ±25)

	Alex C10	VGG C10	Res34 C10	ConvN C10	MbV2 C10	ViT C10	Alex C100	VGG C100	Res34 C100	ConvN C100	MbV2 C100	ViT C100	Alex Tiny	VGG Tiny	ViT Tiny
EA-DST-I	+16	+9.5	+11	+20	+6.1	+9.8	+13	+13	+8.2	+17	+3.2	+13	+17	+16	+9.5
Uniform	+12	+8.7	+11	+20	--	+10	+12	+13	+8.1	+17	--	+12	+15	+15	+8.0
ERK (SET)	+12	+8.5	+11	+20	+5.0	+10	+14	+14	+8.2	+18	+3.4	+12	+16	+15	+8.4
SNIP	+11	+8.3	+11	--	--	--	+12	+14	+8.0	--	--	--	--	--	--
RigL	+13	+7.9	+6.3	+20	+4.1	+10	+11	+13	+3.4	+18	+4.0	+12	+16	+15	+8.3
SRigL	-70	-24	-3.6	--	--	--	--	--	--	+1.1	-8.0	+2.4	--	--	--
CHTss	+5.3	+5.1	--	+2.5	+5.8	--	+4.2	+10	+1.9	--	-1.3	+2.8	+13	+14	+5.1
MEST	+13	+8.0	+6.1	+21	+3.2	+9.8	+12	+13	+3.4	+18	+1.8	+12	+16	+15	+9.0
DynaDiag	+16	+14	+8.8	+21	+5.1	+13	+16	+18	+6.1	+19	+3.1	+15	+19	+18	+12

(b) test-accuracy change vs. dense (points), $d = 0.1$ (blue: more accurate)

	Alex C10	VGG C10	Res34 C10	ConvN C10	MbV2 C10	ViT C10	Alex C100	VGG C100	Res34 C100	ConvN C100	MbV2 C100	ViT C100	Alex Tiny	VGG Tiny	ViT Tiny
EA-DST-I	-2.4	-1.8	+2.9	-6.7	-4.2	+14	-2.0	-2.2	+3.1	-12	-3.4	-6.4	-1.0	-5.7	-3.3
Uniform	-1.7	-1.8	+3.1	-11	--	+8.0	-1.4	-1.9	+3.1	-18	--	-7.3	-0.8	-5.5	-5.0
ERK (SET)	-3.7	+0.0	-0.2	-6.4	-5.8	-0.4	-4.0	-1.2	-3.0	-11	-2.1	-7.0	-1.7	-2.5	-4.6
SNIP	-1.6	+0.3	+1.5	--	--	--	-0.7	+0.5	+0.6	--	--	--	--	--	--
RigL	-9.5	-1.8	+1.9	-13	-24	+12	-1.3	-3.9	+1.3	-18	-22	-6.9	-4.2	-5.1	-6.5
SRigL	-3.2	-1.9	+2.9	--	--	--	--	--	--	-19	-27	-7.0	--	--	--
CHTss	-1.4	-1.9	--	-9.1	-23	--	-0.6	-2.3	+3.1	--	-23	-6.6	-0.6	-5.3	-4.6
MEST	-2.5	-0.1	-0.4	-5.8	-1.2	+12	-2.9	-0.7	+0.3	-9.1	-1.8	-6.6	-1.8	-2.7	-3.9
DynaDiag	-11	-8.8	-1.6	-7.9	-15	+14	-14	-15	-7.3	-16	-16	-4.0	-11	-15	-4.3

Figure 5: Per-model view of the benchmark at density 0.1: energy saving (a) and test-accuracy change (b) relative to dense training for every model–dataset pair and density-preserving method ("–": no converged run available). Energy savings are similar across the unstructured random-growth methods within a column, whereas accuracy differs by more than 20 points in some columns; the structured methods SRigL and DynaDiag are the outliers in energy and accuracy, respectively. Energies are whole-device benchmark estimates (see Section 5 for the controlled GPU measurement).

median 4.7% less energy at $d = 0.5$ and 11.5% less at $d = 0.9$ (up to 22%), at a median cost of 1.2 and 0.8 points; at $d = 0.1$ it uses 1.0% more energy and loses 1.0 point (Table 4). Its energy barely grows with

Table 8: Per-model summary relative to dense training: median energy saving (%) / median test-accuracy change (points) over datasets and densities 0.1, 0.5, 0.9 (number of scenarios in brackets). EA+/released trains at a realized density that differs from the nominal one (Table 10). Failed runs excluded; medians over different subsets are not strictly comparable.

Method	AlexNet	VGG-16	ResNet-34	ConvNeXt	MobileNetV2	ViT
EA+/released	+5.4 / −2.3 (7)	−3.5 / +0.1 (7)	+2.0 / +2.2 (3)	+4.9 / −9.8 (6)	+3.2 / −5.1 (6)	+1.4 / −2.0 (6)
EA-DST-I	−7.6 / −0.8 (7)	−4.6 / −0.1 (7)	+0.8 / +2.6 (6)	−9.5 / −5.6 (6)	+2.0 / −2.1 (6)	−2.9 / −0.7 (7)
Uniform	−8.4 / −1.2 (7)	−4.4 / +0.1 (7)	+0.7 / +2.7 (6)	−9.9 / −8.8 (6)	+0.8 / −3.6 (4)	+8.0 / −5.0 (5)
ERK (SET)	−4.6 / −1.5 (7)	−4.4 / +0.2 (7)	+0.6 / +0.3 (6)	−9.6 / −1.1 (6)	+2.2 / −2.6 (6)	−3.9 / −4.6 (7)
SNIP	−8.5 / −1.1 (6)	−7.7 / +0.4 (6)	−3.5 / +1.5 (6)	–	–	–
RigL	−8.2 / −1.3 (7)	−4.6 / −0.2 (7)	−4.3 / +1.6 (6)	−9.5 / −8.4 (6)	+0.8 / −4.7 (6)	−3.4 / −3.2 (7)
CHTss	−16.0 / −0.6 (7)	−7.1 / −0.4 (7)	−7.2 / +3.1 (3)	−26.3 / −3.5 (3)	−5.5 / −7.7 (6)	−4.5 / −4.8 (4)
MEST	−5.2 / −1.6 (7)	−4.5 / +0.1 (7)	−4.9 / +0.5 (6)	−10.0 / −5.1 (6)	+0.6 / −0.3 (6)	−3.8 / −3.4 (7)
DynaDiag	+3.7 / −2.1 (7)	−1.6 / −0.8 (7)	−2.7 / +1.6 (6)	−8.2 / −4.1 (6)	+1.3 / −4.2 (6)	−1.5 / −0.4 (7)

density (AlexNet, CIFAR-10: 20.1, 22.0, 22.7 Wh at $d = 0.1, 0.5, 0.9$; EA-DST-I: 19.2, 24.6, 29.1 Wh), the signature of a rule that does not conserve density (Section 6.4).

Ablations and proxy fidelity. (Appendix K). With the original implementation, test accuracy of ResNet-34 (CIFAR-100) is flat for $\alpha \in [0, 0.5]$ and falls by 2 points at $\alpha = 1$, while energy decreases by at most 1.6%; on ViT, $\alpha = 0.5$ is 6.2 points (CIFAR-10) and 1.7 points (Tiny-ImageNet) more accurate than uniform allocation, a difference far larger than the allocation change (mean density deviation 0.0008, Table 22) can plausibly cause; these are single-configuration results without spread, and we do not interpret them. In the controlled study with five seeds, EA+ ($\alpha = 0.5$) is within 0.2 points of uniform allocation on AlexNet and VGG-16 (Section B.1). **Proxy fidelity.** Per-layer measured energies agree with the earlier score's ranking ($\rho = 0.88$–0.98; Table 22, Appendix H). **Processors and other tasks.** Runs on two CPUs and two GPUs test the portability of the pipeline, not the allocation, since CPU and GPU scores rank layers identically by construction (Appendix O).

B Replay of the Released Mask Updates and Controlled-Study Details

Replay. `sim_drift.py` instantiates each model and the *unchanged* `Masking` class of the released implementation, initializes the masks with EA+ (or uniform), and calls its topology update every 100 iterations with the death rate of the released cosine schedule (0.5 decayed to 0.005 over $\lceil 45000/128 \rceil \times$epochs iterations, with the epochs of Table 26). Before each update, the active weights are redrawn from a standard normal distribution, so magnitude removal selects a random subset. The number of weights removed ($\lceil \zeta_t \, \mathrm{nnz}^l \rceil$) and the growth probabilities depend only on counts, so the replay reproduces the density trajectory of training up to the randomness of the Bernoulli growth draws. Densities are recorded every 1000 iterations.

Table 10: Global density at the end of training under the released regrowth rule (EA+ initialization, growth probability scaled by $1 + s^l - 1/L$, no renormalization), from an exact replay of the released masking code with the benchmark schedule (Appendix B). Columns: nominal density. Under the same replay, uniform and EA+ initialization with random regrowth end within 0.006 of the nominal density (AlexNet).

Model	Epochs	$d = 0.1$	$d = 0.5$	$d = 0.9$
AlexNet	100	0.167	0.196	0.226
VGG-16	100	0.438	0.442	0.446
ResNet-34	20	0.120	0.565	0.947
ConvNeXt	60	0.064	0.222	0.385
MobileNetV2	50	0.077	0.241	0.407
ViT	100	0.065	0.220	0.383

Mechanism. In the released rule, layer l regrows each inactive weight with probability $q^l = p^l(1 + s^l - 1/L)$, where p^l is the number removed divided by the number of inactive weights, unless

$p^l(1 + s^l - 1/L) < 0.02$. The expected regrowth is $N^l_{\text{grow}}(1 + s^l - 1/L)$, so a layer with $s^l < 1/L$ loses a fraction $\zeta_t(1/L - s^l)$ of its weights at every update, and a layer with $s^l > 1/L$ gains, until it is fully dense. Most layers have $s^l < 1/L$ (the score is concentrated; Table 22), so the network thins; at high density, the few high-score layers saturate at density 1 and cannot absorb the difference.

Controlled study. The runs use the GPU-resident data pipeline of `ea_v6.py` (random crop with reflection padding and horizontal flip applied on the GPU, same normalization as the benchmark), a fixed $45{,}000/5{,}000$ train/validation split, SGD with Nesterov momentum 0.9, learning rate 0.1 divided by 10 at 50% and 75% of training, weight decay $5 \cdot 10^{-4}$, batch size 128, and 30 epochs. Blocks run the dense reference and the seven sparse configurations in an order drawn from a fixed random seed. For each epoch we log the GPU energy, time, enforced power limit, and temperature. The laptop GPU's power limit is set dynamically by the platform between 35 and 45 W: dense and dense-mask runs are GPU-bound and run near 44 W, while sparse-tensor mask runs are CPU-bound and run near 35 W with epoch times that vary between blocks (up to $1.3\times$). We report all runs; no run was repeated or excluded. Before this design, a sequential pilot of nine runs showed that two runs executed while other CPU-heavy work was running on the machine used 38% and 61% more energy than the same configurations in the block design; we then set the pilot aside and ran the block design, during which only light interactive work (text editing, short analysis scripts) ran on the machine; the block-level variation described above remains. The pilot logs are included in the supplementary material; they agree with the block results in accuracy and in final density.

Table 11: Controlled study: AlexNet on CIFAR-10, 30 epochs, three blocks (seeds); in each block all methods and a dense reference run in random order on the RTX A500, GPU energy from NVML. Acc.: test accuracy of the checkpoint selected on the validation split; d_T: global density at the end of training; ΔE_{dense}: energy relative to the dense run of the same block; time: training time relative to dense; ΔA, ΔE: paired differences to uniform allocation in the same block. Mean $\pm$ std over blocks.

Method	**Acc. (%)**	d_T	ΔE_{dense} (%)	**Time**	ΔA (pts)	ΔE (%)
Density 0.1 (dense: $83.65 \pm 0.27\%$, 1.860 ± 0.008 Wh)						
Uniform	80.86 ± 0.36	0.100	$+29.5 \pm 22.0$	$1.55\times$	–	–
ERK	79.03 ± 0.16	0.100	$+18.8 \pm 17.1$	$1.45\times$	-1.83 ± 0.41	-7.6 ± 9.6
FLOP-share (EA+ formula)	81.04 ± 0.64	0.100	$+25.1 \pm 19.6$	$1.49\times$	$+0.18 \pm 0.53$	-3.2 ± 2.7
EA-DST-I	81.48 ± 0.36	0.100	$+27.9 \pm 21.2$	$1.53\times$	$+0.62 \pm 0.49$	-1.1 ± 3.5
EA-DST (Eq. 4, renormalized)	71.11 ± 0.62	0.100	$+19.5 \pm 24.9$	$1.46\times$	-9.75 ± 0.29	-6.6 ± 18.9
EA+ / released regrowth rule	82.41 ± 0.15	0.293	$+40.3 \pm 24.5$	$1.66\times$	$+1.55 \pm 0.22$	$+8.7 \pm 9.5$
Uniform, dense mask application	81.52 ± 0.35	0.100	$+6.1 \pm 0.9$	$1.06\times$	$+0.66 \pm 0.41$	-16.3 ± 15.1
Density 0.9 (dense: $83.53 \pm 0.21\%$, 1.861 ± 0.028 Wh)						
Uniform	83.66 ± 0.37	0.900	$+88.3 \pm 27.5$	$2.20\times$	–	–
ERK	83.53 ± 0.53	0.900	$+72.1 \pm 20.9$	$2.06\times$	-0.13 ± 0.22	-8.2 ± 7.4
FLOP-share (EA+ formula)	83.37 ± 0.22	0.900	$+82.5 \pm 22.0$	$2.19\times$	-0.29 ± 0.47	-2.7 ± 7.3
EA-DST-I	83.31 ± 0.25	0.900	$+91.1 \pm 27.0$	$2.19\times$	-0.35 ± 0.32	$+2.5 \pm 17.2$
EA-DST (Eq. 4, renormalized)	83.59 ± 0.40	0.900	$+79.7 \pm 27.8$	$2.14\times$	-0.07 ± 0.13	-4.4 ± 8.1
EA+ / released regrowth rule	83.30 ± 0.47	0.380	$+41.5 \pm 17.8$	$1.70\times$	-0.36 ± 0.23	-24.5 ± 6.8
Uniform, dense mask application	83.82 ± 0.21	0.900	$+4.6 \pm 0.7$	$1.05\times$	$+0.16 \pm 0.16$	-43.6 ± 8.4

Energy variability and resolution. The dense reference is stable across blocks (1.860 ± 0.008 Wh at $d = 0.1$), but the energy of the sparse-tensor configurations relative to dense varies by 17–28% between blocks (Table 11), and paired differences to uniform allocation vary by 3–19%. Energy does not depend on the position of a run within its block (Spearman $\rho = 0.10$, $p = 0.55$; runtime $\rho = 0.05$, $p = 0.78$), and GPU temperature averaged 67–72°C per run. The dominant source of variation is the block: the mean runtime of the sparse-tensor configurations at $d = 0.1$ is 201 s in the first block and 237 s and 257 s in the second and third. The platform-managed power limit is a second source: in 20 of the 36 sparse-tensor runs its mean exceeded 36 W, and it correlates with the mean GPU power (Spearman $\rho = 0.94$) and, more weakly, with runtime ($\rho = 0.50$). The driver reports the power limit of this laptop GPU as not settable, so we could not fix it; pairing within blocks removes the block component of both effects. With three blocks, the 95% confidence half-width of a paired energy difference is $4.30\, s/\sqrt{3}$, i.e., 8.7% for EA-DST-I vs. uniform at $d = 0.1$ under sparse-tensor masking, and about eight blocks would be needed to resolve 3%. This resolution concerns the masked back ends, where the allocation cannot change the arithmetic anyway; under the CSR back end the paired standard deviations are 0.3–2% at $d = 0.1$ (Table 2), so the allocation effects reported there are resolved. For accuracy, five seeds give a 95% half-width of 0.9 points for EA-DST-I vs. uniform on AlexNet (Table 12).

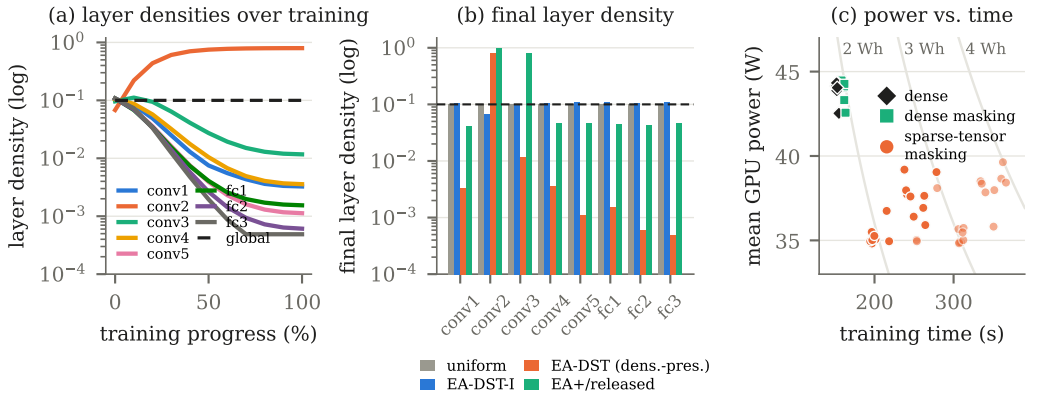

Figure 6: Controlled-study diagnostics (AlexNet, CIFAR-10; means over three blocks). (a) Per-layer density under the renormalized EA-random rule (EA-DST) at $d = 0.1$: the global density stays at 0.1 (dashed), but conv2 grows to 0.80 while all other layers fall to 0.0005–0.012. (b) Final per-layer densities of four configurations at $d = 0.1$ (log scale; dashed: nominal density). (c) Mean GPU power vs. training time of every run (gray lines: equal energy); sparse-tensor masking runs at lower power but for much longer than dense training or dense masking (faded: $d = 0.9$).

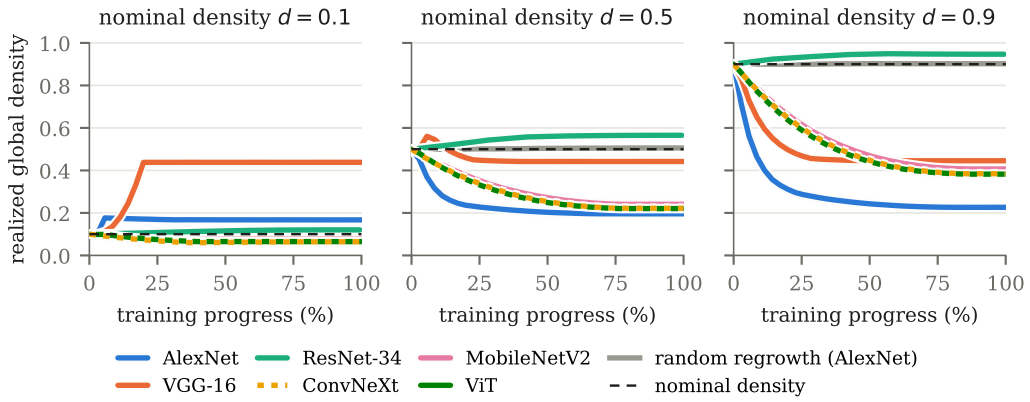

Figure 7: Realized global density during training under the released regrowth rule (colored; one line per model, benchmark schedule) and under random regrowth (gray, AlexNet), from the replay of the released mask updates. Dashed: nominal density. ConvNeXt (dotted) and ViT nearly coincide. Final values are listed in Table 10.

B.1 Accuracy at a Fixed Density

With five seeds, no EA+ strength changes accuracy on AlexNet or VGG-16 after Holm correction (Table 12): the largest effect is -0.62 points for $\alpha = 1$ on AlexNet (corrected $p = 0.20$), and on VGG-16 every EA+ variant is within 0.1 points of uniform. ERK, whose allocation differs from uniform far more than EA+ does (Appendix F), is 2.3 points worse on AlexNet and 1.4 points better on VGG-16. Shape-based allocations can therefore matter for accuracy, while the score-based one is too close to uniform to do so. The legacy benchmark tells the same story at scale: across 39 scenarios with ten DST baselines, EA-DST-I is statistically indistinguishable from every baseline in accuracy, and adding EA+ initialization to RigL, CHTs, CHTss, or MEST changes energy by at most 0.11% in the median (Appendix A.1).

MobileNetV2 check. MobileNetV2 on CIFAR-10 at $d = 0.1$, 20 epochs, uniform and EA-DST-I with random regrowth, seeds 17 and 18: test accuracy 73.60 / 72.58% (uniform) and 73.25 / 73.73%

Table 12: Test accuracy at $d = 0.1$ on CIFAR-10 with random regrowth (mean $\pm$ std over seeds). ΔA: paired difference to uniform allocation with the same seed; p_{Holm}: two-sided paired t-test, Holm-corrected over the allocations of each model. Accuracy does not depend on the execution back end; these are the accuracies used in Table 2.

Allocation	**Acc. (%)**	n	ΔA (pts)	p_{Holm}
AlexNet, 30 epochs				
Uniform (α=0)	81.18 ± 0.54	5	–	–
EA+ (α=0.25)	81.16 ± 0.15	5	-0.02 ± 0.65	1.000
EA+ (α=0.5) = EA-DST-I	81.37 ± 0.30	5	$+0.19 \pm 0.71$	1.000
EA+ (α=0.75)	80.85 ± 0.27	5	-0.33 ± 0.46	0.734
EA+ (α=1)	80.56 ± 0.27	5	-0.62 ± 0.47	0.201
FLOP-share (α=0.5)	80.97 ± 0.46	5	-0.21 ± 0.67	1.000
ERK	78.91 ± 0.22	5	-2.27 ± 0.72	0.013
VGG-16, 20 epochs				
Uniform (α=0)	88.74 ± 0.28	5	–	–
EA+ (α=0.5) = EA-DST-I	88.81 ± 0.06	5	$+0.07 \pm 0.23$	1.000
EA+ (α=1)	88.75 ± 0.18	5	$+0.01 \pm 0.44$	1.000
FLOP-share (α=0.5)	88.81 ± 0.10	5	$+0.07 \pm 0.29$	1.000
ERK	90.12 ± 0.13	5	$+1.38 \pm 0.28$	0.002

(EA-DST-I). The minimum validation accuracy over the first six epochs is 9.7 / 10.0% (uniform) and 25.8 / 9.4% (EA-DST-I).

B.2 Proof of Proposition 6.1

Removal takes $\zeta_t n_t^l$ weights from layer l, so $K_t = \zeta_t \sum_i n_t^i$. (a) Without renormalization layer l regrows $f^l \zeta_t n_t^l$, hence $n_{t+1}^l = n_t^l(1 - \zeta_t + \zeta_t f^l)$; summing over l gives the claim. (b) With renormalization layer l regrows $K_t f^l n_t^l / \sum_i f^i n_t^i = \zeta_t \big(\sum_i n_t^i\big) f^l x_t^l / \bar{f}_t$, so $n_{t+1}^l = n_t^l(1 - \zeta_t) + \zeta_t \big(\sum_i n_t^i\big) f^l x_t^l / \bar{f}_t$; the total is unchanged, and dividing by it gives the recursion. For $f^l > f^m$,

$$\frac{x_{t+1}^l / x_{t+1}^m}{x_t^l / x_t^m} = \frac{1 - \zeta_t + \zeta_t f^l / \bar{f}_t}{1 - \zeta_t + \zeta_t f^m / \bar{f}_t} = 1 + \frac{\zeta_t (f^l - f^m)/\bar{f}_t}{1 - \zeta_t + \zeta_t f^m / \bar{f}_t} \geq 1 + \zeta_t \, \frac{f^l - f^m}{\max_i f^i} \cdot \frac{1}{1 + \max_i f^i / \min_i f^i},$$

using $\min_i f^i \leq \bar{f}_t \leq \max_i f^i$ and $\zeta_t < 1$. With $\log(1 + y) \geq y/2$ for $y \in [0, 1]$, the ratio grows by at least $\exp(c \sum_t \zeta_t)$ with $c = \frac{1}{2}(f^l - f^m) / \big(\max_i f^i (1 + \max_i f^i / \min_i f^i)\big)$. If $\sum_t \zeta_t \to \infty$, every layer whose f is below the maximum loses its share relative to the maximizers. (c) From the recursion, $x_{t+1}^l = x_t^l$ for all l with $x_t^l > 0$ iff $f^l = \bar{f}_t$ for all of them, i.e., all active layers have equal factors; renormalization then assigns each layer exactly its removed count. (d) Since the global count is conserved, $\sum_l n_t^l = \sum_l n_\star^l$, so the anchor terms sum to zero and the total regrowth is still K_t. With $e_t^l = n_t^l - n_\star^l$, Eq. 7 gives $e_{t+1}^l = (1 - \eta) e_t^l + \zeta_t n_t^l \big(f^l / \bar{f}_t - 1\big)$, and $|f^l / \bar{f}_t - 1| \leq \delta$ because $\min_i f^i \leq \bar{f}_t \leq \max_i f^i$. Writing $n_t^l = n_\star^l + e_t^l$ gives $|e_{t+1}^l| \leq (1 - \eta + \zeta_{\max}\delta)|e_t^l| + \zeta_{\max}\delta \, n_\star^l$, a contraction when $\zeta_{\max}\delta < \eta$, whose fixed point is the stated bound (we ignore the cap at full density and the clipping of negative budgets). The bound is loose: for AlexNet with $\beta = 0.5$ and without the gradient term it is 0.6, whereas the largest deviation measured in training is 0.6% (Section 6.4).

Magnitude in practice. With the cosine schedule from 0.5 to 0.005, $\sum_t \zeta_t \approx 0.25$ per update, i.e., about 88 for AlexNet's 352 updates over 100 epochs and about 26 over the 30 epochs of the controlled study. With $\beta = 0.5$, two AlexNet layers with shares 0.4 and 0.01 differ by $f^l - f^m \approx 0.2$, so their ratio grows per update by about $1 + \zeta_t(f^l - f^m)/\bar{f}_t$, i.e., by a factor of roughly $\exp(0.2 \sum_t \zeta_t) \approx e^5$ over the controlled study and e^{17} over the benchmark schedule (the bound in (b) is looser); this matches the 80%-dense second convolution in Section 6.4. The cap at full density stops a layer's growth once it is dense, which is why, in the unrenormalized form, the first layers saturate at density 1 and the remaining budget change appears as global drift.

B.3 Share Dynamics on the Training Logs

Check against the logs. Figure 8 plots $\ln(d^{\text{conv2}}/d^l)$ for the renormalized EA-random rule (EA-DST) at $d = 0.1$ (AlexNet, mean of three blocks) against $\sum_u \zeta_u$, computed from the schedule of the harness ($\zeta_0 = 0.5$, cosine decay, update every 100 iterations). All seven curves are monotone and nearly linear (Pearson $r = 0.996$–1.000), the winning layer is the one with the highest score share (lowest

initial density), and conv3, the layer with the second-highest score share, diverges least. The gradient term γ makes f^l time-varying, which explains why the slopes do not follow the initial score shares exactly. At $d = 0.9$ the dynamic saturates six layers at density 1, as predicted by Proposition 6.1(b) together with the cap at full density.

Per-seed fits, a second model, and simulation. Fitted per seed rather than on the block mean, all AlexNet curves at $d = 0.1$ have Pearson $r \geq 0.995$. Table 13 iterates the recursion of Proposition 6.1(b) with expected counts (no training) for all six models, and Table 14 reports the trained runs of both regrowth rules on AlexNet and VGG-16. On VGG-16 the per-seed fits give $r \geq 0.987$. The simulation, whose only inputs are the layer sizes and score shares, predicts the densest and the sparsest layers of the trained VGG-16 runs closely (0.179 vs. 0.181; 0.043 vs. 0.033) and the other layers within 0.04 in absolute density (Figure 3c); we attribute the residual to the omitted gradient term. Because the gradient term is omitted and the cap at full density is enforced, it is a prediction of where concentration is strong, not of accuracy: on VGG-16, a four-fold spread in layer densities within 20 epochs costs no accuracy.

Table 13: Expected-count simulation of renormalized score-weighted regrowth (Proposition 6.1(b); EA+ initialization at $d = 0.1$, $\beta = 0.5$, gradient term omitted, cosine drop schedule, update every 100 iterations; no training) with the real layer sizes and score shares of each model. max s^l: largest score share; top d^l: final density of the densest layer; < 0.01: fraction of layers that end below density 0.01; top-3 share: fraction of all active weights held by the three layers with the most active weights, at initialization $\rightarrow$ end; last two columns: top d^l with the 30-epoch controlled schedule, and the largest relative deviation $|d^l/d^l_\star - 1|$ of any layer from its initial allocation under the anchored rule (Eq. 7, $\eta = 0.5$, benchmark schedule).

			Renormalized, benchmark schedule				**30 ep.**	**Anchored**
Model	L	max s^l	epochs	top d^l	< 0.01	top-3 share	top d^l	max dev.
AlexNet	8	0.69	100	0.82	0.88	0.68→1.00	0.81	0.003
VGG-16	14	0.18	100	0.30	0.57	0.50→0.98	0.22	0.000
ResNet-34	37	0.05	20	0.11	0.00	0.33→0.35	0.11	0.000
ConvNeXt	59	0.07	60	0.16	0.00	0.25→0.40	0.13	0.000
MobileNetV2	53	0.23	50	0.46	0.94	0.38→0.95	0.35	0.001
ViT	38	0.04	100	0.11	0.00	0.11→0.12	0.10	0.000

Table 14: Regrowth rules at $d = 0.1$ with EA+ initialization ($\alpha = 0.5$, $\beta = \gamma = 0.5$, $\eta = 0.5$) on CIFAR-10. ΔA: paired difference to uniform allocation with random regrowth (same seed); p_{Holm}: paired t-test, Holm-corrected within each model; top / min: final density of the densest and of the sparsest layer (means over seeds); dev: largest relative deviation $|d^l_T/d^l_0 - 1|$ of any layer from its initial density. The global density is exactly 0.1 in every run. Random regrowth keeps every layer at its initial density by construction.

Initialization, regrowth	**Acc. (%)**	n	ΔA (pts)	p_{Holm}	top / min d^l_T	dev
AlexNet, $d = 0.1$, 30 epochs						
Uniform, random regrowth	81.18 ± 0.54	5	–	–	0.100 / 0.1000	0.00
EA+, random regrowth (EA-DST-I)	81.37 ± 0.30	5	$+0.19 \pm 0.71$	0.776	0.108 / 0.0681	0.00
EA+, renormalized EA-random (EA-DST)	71.42 ± 0.61	5	-9.77 ± 0.24	<0.001	0.796 / 0.0005	10.70
EA+, anchored EA-random (Eq. 7)	81.38 ± 0.24	5	$+0.20 \pm 0.46$	0.776	0.108 / 0.0685	0.01
VGG-16, $d = 0.1$, 20 epochs						
Uniform, random regrowth	88.74 ± 0.28	5	–	–	0.100 / 0.1000	0.00
EA+, random regrowth (EA-DST-I)	88.81 ± 0.06	5	$+0.07 \pm 0.23$	0.938	0.105 / 0.0950	0.00
EA+, renormalized EA-random (EA-DST)	88.96 ± 0.35	3	$+0.27 \pm 0.20$	0.449	0.181 / 0.0327	0.91
EA+, anchored EA-random (Eq. 7)	88.84 ± 0.10	3	$+0.15 \pm 0.29$	0.938	0.104 / 0.0951	0.01

Strength of the anchor. Table 15 repeats the AlexNet runs of Table 14 with weaker anchors (one or two seeds each, a limit set by our compute budget). Weaker anchors let the allocation drift further, but accuracy degrades gracefully: $\eta = 0.2$: $+0.59$ points, largest deviation 2%; $\eta = 0.05$: -0.39 points, largest deviation 130%; $\eta = 0.01$: -1.56 points, largest deviation 688%. Without an anchor the same rule loses 9.8 points.

Implication. A growth rule of the form $N^l_{\text{grow}} \propto f^l N^l_{\text{rm}}$ with fixed or slowly varying, unequal f^l is unstable under any global renormalization (Proposition 6.1). Stable energy-aware regrowth requires per-layer targets, as in the anchored rule of Proposition 6.1(d), which we ran with $\eta = 0.5$ toward

Table 15: Strength of the anchor (Eq. 7) on AlexNet at $d = 0.1$ with EA+ initialization, $\beta = \gamma = 0.5$, 30 epochs. ΔA: paired difference to uniform allocation with random regrowth (same seed); p_{Holm}: paired t-test, Holm-corrected over the rows; top d_T^l: final density of the densest layer (its initial density is 0.108); dev: largest relative deviation of any layer from its initial density.

η	n	**Acc. (%)**	ΔA (pts)	p_{Holm}	top d_T^l	dev
0.5	5	81.38 ± 0.24	$+0.20 \pm 0.46$	0.388	0.108	0.6%
0.2	2	81.62 ± 0.06	$+0.59 \pm 0.35$	–	0.107	2.5%
0.05	1	80.85 ± 0.00	-0.39 ± 0.00	–	0.156	129.7%
0.01	1	79.68 ± 0.00	-1.56 ± 0.00	–	0.537	688.5%
0 (renormalized)	3	71.11 ± 0.62	-9.75 ± 0.29	<0.001	0.797	1070.1%

the EA+ initialization: its largest per-layer deviation is 0.6% on AlexNet and 0.6% on VGG-16 (Table 14).

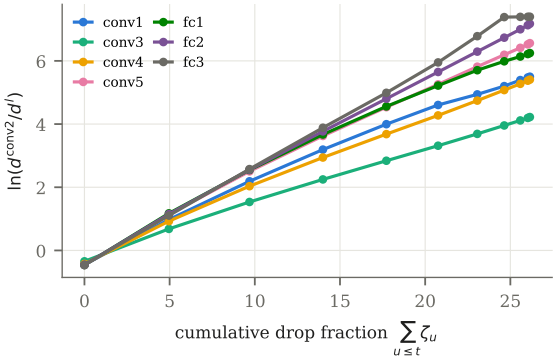

Figure 8: Log density ratio of the winning layer (conv2) to every other layer under the renormalized EA-random rule (EA-DST) ($d = 0.1$, AlexNet, mean of three blocks) against the cumulative drop fraction. The near-linear, monotone growth is the signature of the replicator dynamic of Proposition 6.1(b).

C SKIP-ZERO CSR BACK END

`csr_backend.py` replaces the forward of every sparsified `nn.Conv2d` and `nn.Linear` module. For a layer with weight W (reshaped to $C_{\text{out}} \times C_{\text{in}}K^2$ for convolutions) and mask M, the non-zero positions are stored as a CSR index set (crow, col) and a second index set for $W^\top$ with a permutation of the values; both are rebuilt after every topology update. With X the layer input (for convolutions, the im2col matrix `unfold`(x) of size $C_{\text{in}}K^2 \times BHW$):

$$Y = W_{\text{csr}}X, \qquad \frac{\partial \mathcal{L}}{\partial X} = W_{\text{csr}}^\top \frac{\partial \mathcal{L}}{\partial Y}, \qquad \frac{\partial \mathcal{L}}{\partial W}\Big|_M = \text{sampled}_M\Big(\frac{\partial \mathcal{L}}{\partial Y}X^\top\Big),$$

computed with `torch.sparse.mm` (cuSPARSE SpMM) and `torch.sparse.sampled_addmm` (cuSPARSE SDDMM). The values are gathered from the dense weight tensor at the non-zero positions in every forward pass (cost proportional to nnz), and the sparse weight gradient is scattered back, so the optimizer (SGD with momentum and weight decay) runs on the dense tensor; gradients off the mask are exactly zero and weight decay keeps zeros at zero, so no mask multiplication is needed between topology updates. Against dense masked computation on AlexNet at $d = 0.1$, the forward outputs agree exactly and the weight gradients agree to within $1.5 \cdot 10^{-9}$, with exactly zero gradient off the mask; test accuracy after two epochs is 58.16% (CSR) and 58.20% (dense masks). Grouped convolutions are not supported, which excludes MobileNetV2 and ConvNeXt. The CSR runs are two epochs (AlexNet) or one epoch (VGG-16) long, and we report energy per epoch, which differs between the two epochs of an AlexNet run by a median of 2.0% (maximum 9.7%); topology updates are included. Where the time goes: on AlexNet at $d = 0.1$ a CSR epoch takes 51–71 s against 5–6 s dense; the im2col matrices of the early

convolutions have $B \cdot H \cdot W$ up to 32,768 columns, for which cuSPARSE SpMM and SDDMM at 90% sparsity are an order of magnitude slower than the corresponding cuDNN convolutions on this GPU. **Per-layer benchmark** (`layer_bench.py`). For every sparsified layer of AlexNet and VGG-16 at the CIFAR training shapes (batch 128), we time one forward and backward pass of the layer alone for at least 2 s with the dense kernel and with the CSR back end at layer densities 0.01, 0.05, 0.1, 0.3, and 0.5, and read the NVML energy. At $d = 0.1$ the CSR layer costs 1.1–24.7× (AlexNet) and 1.2–28.9× (VGG-16) the dense layer's energy; the CSR layer is cheaper than or equal to the dense one only for AlexNet's first convolution at $d = 0.01$. The score's ranking of the layers agrees with their measured CSR energy at $d = 0.1$ with Spearman $\rho = 0.976$ (AlexNet) and 0.940 (VGG-16), against 0.952 and 0.910 for FLOPs. On AlexNet the difference is one adjacent rank swap ($\sum \delta^2 = 2$ vs. 4), and on VGG-16 it is a sum of squared rank differences of 27 vs. 41; with 8 and 14 layers, neither is evidence that the score ranks layers better than FLOPs.

Additive per-layer model and the measured-cost optimum (`csr_additive.py`). For an allocation $\{d^l\}$, we predict the energy of the sparsified layers as $\sum_l \hat{E}^l(d^l)$, where $\hat{E}^l$ interpolates the isolated per-layer CSR measurements above piecewise linearly in the density. The per-layer costs are strongly non-monotone in density (AlexNet's conv2 costs 3× more at $d = 0.1$ than at $d = 0.3$; we did not investigate the kernel-level cause), so we do not assume any functional form. The model has no free parameter and never sees the training runs. It predicts the uniform-allocation epoch energy of AlexNet within 4% (the remainder is pooling, normalization, loss, and the optimizer) and every allocation effect of Table 2 on AlexNet within 2.8 points (Table 16). Minimizing the same sum under the global budget by dynamic programming over a density grid (0.01–0.5) gives the *measured-cost optimum*: every AlexNet convolution at 0.01 and the classifier layers at 0.5, 0.32, and 0.10. We trained it with five seeds and measured its CSR energy in three blocks with their own uniform and dense references (Table 17). It saves 70.4% energy but loses 10.75 points of accuracy (70.44%), whereas EA+ at $\alpha = 1$ saves 26.7% at 0.6 points: energy-only allocation reproduces the extreme solution of the stylized model in Appendix G.

Table 16: Additive per-layer model of CSR training energy at $d = 0.1$: the energy of the sparsified layers predicted by summing the per-layer CSR costs measured in isolation (`layer_bench.py`; piecewise-linear interpolation between layer densities 0.01, 0.05, 0.1, 0.3, 0.5), as a change relative to uniform allocation, against the measured change per epoch (AlexNet α family: joint blocks; other rows: Table 2; optimum: Table 17). Predicted uniform energy of the sparsified layers: 0.808 Wh/epoch against 0.840 Wh/epoch measured for the whole network. ERK on VGG-16 places the early, high-resolution convolutions at densities 0.5–1, beyond the benchmarked range, where the model extrapolates.

Model	Allocation	Predicted ΔE (%)	Measured ΔE (%)
AlexNet	EA+ (α=0.25)	−5.3	−5.0
AlexNet	EA+ (α=0.5)	−10.7	−11.3
AlexNet	EA+ (α=0.75)	−16.0	−18.5
AlexNet	EA+ (α=1)	−26.8	−26.7
AlexNet	FLOP-share (α=0.5)	−8.0	−6.8
AlexNet	ERK	−22.9	−25.7
AlexNet	Measured-cost optimum	−68.5	−70.4
VGG-16	EA+ (α=0.5)	−1.6	−0.7
VGG-16	EA+ (α=1)	−3.2	−1.8
VGG-16	FLOP-share (α=0.5)	−1.2	−0.2
VGG-16	ERK	−6.3	+6.4

Why the CSR cost is not monotone in density (`csr_kernels2.py`). We timed every operation of the back end separately, exactly as `csr_backend.py` executes it (same index structures, same operand layouts), for AlexNet's second and third convolutions at the CIFAR training shapes (batch 128), at least half a second per measurement, and read the NVML energy (Table 18). The value gather, the gradient scatter, and the copy of the transposed input are negligible or independent of the density, and the forward and input-gradient SpMM products grow roughly in proportion to it. The weight-gradient SDDMM dominates: on the second convolution it accounts for 97% of the layer's energy at density 0.1, stays in a slow regime from density 0.05 to 0.2, and becomes 13× cheaper at 0.3; the whole layer is 2.6× more expensive at density 0.1 than at 0.3. Our back end passes the dense operand of this product as a contiguous transposed copy. Repeating the product with a transposed view, the layout of our first microbenchmark, makes it 8× cheaper at density 0.1 and 20× cheaper at 0.05, which would bring the layer at density 0.1 to about 0.15 of its measured energy. We did not

Table 17: Skip-zero back end, supplementary blocks (columns as in Table 2). Top: the measured-cost optimum of Table 16 in three blocks with uniform allocation and a dense reference; its test accuracy (dense masking, 30 epochs, five seeds) is 70.44%. Bottom: the earlier α sweep measured in separate blocks with their own uniform reference, superseded by the joint blocks of Table 2; its first block ran while analysis scripts were running on the same machine (epochs up to $3\times$ slower) and was repeated, and the set-aside runs are in the logs.

Allocation	**Wh / epoch**	**×dense**	ΔE vs. uniform (%)	p_{Holm}
AlexNet, $d = 0.1$, measured-cost optimum (dense: 0.061 Wh/epoch; 3 blocks)				
Uniform (α=0)	0.833 ± 0.005	$13.7\times$	–	–
Measured-cost optimum	0.247 ± 0.017	$4.0\times$	-70.4 ± 1.8	<0.001
AlexNet, $d = 0.1$, α sweep in separate blocks (earlier design) (dense: 0.058 Wh/epoch; 3 blocks)				
Uniform (α=0)	0.842 ± 0.005	$14.5\times$	–	–
EA+ (α=0.25)	0.804 ± 0.004	$13.8\times$	-4.5 ± 0.3	0.002
EA+ (α=0.75)	0.693 ± 0.005	$11.9\times$	-17.7 ± 0.9	0.002

rerun the CSR training experiments with this layout, so all CSR energies in the paper refer to the implementation as released. The additive model uses the per-layer costs of that same implementation and therefore captures the regime, which an analytical score cannot. The practical consequence is that a CSR-trained network is most expensive near the densities most often used in the DST literature, and that the kernels, not the energy score, decide which allocation is cheap on this back end.

Table 18: Energy (mJ per call, NVML) of each operation of the CSR back end, executed exactly as in training (AlexNet, CIFAR shapes, batch 128), by layer density. "SDDMM with $X^\top$ as a view" repeats the weight-gradient product with a non-contiguous operand, as in our first microbenchmark.

		Layer density				
Layer	**Operation**	0.05	0.1	0.15	0.2	0.3
conv2	value gather	1	1	2	2	3
conv2	forward SpMM WX	153	276	255	409	625
conv2	input-gradient SpMM $W^\top \partial Y$	95	170	195	251	432
conv2	copy $X^\top$ (contiguous)	127	135	117	130	134
conv2	weight-gradient SDDMM (as run)	2329	3926	4197	4376	339
conv2	SDDMM with $X^\top$ as a view	115	487	537	843	192
conv2	gradient scatter	1	2	2	2	3
conv2	**whole layer, forward + backward**	3333	4037	4726	5088	1561
conv3	value gather	1	2	2	3	4
conv3	forward SpMM WX	59	121	164	252	306
conv3	input-gradient SpMM $W^\top \partial Y$	41	68	99	124	201
conv3	copy $X^\top$ (contiguous)	32	30	32	31	31
conv3	weight-gradient SDDMM (as run)	792	1681	1772	1507	163
conv3	SDDMM with $X^\top$ as a view	42	156	167	234	108
conv3	gradient scatter	1	2	2	3	3
conv3	**whole layer, forward + backward**	1307	1896	1844	2054	734

D CHANNEL-STRUCTURED BACK END

Execution. `ea_chan.py` keeps, for every convolution and every hidden linear layer, a sorted index set of active output channels (neurons); the input image and the classifier outputs are never pruned. The forward pass walks the network in order and, for each layer, gathers $W[\text{out}][:, \text{in}]$ with `index_select` from the full weight tensor, where `in` is the active set of the previous layer, and calls the ordinary `F.conv2d` or `F.linear`. Batch normalization runs on the gathered parameters and running statistics, which are scattered back after the step. Both models end at a spatial size of 1×1 before the classifier, so the flattened features are exactly the active channels of the last convolution. Gradients flow through the gather to the full tensors, and the optimizer (SGD with Nesterov momentum and weight decay) updates the full tensors; inactive slices receive zero gradient. The per-layer channel counts are constant between topology updates, so cuDNN's algorithm selection is cached. On a test batch, the output of the structured forward pass and that of the dense network with the corresponding channel masks applied differ by at most $2.4 \cdot 10^{-7}$.

Table 19: Skip-zero back end, all blocks at $d = 0.1$ and 0.01 (Section 6.2): GPU energy per training epoch with every Conv and FC layer executed through cuSPARSE CSR products in the forward and both backward passes, random regrowth. $\times$dense: ratio to the dense run of the same block; ΔE: paired difference to uniform allocation in the same block (mean $\pm$ std over blocks); p_{Holm}: paired t-test, Holm-corrected within each group ("–": fewer than three blocks). The α sweep was rerun in 3 joint blocks that contain all five values of α and a dense reference in random order; the earlier split-block sweep and its repeated block are in Appendix C. One VGG-16 uniform run took 1030 s instead of about 350 s at the same power and was repeated (355 s); both runs are in the logs. Accuracy does not depend on the back end (Table 12).

Allocation	**Wh / epoch**	**$\times$dense**	ΔE vs. uniform (%)	p_{Holm}
AlexNet, $d = 0.1$ (dense: 0.058 Wh/epoch; 3 blocks)				
Uniform (α=0)	0.840 ± 0.005	14.5×	–	–
EA+ (α=0.5)	0.747 ± 0.002	12.8×	-11.1 ± 0.6	0.005
EA+ (α=1)	0.610 ± 0.020	10.5×	-27.4 ± 2.0	0.005
FLOP-share (α=0.5)	0.783 ± 0.009	13.5×	-6.8 ± 1.3	0.013
ERK	0.624 ± 0.016	10.7×	-25.7 ± 1.6	0.005
AlexNet, $d = 0.1$, α *sweep in joint blocks* (dense: 0.060 Wh/epoch; 3 blocks)				
Uniform (α=0)	0.840 ± 0.002	14.0×	–	–
EA+ (α=0.25)	0.798 ± 0.005	13.3×	-5.0 ± 0.5	0.006
EA+ (α=0.5)	0.744 ± 0.007	12.4×	-11.3 ± 0.7	0.003
EA+ (α=0.75)	0.684 ± 0.016	11.4×	-18.5 ± 1.8	0.006
EA+ (α=1)	0.615 ± 0.001	10.3×	-26.7 ± 0.1	<0.001
AlexNet, $d = 0.01$ (dense: 0.058 Wh/epoch; 3 blocks)				
Uniform (α=0)	0.246 ± 0.022	4.2×	–	–
EA+ (α=0.5)	0.242 ± 0.020	4.1×	-1.7 ± 3.7	1.000
EA+ (α=1)	0.252 ± 0.014	4.3×	$+2.4 \pm 3.9$	1.000
FLOP-share (α=0.5)	0.245 ± 0.029	4.2×	-0.8 ± 3.9	1.000
ERK	0.242 ± 0.004	4.1×	-1.2 ± 7.6	1.000
VGG-16, $d = 0.1$ (dense: 0.296 Wh/epoch; 3 blocks)				
Uniform (α=0)	4.245 ± 0.012	14.4×	–	–
EA+ (α=0.5)	4.217 ± 0.003	14.3×	-0.7 ± 0.3	0.160
EA+ (α=1)	4.170 ± 0.027	14.1×	-1.8 ± 0.6	0.114
FLOP-share (α=0.5)	4.236 ± 0.028	14.3×	-0.2 ± 0.9	0.705
ERK	4.519 ± 0.047	15.3×	$+6.4 \pm 0.9$	0.026

Channel-level DST. Every 100 iterations, each layer drops the $\lceil \zeta_t \, k \rceil$ active channels with the smallest weight norm (restricted to active inputs), where k is its number of active channels and ζ_t decays from 0.1 to 0 with a cosine schedule, and activates the same number of inactive channels chosen uniformly at random. A regrown channel receives Kaiming-normal incoming weights, a zero bias, reset batch-normalization statistics and affine parameters, and zero outgoing weights in the next layer, so the network function is unchanged at the moment of regrowth; its momentum buffers are reset. Because regrowth is per-layer conserving, the allocation stays at its target throughout training (Proposition 6.1(c)).

Allocation. Each layer with prunable outputs receives a keep ratio $k_l \in [0.02, 1]$; the weight density of layer l is $k_l k_{l-1}$ with $k_0 = 1$ for the input and $k_L = 1$ for the classifier. Uniform allocation uses one k for all layers; EA+, FLOP-share, EPW, and reverse EPW apply their formulas (Section 3, Eq. 6) to the keep ratios, and the scale is found by bisection so that the total number of kept weights is $d \sum_l N^l$. Table 20 lists the resulting keep ratios at $d = 0.1$.

Pilot. The value $\nu = 0.25$ for EPW was chosen from a one-seed pilot at $\nu \in \{0.25, 0.5\}$ (seed 101), because $\nu = 0.5$ cost 12 points on VGG-16. Reverse EPW ($\nu = -0.25$) was added after the pilot showed that EPW lies below the uniform curve. The pilot runs are released and are not used in any comparison.

E Method Details

Initialization variants (ablations only). *EA*: $S^l = s^l$ ($\alpha = 1$), optionally smoothed by an entropy term $\lambda \log s^l$ ($\lambda = 0.5$). *EA++*: EA+ applied only to layers whose score share exceeds $\tau = 10^{-4}$; layers below τ (typically small FC layers) are kept dense at initialization. *EA-uniform*: $d^l \propto d\,(1 + s^l - 1/L)$ rescaled to the global density. *EA-uniform+*: layers other than the classifier receive half the density of the classifier before global rescaling. All variants use the parameter-weighted global constraint $\sum_l N^l d^l = d \sum_l N^l$.

Table 20: Channel keep ratios k_l of the channel-structured allocations at $d = 0.1$ (hidden layers in forward order; the classifier outputs are never pruned) and the fraction of the dense FLOPs they keep.

AlexNet (layers in order; last column: FLOPs kept) Allocation	1	2	3	4	5	6	7	FLOPs
Uniform	0.31	0.31	0.31	0.31	0.31	0.31	0.31	0.120
EA+ (α=1)	0.36	0.11	0.30	0.35	0.37	0.37	0.37	0.082
EPW (ν=0.25)	0.18	0.24	0.29	0.34	0.36	0.36	0.36	0.071
Reverse EPW (ν=−0.25)	0.51	0.39	0.31	0.27	0.25	0.25	0.25	0.196

VGG-16 (layers in order; last column: FLOPs kept) Allocation	1	2	3	4	5	6	7	8	9	10	11	12	13	FLOPs
Uniform	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.101
EA+ (α=1)	0.35	0.31	0.34	0.31	0.34	0.30	0.30	0.33	0.29	0.29	0.34	0.34	0.34	0.100
EPW (ν=0.25)	0.11	0.12	0.17	0.17	0.24	0.24	0.24	0.31	0.31	0.31	0.36	0.36	0.36	0.061
Reverse EPW (ν=−0.25)	0.84	0.78	0.55	0.55	0.40	0.40	0.40	0.31	0.31	0.31	0.26	0.26	0.26	0.235

Table 21: Channel-structured back end on CIFAR-10: whole-training GPU energy, energy relative to the dense run of the same block (mean ± std over blocks), test accuracy, and the equal-energy gain G against the uniform rows (as in Table 2; †: cheaper than every uniform row, compared with the cheapest). Three blocks per model (seeds 17–19), each with all nine configurations in random order; the short schedules (AlexNet 8 epochs, VGG-16 5 epochs) fit nine configurations per block into our compute budget.

Allocation	d	**Wh**	**Energy** (×dense)	**Accuracy (%)**	G (pts)
AlexNet, 8 epochs (dense: 0.49 Wh, 73.26 ± 0.90%)					
Uniform	0.05	0.19	0.40 ± 0.01	67.26 ± 0.64	–
Uniform	0.1	0.22	0.46 ± 0.02	69.80 ± 0.62	–
Uniform	0.2	0.30	0.62 ± 0.01	71.25 ± 0.71	–
Uniform	0.3	0.36	0.75 ± 0.01	71.98 ± 0.96	–
EA+ (α=1)	0.1	0.22	0.45 ± 0.01	68.50 ± 0.97	−0.83 [−2.1, +0.3]
EPW (ν=0.25)	0.1	0.21	0.43 ± 0.02	68.44 ± 0.90	−0.14 [−1.5, +1.0]
Reverse EPW (ν=−0.25)	0.05	0.23	0.47 ± 0.01	68.13 ± 0.90	−1.73 [−2.6, −0.6]
Reverse EPW (ν=−0.25)	0.1	0.27	0.55 ± 0.00	70.81 ± 0.56	+0.12 [−0.6, +0.9]
VGG-16, 5 epochs (dense: 1.44 Wh, 82.19 ± 0.30%)					
Uniform	0.05	0.44	0.30 ± 0.01	72.94 ± 0.88	–
Uniform	0.1	0.58	0.40 ± 0.01	76.38 ± 0.46	–
Uniform	0.2	0.78	0.54 ± 0.01	78.31 ± 0.09	–
Uniform	0.3	0.95	0.66 ± 0.02	79.74 ± 0.32	–
EA+ (α=1)	0.1	0.59	0.41 ± 0.00	76.16 ± 0.91	−0.37 [−1.4, +0.5]
EPW (ν=0.25)	0.1	0.43	0.30 ± 0.00	72.31 ± 1.27	−0.63 [−2.0, +0.7]†
Reverse EPW (ν=−0.25)	0.05	0.67	0.46 ± 0.01	75.30 ± 0.62	−2.01 [−2.6, −1.4]
Reverse EPW (ν=−0.25)	0.1	0.93	0.64 ± 0.01	77.74 ± 0.57	−1.82 [−2.4, −1.2]

Score implementation (reference implementation). The score of Eq. 2 is computed from forward hooks on every sparsified Conv and Linear layer during one forward pass of a $2 \times 3 \times 32 \times 32$ zero tensor: output spatial size, groups, kernel size, and the number of tokens are read from the actual tensors. For ViT, the attention output projection, the two MLP layers, the patch embedding, and the classifier are scored; the fused query/key/value input projection of `nn.MultiheadAttention` is not a separate module in the implementation and is neither sparsified nor scored (it holds about a quarter of ViT's weights, so ViT's global density refers to the sparsified layers only). The *earlier* implementation used for the benchmark computed the score from hand-written layer lists at ImageNet reference resolutions (227×227 for AlexNet, 224×224 for VGG-16 and ConvNeXt), did not model strides or groups, and multiplied the score by layer-type weights (Section 3).

Algorithm 1. EA-DST (reference implementation): initialization and one topology update.

Input: layers $l = 1..L$ with N^l weights, global density d, (e^{op}, e^{mov}), α, β, γ, drop fraction ζ_t (cosine-decayed from 0.5 to 0.005), update period $\Delta T = 100$.

Initialization.

1. Compute S^l (Eq. 2) from the layer shapes observed in one forward pass; set $s^l = S^l / \sum_i S^i$.
2. $S^l_\alpha \leftarrow \alpha s^l + (1-\alpha)/L$; find c by bisection such that $\sum_l N^l \min(1, c\, d\, (1 - S^l_\alpha)) = d \sum_l N^l$; set $d^l \leftarrow \min(1, c\, d\, (1 - S^l_\alpha))$, clamp sparsity at 0.99.
3. In each layer, activate exactly $\mathrm{round}(d^l N^l)$ connections chosen uniformly at random.

After every optimizer step: multiply each weight tensor by its mask.

Every ΔT iterations (topology update).

1. For each layer, remove the $\lceil \zeta_t \cdot \mathrm{nnz}^l \rceil$ active weights with the smallest magnitude; let N^l_{grow} be the number removed and $K = \sum_l N^l_{\text{grow}}$.
2. EA-random: compute r^l and $N^{l,\text{new}}_{\text{grow}} = K r^l / \sum_i r^i$ (Eq. 4); round, cap at the number of inactive weights, and assign the remainder to the layers with the largest r^l, so that $\sum_l N^{l,\text{new}}_{\text{grow}} = K$. Random regrowth: $N^{l,\text{new}}_{\text{grow}} = N^l_{\text{grow}}$.
3. In each layer, activate $N^{l,\text{new}}_{\text{grow}}$ inactive connections uniformly at random, with weight zero.
4. Log the global and per-layer density.

F Energy Proxy: Metrics, Robustness, and Design Choices

F.1 Robustness of the Score and of the EA+ Allocation

Table 22 summarizes the score from one forward pass at the CIFAR training resolution. The ranking is insensitive to the coefficients: changing e^{mov}/e^{op} by three orders of magnitude keeps Spearman $\rho \geq 0.76$, and the hand-set layer-type weights of an earlier implementation change the ranking only on AlexNet ($\rho = 0.67$). The score is strongly but not perfectly correlated with FLOPs ($\rho = 0.73$–0.98), which is why FLOP-share allocation is our control. EA+ is a mild perturbation of uniform allocation: at $\alpha = 0.5$ the mean deviation of the layer densities from d is 0.0007–0.0098, one to two orders of magnitude below ERK, and the largest-to-smallest layer density ratio is at most 1.59. The last rows give the energy kept by EA+ in the sense of Proposition 4.1, that is, $r \cdot d$.

Table 22: Offline analysis of the score (Eq. 2; training resolution, groups modelled) and of the EA+ allocation at $d = 0.1$ on CIFAR (32×32), computed from one forward pass (no training). ρ: Spearman rank correlation of layer scores; spread: largest over smallest layer density; dev: mean $|d^l - d|$; energy kept: $\sum_l d^l E^l_{\text{step}} / \sum_l E^l_{\text{step}}$ (uniform allocation keeps 0.100). Last row: Spearman ρ between measured per-layer energy and the *earlier* score implementation (Appendix H).

	AlexNet	**VGG-16**	**ResNet-34**	**ConvNeXt**	**MobileNetV2**	**ViT**
Layers L / top-3 score share	8 / 0.96	14 / 0.48	37 / 0.14	59 / 0.22	53 / 0.45	38 / 0.12
ρ(score, FLOPs) / ρ(score, energy only)	0.98 / 0.93	0.96 / 0.88	0.73 / 1.00	0.87 / 0.95	0.96 / 0.98	0.89 / 1.00
ρ(score; e^{mov}/e^{op}=1) / (= 1000)	1.00 / 0.93	1.00 / 0.76	1.00 / 0.97	1.00 / 1.00	0.99 / 0.97	1.00 / 1.00
ρ(score, score with ω_{FC}=100)	0.67	1.00	1.00	0.99	0.99	1.00
Spread: EA+ (α=0.5) / ERK	1.59 / 20.3	1.10 / 12.4	1.02 / 17.2	1.04 / 27.8	1.13 / 31.5	1.02 / 12.7
Dev. from uniform: EA+ / ERK	0.0098 / 0.206	0.0032 / 0.241	0.0007 / 0.224	0.0018 / 0.306	0.0034 / 0.368	0.0008 / 0.044
Est. energy kept: EA+ α=0.5 / α=1	0.091 / 0.083	0.099 / 0.098	0.100 / 0.101	0.100 / 0.101	0.101 / 0.102	0.100 / 0.100
Measured vs. earlier score, ρ	0.98	0.97	0.92	0.95	0.91	0.88

F.2 Proxy-Accuracy Metrics

We compare the proxy with per-layer energy measured by CodeCarbon using three metrics. **Rank mismatches.** Sort the layers by energy for both the proxy and the measurement and count the layers whose ranks differ,

$$\text{Mismatch} = \sum_{\ell=1}^{L} \mathbb{I}\big[r^{\text{est}}_\ell \neq r^{\text{meas}}_\ell\big], \tag{8}$$

where $r^{(\cdot)}_\ell$ is the rank of layer ℓ (lower is better, ideally 0). **Percentage error.** Rank mismatches do not measure the size of the error. Because EA-DST needs the *relative* trend across layers, we

normalize by the layer with the highest measured energy, $\ell^\star = \arg\max_\ell E_\ell^{\text{meas}}$, and compute for each layer the reduction ratio $R_\ell^{(\cdot)} = (E_{\ell^\star}^{(\cdot)} - E_\ell^{(\cdot)})/E_{\ell^\star}^{(\cdot)}$ for $(\cdot) \in \{\text{est}, \text{meas}\}$. The reported %Err is the mean over layers of $|R_\ell^{\text{est}} - R_\ell^{\text{meas}}|/R_\ell^{\text{meas}} \times 100\%$. **Spearman correlation** ρ is defined in Appendix H.

F.3 Invariances of the Score Shares

Let $E_{\text{true}}^l = \kappa S^l + u^l$ with $\kappa > 0$. For any two layers i, j, $E_{\text{true}}^i - E_{\text{true}}^j = \kappa(S^i - S^j) + (u^i - u^j)$, so the order of i and j can differ from the order of their scores only if $|u^i - u^j| \geq \kappa|S^i - S^j|$. This is an elementary observation: it does not bound u^l, which is why we measure the ranking (Section H). If $u^l = u_0$ for all layers and $w = \kappa S^{\text{tot}}/(\kappa S^{\text{tot}} + Lu_0) \in (0, 1]$ with $S^{\text{tot}} = \sum_i S^i$, then

$$\frac{E_{\text{true}}^l}{\sum_i E_{\text{true}}^i} = \frac{\kappa S^l + u_0}{\kappa S^{\text{tot}} + Lu_0} = w\, s^l + (1 - w)\, \frac{1}{L},$$

and substituting into Eq. 3 gives $S_{\alpha,\text{true}}^l = \alpha w\, s^l + (1 - \alpha w)/L$, i.e., EA+ on the score with α replaced by $\alpha w \leq \alpha$. Effects that scale all layers proportionally are absorbed by κ and cancel exactly; layer-uniform additive effects move the allocation toward uniform; only the layer-specific remainder can reorder layers.

F.4 Backward-Pass Cost

For a layer $Y = XW$, the backward pass computes $\partial\mathcal{L}/\partial W = X^\top(\partial\mathcal{L}/\partial Y)$ and $\partial\mathcal{L}/\partial X = (\partial\mathcal{L}/\partial Y)W^\top$, each with the same number of multiply–accumulates as the forward pass, so the training cost is about $3\times$ the forward cost (Griewank & Walther, 2008); cached activations add memory traffic. Table 23 reports measured backward-to-forward energy ratios per training iteration on the GPU: $[2.10, 2.70]$ across Conv, FC, and self-attention layers at densities 1.0 and 0.1. Because the score uses only shares, a backward multiplier that is common to all layers cancels and is therefore not part of Eq. 2; the measured ratios vary by at most 0.3 between layer types, so a layer-specific multiplier would change the shares by at most about 20% relative, within the range explored by α.

Table 23: Measured backward-to-forward energy ratios per training iteration (GPU).

Layer type	**Density**	$E_{\text{bwd}}/E_{\text{fwd}}$	$E_{\text{total}}/E_{\text{fwd}}$
ResNet Conv 3×3	1.0	2.10	3.10
ResNet Conv 3×3	0.1	2.41	3.41
VGG FC	1.0	2.40	3.40
VGG FC	0.1	2.70	3.70
ViT self-attention	1.0	2.40	3.40
ViT self-attention	0.1	2.70	3.70

F.5 Hardware Coefficients

The coefficients e^{op} and e^{mov} are taken from the literature and differ across hardware generations. Because EA-DST uses only shares, a common rescaling of both coefficients or of the throughput Ψ leaves the allocation unchanged; only the ratio e^{mov}/e^{op} (here 10 for both CPU and GPU) and the layer-type weights ω^l matter. Moving data costs at least an order of magnitude more energy than an arithmetic operation (Horowitz, 2014; Delestrac et al., 2024). Table 22 shows that the ranking is insensitive to this ratio: varying it from 1 to 1000 keeps Spearman $\rho \geq 0.96$ with the default, and the CPU and GPU scores rank layers identically. Both the coefficient values and the layer-type weights are therefore unlikely to matter for the ranking, but they are not calibrated for any processor.

F.6 Allocation Safeguards

With EA+, $S_\alpha^l \leq \alpha + (1 - \alpha)/L < 1$ for $L > 1$, so every layer keeps a positive density $d^l = c\, d\,(1 - S_\alpha^l)$ and no layer can be disconnected. Densities that would exceed 1 after the global rescaling are saturated, and layer sparsity is clamped at $s_{\max} = 0.99$. The threshold τ of EA++ keeps layers with a negligible score share dense. α interpolates between uniform allocation and pure score-driven allocation; Table 22 shows that even $\alpha = 1$ perturbs uniform densities only moderately (spread 1.07–2.33).

F.7 Energy and Layer Importance

EA-DST does not assume that high-energy layers are less important. Energy decides where sparsity is *cheap*, while magnitude-based removal and gradient-guided regrowth (the γ term in Eq. 4) decide which connections are *useful*. We also explored allocating sparsity from explicit layer-importance scores, but computing such scores requires additional training passes whose energy offset the savings, so we leave energy-free importance estimates to future work. Unlike the Lottery Ticket Hypothesis (Frankle & Carbin, 2019), which seeks a fixed, accuracy-preserving subnetwork of a dense model, EA-DST never trains a dense model and keeps changing the mask during training, so conclusions about fixed winning tickets do not directly apply.

G A Stylized Allocation Model (Motivation Only)

This section gives the idealized model that motivates allocating sparsity by energy. It is *not* an analysis of EA-DST: it assumes energy linear in density, ignores accuracy, and treats masks as static, whereas Section 5 shows that energy is not linear in density on standard hardware and EA-DST adapts masks during training.

Model. Let layer l have N^l weights, density $d^l \in [d_{\min}, 1]$, and expected energy per step $a^l d^l$, where a^l is the energy of the dense layer. Under the parameter-weighted budget,

$$\min_{\mathbf{d}} \sum_l a^l d^l \quad \text{s.t.} \quad \sum_l N^l d^l = d \sum_l N^l, \qquad d_{\min} \le d^l \le 1. \tag{9}$$

This is a fractional knapsack problem: its solution sets $d^l = d_{\min}$ for the layers with the largest *energy per weight* a^l/N^l, in that order, until the budget is met; one layer may be fractional. Two consequences are relevant. First, the relevant ordering is by a^l/N^l, not by a^l; EA+ ranks by energy share and smooths the allocation with α and the global rescaling, so it is a heuristic and not the optimizer of Eq. 9. Second, the unconstrained optimum is extreme (layers at $d_{\min}$), which is precisely why the energy objective alone is not a useful training objective: without an accuracy term the model prefers to disconnect expensive layers. EA-DST addresses this with the mixing weight α, the clamp $s_{\max} = 0.99$, and regrowth.

H Rank Agreement of the Score with Measurements

EA-DST needs the *ordering* of layers, so we compare the score's ranking with per-layer energies measured on the GPU with CodeCarbon, using three metrics. *Rank mismatches*: the number of layers whose rank differs. *Percentage error*: normalizing by the layer $\ell^\star$ with the highest measured energy, $R_\ell = (E_{\ell^\star} - E_\ell)/E_{\ell^\star}$ for both estimates, the mean of $|R_\ell^{\text{score}} - R_\ell^{\text{meas}}|/R_\ell^{\text{meas}}$ over layers $\ell \neq \ell^\star$. *Spearman* $\rho = 1 - 6\sum_i \delta_i^2/(L(L^2-1))$ with δ_i the rank difference of layer i.

Table 24: Score vs. measured layer energy: rank mismatches, mean percentage error of the normalized energies, and Spearman ρ.

	AlexNet	**VGG-16**	**ResNet-34**	**ConvNeXt**	**MobileNetV2**	**ViT**
Mismatches	0	0	4	0	4	8
%Err	5.33	4.65	7.01	8.22	8.75	11.21
Spearman ρ	0.98	0.97	0.92	0.95	0.91	0.88

Mismatches occur in architectures with many similarly sized layers (ResNet-34, MobileNetV2, ViT). A device-level meter such as CodeCarbon samples power at a coarse interval, so attributing energy to individual layers is itself approximate: it ignores inter-layer effects such as caching and kernel fusion, and its resolution limits the comparison of layers with similar energy. The rank agreement should therefore be read as a consistency check rather than a precise validation.

I Configuration Setup

Table 26: Models, datasets, and training settings shared by all methods.

Model	Data	LR	Momentum	Train BS	Test BS	Epochs
AlexNet	CIFAR-10/100, Tiny-IN	0.1	0.9	128	100	100
VGG-16	CIFAR-10/100, Tiny-IN	0.1	0.9	128	100	40/100
ResNet-34	CIFAR-10/100, Tiny-IN	0.1	0.9	128	100	20
ConvNeXt	CIFAR-10/100, Tiny-IN	0.1/0.01	0.9	128	100	60
MobileNetV2	CIFAR-10/100, Tiny-IN	0.1/0.01	0.9	128	100	50
ViT-B/16	CIFAR-10/100, Tiny-IN	0.1/0.01	0.9	128	100	100

Table 25: Default hyperparameter configurations.

Configuration	Value
batch_size	128
momentum	0.9
seed	17 (base; 3 runs)
optimizer	SGD (Nesterov)
lr_schedule	$\times 0.1$ at 50%, 75%
decay_frequency	25000
L2	0.0005
sparse	True
death	magnitude
death_rate	0.5
update_frequency	100
death_rate_decay	cosine
mest_mu_init	0.02
mest_decay_milestones	0.6, 0.85

Training protocol. All models are trained from scratch with SGD (Nesterov momentum 0.9, weight decay $5{\cdot}10^{-4}$, batch size 128); the learning rate (Table 26) is divided by 10 at 50% and 75% of the epochs. 10% of the training set is held out for checkpoint selection, and the selected checkpoint is evaluated on the test set (for Tiny-ImageNet, on the official validation set); "Eval" accuracy in all tables is this test accuracy, and "Train" accuracy is the running training accuracy. Topology updates occur every 100 iterations with a cosine-decayed drop fraction. Results are averaged over three runs with the base seed 17. **Datasets and models.** CIFAR-10/100 (32×32) and Tiny-ImageNet (200 classes, 100k training images, 64×64) (Le & Yang, 2015). The models are AlexNet (CIFAR-sized variant), VGG-16 with batch normalization, ResNet-34 (BasicBlock, $[3, 4, 6, 3]$), ConvNeXt, MobileNetV2, and ViT-B/16 (patch size 16, 12 layers, width 768). **EA-DST hyperparameters.** Table 27 lists the EA-DST-specific hyperparameters, their defaults, search ranges, and selection criteria. For the benchmark they were selected by grid search on a held-out 20% split of the CIFAR-10 training set and then kept fixed across models, datasets, densities, and processors; baselines use the hyperparameters of their reference implementations with the shared training protocol above and were not tuned further, so the tuning budgets of the benchmark are not equal, and the energy of that search was not recorded. The controlled study (Section 5) uses the same values ($\alpha = \beta = \gamma = 0.5$) with *no* tuning of any method, so its comparisons have equal (zero) tuning budgets.

Table 27: EA-DST hyperparameters: defaults, grid-search ranges, and roles.

Symbol	Default	Range	Selection criterion	Role
α	0.5	$[0, 1]$	Maximize validation accuracy under the energy constraint	Balances energy-driven ($\alpha = 1$) and uniform ($\alpha = 0$) allocation (Eq. 3)
τ	10^{-4}	$[10^{-5}, 10^{-2}]$	Avoid collapsing negligible-energy layers	EA++ threshold on the energy share below which a layer is not sparsified
λ	0.5	$[0.1, 2]$	Smooth layer-density distribution	Entropy regularization preventing over-concentration of sparsity
β	0.5	$[0, 1]$	Energy–accuracy trade-off in regrowth	Weight of the energy term in Eq. 4
γ	0.5	$[0, 1]$	Preserve gradient information	Weight of the gradient term in Eq. 4
ε	–	–	Match the target density	Overall scaling in EA-uniform

The experimental configuration, including all hyperparameters, is provided in Tables 26 and 25. The benchmark experiments were conducted on Ubuntu 22 to limit background activity. All non-essential services were disabled during experimentation.

J Energy Measurement

What is measured. We use CodeCarbon v2.4.1 (Courty et al., 2024) with its default settings. The tracker is started and stopped around each training epoch, so the reported energy covers the training steps including topology updates and mask handling, and excludes evaluation passes. CPU and GPU energy are read from RAPL and NVML where available (otherwise CodeCarbon falls back to a TDP-based estimate), and RAM energy is *modelled* as a constant power per GB of memory multiplied by runtime; it therefore tracks runtime rather than memory traffic. The reported energy thus includes both dynamic and static power of these components. Static and background power are not identical across methods, because static energy grows with runtime; differences between methods therefore reflect both their sparse computation and their runtime. We use CodeCarbon only to *evaluate* methods; EA-DST relies solely on the score. No external power-meter validation was performed, and the run-to-run variation of the measurement was not characterized separately; we therefore treat differences below 1% as ties (Appendix A).

Controlled study (Section 5). The controlled study reads the cumulative energy counter of the GPU through NVML (`nvmlDeviceGetTotalEnergyConsumption`, millijoule resolution) immediately before and after each training epoch, after synchronizing the device; evaluation passes are excluded. It therefore measures GPU energy only, but with a resolution far below the differences we interpret, and it is unaffected by CPU-side activity. Runs are executed strictly one at a time.

Carbon. Carbon emissions are energy multiplied by the grid carbon intensity of the location and time of the run; for comparing methods trained in one location they carry no information beyond energy, so we do not report them.

K Sensitivity and Ablation: Full Results

Tables 28 and 29 report the sensitivity sweeps at density 0.1 (no spread reported). **Provenance.** Rows marked § are the same runs as in the main result tables ($\alpha = 0$: uniform/random; $\alpha = 0.5$: EA+/random = EA-DST-I; $\beta = \gamma = 0$: EA+/random; $\beta = \gamma = 0.5$: EA+/EArandom = EA-DST); the other rows are separate runs that do not appear elsewhere. Because $\alpha = 0$ is uniform allocation, the ViT (CIFAR-10) $\alpha = 0$ row is the uniform/random run of Supplementary Table S1.

Initialization. On ResNet-34 (CIFAR-100), test accuracy is flat for $\alpha \in [0, 0.5]$ and decreases for $\alpha \geq 0.75$, while energy decreases monotonically with α by up to 1.6%: stronger score-driven allocation saves some energy but eventually over-prunes. On ViT, $\alpha = 0.5$ improves test accuracy over uniform allocation by 6.2 points (CIFAR-10) and 1.7 points (Tiny-ImageNet), and $\alpha = 1$ gives the lowest energy. The threshold τ keeps negligible-score layers dense, which here improves accuracy by 0.6 points at no energy cost.

Regrowth. On VGG-16 (CIFAR-100), the energy term (β) and the gradient term (γ) each improve test accuracy over random regrowth ($+0.65$ and $+0.85$ points) and their combination by $+1.30$ points, at 3.1–4.9$\%$ higher energy; on VGG-16 (Tiny-ImageNet) the joint rule matches random regrowth in test accuracy and energy, and on ConvNeXt (CIFAR-100) it lowers energy by 6.6% at similar test accuracy. These regrowth ablations were run with the original implementation, whose final density was not logged; given Section 6.4, their energy differences cannot be attributed to the regrowth weights.

L Layers' Density

The layer-wise densities for AlexNet, VGG16, and ResNet-34 at model densities of 0.1, 0.5, and 0.9 are shown in Figure 9. In EA++, although certain low-energy layers are excluded from sparsification, their densities still adjust during global sparsity normalization to meet the target overall density (0.1, 0.5, or 0.9). In VGG, SNIP and ERK tend to assign higher density to early convolutional layers due to their respective scoring mechanisms through extra computation, though this effect reduces as overall density increases. Both methods also typically preserve higher density in the FC layer. The fluctuations observed in EA-uniform variants around the uniform baseline arise from incorporating layer-wise energy sensitivity. For ResNet, EA++, ERK, and SNIP yield similar densities in layers 10, 19, and 32, which is notable given that each method defines layer importance differently through

Table 28: Sensitivity of the sparsity-allocation hyperparameters α, τ, and λ (density 0.1).

Model (dataset)	Setting	Train acc. (%)	Eval acc. (%)	Energy (Wh)
ResNet-34 (CIFAR-100)	$\alpha = 0$ (uniform)[§]	80.43	**69.13**	55.18
	$\alpha = 0.25$	80.38	69.12	55.12
	$\alpha = 0.5$ (default)[§]	80.34	69.11	55.08
	$\alpha = 0.75$	79.95	68.60	54.85
	$\alpha = 1$ (pure energy)	78.40	67.10	**54.30**
	$\tau = 0$ (no exclusion)	79.80	68.50	55.10
	$\tau = 10^{-4}$ (default)	80.34	69.11	55.08
	$\lambda = 0.5$ (default)	80.34	69.11	55.08
ViT (CIFAR-10)	$\alpha = 0$ (uniform)[§]	60.33	57.12	116.25
	$\alpha = 0.5$ (default)[§]	**65.04**	**63.32**	116.85
	$\alpha = 1$ (pure energy)	63.80	61.90	**114.50**
ViT (Tiny-ImageNet)	$\alpha = 0$ (uniform)[§]	47.19	28.07	1143.09
	$\alpha = 0.5$ (default)[§]	**48.85**	**29.78**	1124.62
	$\alpha = 1$ (pure energy)	45.20	26.50	**1112.10**

Table 29: Sensitivity of the regrowth weights β (energy) and γ (gradient) (density 0.1).

Model (dataset)	Setting	Train acc. (%)	Eval acc. (%)	Energy (Wh)
VGG-16 (CIFAR-100)	$\beta = 0, \gamma = 0$ (random)[§]	85.77	68.25	**34.05**
	$\beta = 0.5, \gamma = 0$ (energy only)	89.20	68.90	35.10
	$\beta = 0, \gamma = 0.5$ (gradient only)	89.80	69.10	35.50
	$\beta = 0.5, \gamma = 0.5$ (default)[§]	**91.12**	**69.55**	35.72
VGG-16 (Tiny-ImageNet)	$\beta = 0, \gamma = 0$ (random)[§]	**96.51**	52.04	**514.99**
	$\beta = 0.5, \gamma = 0.5$ (default)[§]	94.49	**52.11**	515.87
ConvNeXt (CIFAR-100)	$\beta = 0, \gamma = 0$ (random)	30.45	**31.77**	59.67
	$\beta = 0.5, \gamma = 0.5$ (default)[§]	**30.96**	31.70	**55.74**

energy, parameter count, or sensitivity. In low-density settings, EA+ produces similar sparsity across several layers because these layers contain fewer parameters and thus incur lower computational and data-movement costs. Additionally, since results are expressed as percentages, two layers may share the same density despite having different absolute parameter counts.

Effect of layer density on training performance. We remove the same number of parameters (506k, i.e., 10% of the model, which explains the small energy changes) from each sufficiently large AlexNet layer and measure training energy and test accuracy after 10 epochs (Figure 10, left). Removing the parameters from Conv layers reduces energy more than removing them from FC layers, and sparsifying the final FC layer has almost no energy effect. In this short run, sparsifying Conv3 or Conv4 slightly increased test accuracy; we did not investigate the cause, and single short runs of this kind are within run-to-run variation. The same number of removed parameters therefore yields different energy savings depending on the layer, which motivates allocating sparsity per layer.

From these measurements we draw two observations. First, training energy can diverge from FLOP or parameter counts, so optimizing these counts does not necessarily reduce energy. Second, energy is non-uniform across layers; whether allocating more sparsity to high-energy layers improves the energy–accuracy trade-off is tested in Section 5, where it improves accuracy at nearly equal energy with random regrowth but adds little to strong DST methods.

M Accuracy and Energy Consumption

This section lists the raw per-configuration results from which all statistics in Section 5 and Appendix A are computed: total training energy, training accuracy, and test ("Eval") accuracy for AlexNet, VGG-16, ResNet-34, ConvNeXt, MobileNetV2, and ViT on CIFAR-10/100 and Tiny-ImageNet (Tables 31–32 here, and the complete CIFAR-10/100 tables, including EA+ combined

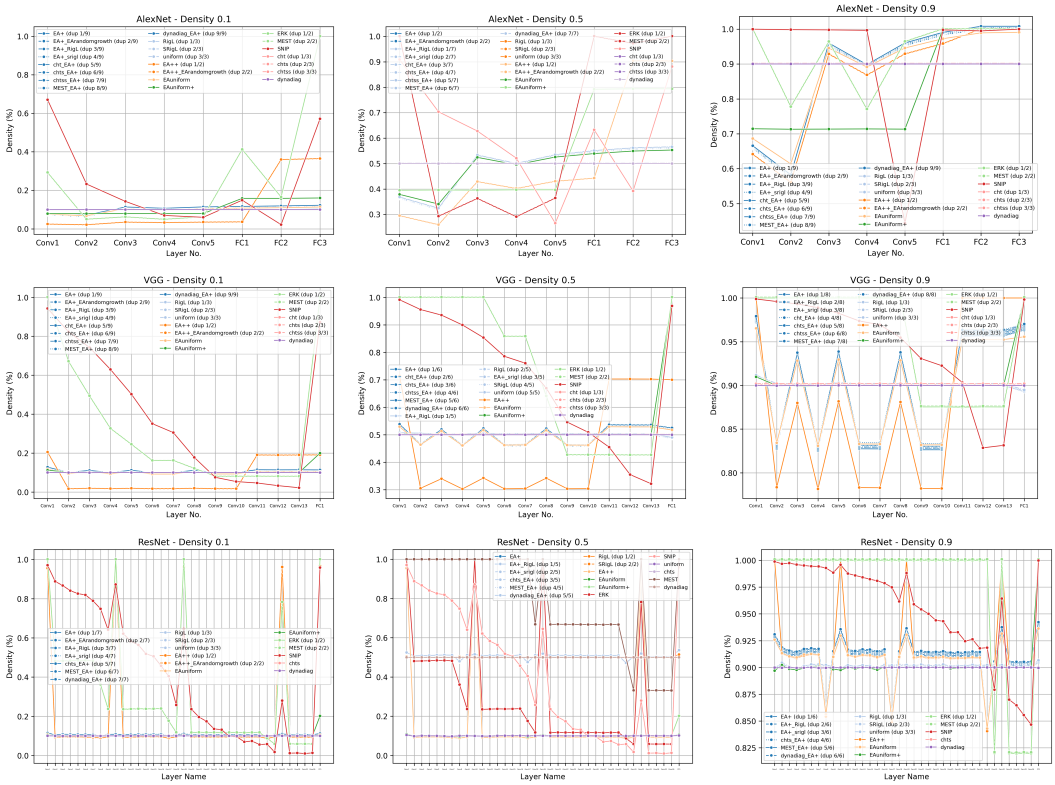

Figure 9: Density of AlexNet, VGG, and ResNet layers after applying various initialization criteria on the CIFAR-10 dataset (duplicate-valued curves are marked for clarity).

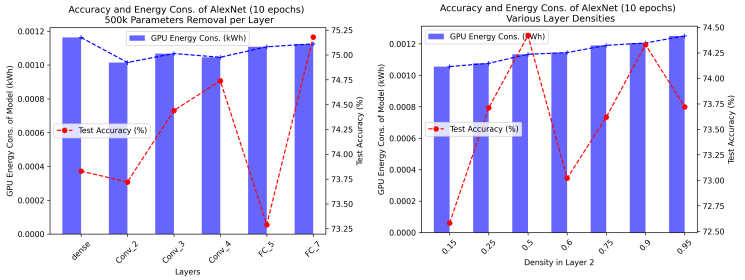

Figure 10: Test accuracy and GPU energy consumption for training AlexNet for 10 epochs on CIFAR-10 when removing 506k parameters from each layer sequentially (total model density fixed at 0.9, left). Test accuracy and GPU energy consumption for varying sparsity levels applied to the second convolutional layer (right).

with each DST method, in Supplementary Tables S1–S5). Highlighted cells mark the best values per column within a block and do not account for measurement noise; differences below about 1% in energy are not meaningful. Training-accuracy curves are shown in Figure 11 (Tiny-ImageNet) and in Supplementary Figures S1–S2 (CIFAR-10/100). All aggregate statistics use the fixed configurations of Appendix A.

CIFAR-10. As model density increases, the accuracy differences across initialization and growth methods diminish, which is expected given the reduced sparsity variation across layers.

SRigL is designed primarily for inference with a constant fan-in constraint, which explains its higher training energy in our setting. On AlexNet, CHTss achieves the highest accuracy at density 0.1 but

also the highest energy among the unstructured methods; EA++ and RigL have low energy and lower accuracy. SNIP is competitive on VGG-16 but fails on ConvNeXt, MobileNetV2, and ViT, and DynaDiag performs well mainly on ViT.

Dense training uses more energy than sparse training at density 0.1 but less than most sparse methods at densities 0.5 and 0.9, because the benchmark code applies masks through sparse tensors after every step and adds the remove–regrow cycle, while all products stay dense (Section 6.1). Energy benefits on this hardware therefore appear only at densities of about 0.1 and below. For VGG-16, ResNet-34, ConvNeXt, and ViT, test accuracy can exceed dense accuracy at density 0.9 and occasionally at 0.1 (ResNet-34, ViT); at the shared, short training budget the dense ResNet-34 and ViT are under-trained, so these gains should not be read as a regularization benefit of sparsity.

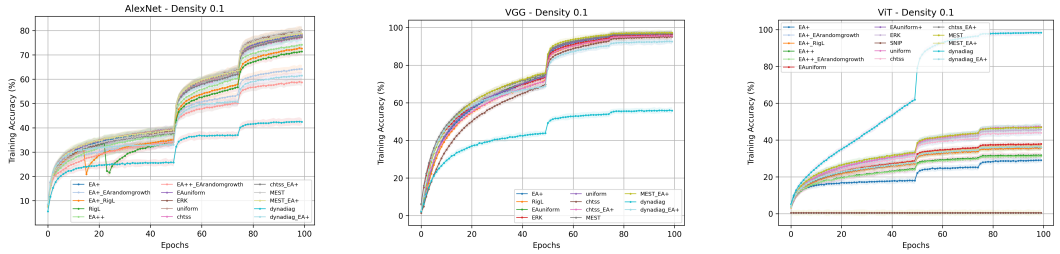

Figure 11: Training accuracies for various models on the Tiny-ImageNet dataset using various techniques at model densities of 0.1.

CIFAR-100. Accuracy is lower than on CIFAR-10 at the same number of epochs, and energy is higher because of the larger output layers. Accuracy differences between methods are more pronounced for deeper networks and at lower densities, where allocations differ most. On ViT, DynaDiag achieves the best accuracy–energy combination.

Tiny-ImageNet. Energy increases with the larger inputs and output layers. For AlexNet and VGG-16, EA-DST-I is among the lowest-energy unstructured methods, and its test accuracy is within 0.4 points (AlexNet) and 3.2 points (VGG-16) of the most accurate method; on ViT it is the most accurate method. DynaDiag has the lowest energy on all three models, with large accuracy losses on AlexNet and VGG-16.

Longer Tiny-ImageNet training. Training-accuracy curves plateau within the epoch budget for VGG-16 and ViT on Tiny-ImageNet (Figure 11). For AlexNet, training accuracy was still increasing at 100 epochs, so we trained for up to 300 epochs: test accuracy increased until about epoch 160 and then decreased due to overfitting. Table 30 reports the 160-epoch results; the four methods lie within 2% in energy and 0.15 points in test accuracy.

Table 30: AlexNet on Tiny-ImageNet trained for 160 epochs at density 0.1.

Method	Energy (Wh)	Train acc. (%)	Eval acc. (%)
EA+	187.41	91.22	46.11
EA++	**186.94**	91.19	46.01
ERK	189.60	**91.28**	45.98
CHTss	190.72	90.34	**46.13**

N Extreme Sparsity

Results at density 0.01 are in Table 32 (all methods, CIFAR-10) and Table 33 (fixed configurations, including AlexNet on CIFAR-100); pairs of one EA variant and one strong baseline for the remaining model–dataset combinations are listed in Supplementary Table S5 for completeness. At this density all methods save 11–15% energy relative to dense training in the median while losing 4–18 points, so the choice of such an extreme density depends entirely on the application.

O Various Processor Architectures

EA-DST-I at density 0.01 was run on four processors: an Intel i7-2600 CPU, an Intel i7-13700H CPU, an NVIDIA RTX A500 laptop GPU, and a cloud NVIDIA A100 GPU (Table 34, Section A.1). CPU-only training uses several times more energy than GPU training, and on the CPU platforms

Table 31: Total energy consumption and training/evaluation accuracy of the AlexNet, VGG16, and ViT models on the Tiny-ImageNet dataset at 0.1 density across various initialization and growth criteria. Top-performing values are highlighted.

Init.	Growth	Model								
		AlexNet			VGG16			ViT		
		Energy (Wh)	Train Acc.	Eval Acc.	Energy (Wh)	Train Acc.	Eval Acc.	Energy (Wh)	Train Acc.	Eval Acc.
Dense		140.39	99.14	46.56	610.49	99.96	57.78	1243.10	92.63	33.05
EA+	random	**117.13**	79.12	45.59	**514.99**	96.51	52.04	1124.62	48.85	29.78
EA+	EArandom	120.07	64.19	45.04	515.87	94.49	52.11	1118.53	35.84	26.38
uniform	random	119.01	77.70	45.81	516.08	96.74	52.32	1143.09	47.19	28.07
EAuniform	random	119.00	78.32	45.91	515.89	96.66	52.80	1135.56	37.85	27.01
EAuniform+	random	117.21	78.19	44.28	516.01	96.37	55.01	1135.88	47.19	28.07
EA++	random	117.91	74.14	42.90	515.21	96.11	54.99	1142.69	31.72	24.97
EA++	EArandom	**116.92**	58.72	43.28	515.99	96.47	52.90	1126.37	31.18	24.78
ERK (SET)	random	118.28	**80.19**	44.84	515.95	96.15	**55.24**	1139.10	45.60	28.43
SNIP	random	118.21	$\leq$10	$\leq$10	519.65	$\leq$10	$\leq$10	1141.32	$\leq$10	$\leq$10
RigL†		117.59	71.37	42.37	517.44	95.53	52.66	1139.65	35.42	26.55
EA+	RigL	117.24	72.60	43.28	517.32	95.61	52.71	1129.35	35.82	26.98
CHTss†		122.77	77.19	**45.99**	523.15	95.07	52.45	1179.91	43.94	28.41
EA+	CHTss	122.59	77.36	45.10	523.01	95.60	53.60	1181.74	36.11	27.70
MEST†		118.49	80.16	44.76	518.75	96.77	55.06	1130.74	46.97	**29.10**
EA+	MEST	118.81	78.73	45.71	518.56	**97.32**	52.57	1131.49	36.65	27.27
DynaDiag		**113.66**	42.44	35.17	499.31	55.83	42.76	**1092.41**	**98.34**	28.79
EA+	DynaDiag	118.01	61.48	43.31	517.02	92.60	50.75	1131.29	36.10	26.82

Notes. † Results are based on running the codes and directly measuring energy consumption, which differs from FLOP-based calculations in references; see Section 5.

Table 32: Total energy consumption and training/evaluation accuracy of the AlexNet, VGG16, and ResNet models on the CIFAR-10 dataset at 0.01 density (extreme sparsity) across various initialization and growth criteria. Top-performing values are highlighted.

Init.	Growth	Model								
		AlexNet			VGG16			ResNet		
		Energy (Wh)	Train Acc.	Eval Acc.	Energy (Wh)	Train Acc.	Eval Acc.	Energy (Wh)	Train Acc.	Eval Acc.
Dense		22.89	96.58	86.34	32.41	98.45	92.10	54.19	91.92	87.22
EA+	random	**18.14**	75.01	74.68	28.26	89.26	86.12	48.14	87.50	86.24
EA+	EArandom	19.14	73.01	71.68	28.26	84.26	83.12	48.35	85.20	84.24
uniform	random	19.43	74.68	73.42	28.26	84.07	82.86	48.35	84.89	82.90
EAuniform	random	19.32	73.04	71.72	28.27	81.56	79.12	48.14	85.02	83.66
EAuniform+	random	19.28	72.12	70.88	28.11	87.11	–	48.14	84.95	83.44
EA++	random	19.01	75.87	74.35	28.07	86.99	84.21	**48.13**	83.31	82.86
EA++	EArandom	18.99	75.77	74.11	28.05	87.01	83.27	48.14	83.82	83.12
ERK (SET)	random	19.23	74.60	72.88	28.37	89.53	87.16	48.21	87.43	84.98
SNIP	random	19.36	$\leq$10	$\leq$10	28.35	$\leq$10	$\leq$10	48.39	$\leq$10	$\leq$10
RigL†		18.96	73.58	72.38	28.06	79.08	77.82	48.19	69.67	68.96
EA+	RigL	**18.80**	70.28	68.24	28.00	73.52	71.66	48.17	74.74	74.16
SRigL†		21.80	69.00	69.14	28.28	85.45	82.78	48.21	84.32	79.12
EA+	SRigL‡	22.13	68.94	68.98	28.17	86.11	83.31	48.19	84.37	81.99
CHT†		18.83	71.87	71.00	28.41	84.98	83.01	48.32	83.22	82.41
EA+	CHT	18.87	72.69	71.84	28.38	85.32	83.47	48.25	83.98	83.00
CHTs†		19.05	76.06	75.04	28.35	86.87	83.11	48.19	85.47	84.22
EA+	CHTs	19.04	75.84	**75.10**	28.22	86.98	83.65	48.21	84.51	83.40
CHTss†		20.44	76.51	75.08	29.07	85.77	84.50	48.27	83.40	82.87
EA+	CHTss	20.24	**76.73**	74.94	29.06	85.58	84.64	48.20	83.89	83.35
MEST†		19.03	69.87	68.64	28.42	**89.83**	**87.80**	48.81	**88.13**	**86.44**
EA+	MEST	18.94	69.57	68.14	28.16	76.54	74.52	48.35	80.92	79.90
DynaDiag		19.19	68.57	68.57	**27.62**	55.52	58.00	**47.09**	67.88	68.76
EA+	DynaDiag	19.01	71.11	69.99	28.21	80.94	79.88	48.54	85.49	84.26

Notes. † Results are based on running the codes and directly measuring energy consumption, which differs from FLOP-based calculations in references; see Section 5. ‡ Method is designed primarily for inference efficiency, not training.

RAM accounts for a large share, partly because CodeCarbon models RAM energy as a constant power per GB and thus in proportion to runtime. Since the score ranks layers identically on CPUs and

Table 33: Extreme sparsity ($d = 0.01$): energy (Wh) / test accuracy (%). Bold: best among sparse methods; "fail": accuracy $\leq 10\%$.

Method	AlexNet C10	VGG-16 C10	ResNet-34 C10	AlexNet C100
Dense	22.89 / 86.34	32.41 / 92.10	54.19 / 87.22	26.76 / 59.69
EA-DST	19.14 / 71.68	28.26 / 83.12	48.35 / 84.24	22.70 / 45.57
EA-DST-I	**18.14** / 74.68	28.26 / 86.12	48.14 / 86.24	22.74 / 45.57
Uniform	19.43 / 73.42	28.26 / 82.86	48.35 / 82.90	22.80 / 46.36
ERK (SET)	19.23 / 72.88	28.37 / 87.16	48.21 / 84.98	22.72 / 47.57
RigL	18.96 / 72.38	28.06 / 77.82	48.19 / 68.96	**22.21** / 43.27
CHTss	20.44 / **75.08**	29.07 / 84.50	48.27 / 82.87	23.97 / **48.41**
MEST	19.03 / 68.64	28.42 / **87.80**	48.81 / **86.44**	22.33 / 41.69
DynaDiag	19.19 / 68.57	**27.62** / 58.00	**47.09** / 68.76	fail

Table 34: Energy (Wh), EA-DST-I, $d = 0.01$, CIFAR-10, on two CPUs and two GPUs (†incl. some CPU energy).

Model	i7-2600		i7-13700H		RTX A500†		A100	
	CPU	RAM	CPU	RAM	GPU	RAM	GPU	RAM
AlexNet	76.54	77.80	86.60	23.69	13.41	5.56	7.41	3.22
VGG-16	135.77	135.69	156.28	42.76	20.90	6.72	11.65	3.78
ResNet-34	189.29	182.79	197.37	54.00	19.11	6.49	10.99	3.69

GPUs (Table 22), these runs test the portability of the pipeline rather than of the allocation; accuracies are not listed because they were not recorded for all platforms. On the A100 cloud instance, co-tenant load was not controlled. A comparison of methods on one processor (AlexNet, CIFAR-10, i7-2600, $d = 0.01$) is given in Table 35: all methods lie within 2.5% of each other.

Table 35: Total energy consumption (Wh) for training the AlexNet model on the CIFAR-10 dataset using various sparsification methods, with 100 epochs on the i7-2600 CPU at density 0.01.

Method	EA+	EA++	uniform	EAuniform	ERK	RigL	CHTss	MEST
Energy (Wh)	154.34	154.10	156.43	155.97	155.70	154.11	157.90	154.13