ON THE STABILITY OF GROWTH IN STRUCTURAL PLASTICITY: FORWARD-ACTIVE YET BACKWARD-STARVED

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard deep-learning pipelines usually choose the network architecture before training and keep it fixed throughout optimization. In contrast, a model can also be adapted by editing its structure during training, for example by pruning existing hidden units or growing new ones; however, growth is not simply the inverse of pruning. Pruning selects among units trained from initialization, whereas growth inserts new capacity into an already specialized optimization trajectory. We isolate this insertion problem and show that newborn units can be forward-active yet backward-starved, receiving substantially weaker gradient signal than incumbent units. This asymmetry is weak in small MLPs but emerges in more challenging convolutional settings, where GROW reaches competitive final architectures despite weaker trajectory-level performance. Extending structural edits to residual networks reveals a strong dependence on edit location: head-localized growth produces concentrated but task-aligned representations and faster bounded label remapping, whereas growth throughout the residual hierarchy yields lower effective feature support, and a lower bounded adaptation endpoint. Interventions targeting optimizer state, insertion, selection, and trainability improve newborn integration, but not necessarily final subnetwork quality. Across continual-learning benchmarks, growth is most competitive when newborn units have sufficient time and trainability to integrate. Therefore, GROW should be evaluated not only by the architecture it ultimately discovers, but by whether late-arriving capacity becomes usable before a new structural change occurs during training.

1 INTRODUCTION

Structural plasticity—the ability to modify network architectures during training—is a natural primitive for automated machine learning under explicit resource constraints such as parameters, FLOPs, or latency. We distinguish this architectural notion from the classical use of *plasticity*, which refers to a network's capacity, from its current state, to improve under a changed learning signal. The two concepts are related but not equivalent: changing a model's structure creates new capacity, but that capacity contributes to plasticity only if it can be effectively integrated and optimized. The two basic operators of structural plasticity are *pruning*, which removes capacity from an overparameterized model (Han et al., 2015b;a), and *growth*, which adds capacity to a compact one (Yang et al., 2021; Wu et al., 2020; Dai et al., 2019; Rusu et al., 2016). Both can be viewed as search operators in architecture space, and dynamic sparse training (DST) shows that structure can be updated online while respecting a fixed parameter budget (Lasby et al., 2023; Mocanu et al., 2018; Bellec et al., 2017; Dettmers & Zettlemoyer, 2019). In practice, however, structural adaptation remains dominated by pruning-based approaches.

Growth and pruning are not optimization-symmetric. Pruning selects among units that have participated in training from initialization, consistent with the lottery-ticket view that dense networks may contain sparse, trainable subnetworks (Frankle & Carbin, 2018; Frankle et al., 2019). Growth adds capacity as needed, making it attractive for adaptive and continual systems; here we study shared-parameter growth, where incumbent and newborn units remain jointly trainable rather than modular or frozen (Miconi, 2016; Yoon et al., 2017; Li et al., 2019; Yang et al., 2021). Because newborn units enter after specialization, they must integrate into a mature optimization trajectory before the next shift; otherwise, repeated expansion can itself destabilize learning (Zhao et al., 2024). Consequently, GROW–PRUNE comparisons can conflate final sparse-architecture quality with the efficiency of the insertion process used to obtain it.

Therefore, we study growth as a *structural plasticity primitive* and frame this process through *insertion stability*: the inability of a model to absorb a structural addition within the available training window.

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.

From this perspective, newly added units face three birth-time disadvantages, depending on the growth mechanism: (i) *function shock*, where insertion perturbs the learned input–output mapping; (ii) optimizer *cold start* when new parameters lack optimizer state; and (iii) *weak learning signal*, where newborn units receive disproportionately little credit relative to incumbent units. Throughout, "units" refers to hidden neurons added or removed at the neuron level, not individual connections; our study therefore concerns unit-level structural edits rather than unstructured synapse-level rewiring (Cheney et al., 2017).

The central claim is not that growth is primarily limited because it fails to discover useful sparse architectures, but that growth is a time-sensitive optimization process whose success depends on whether late-arriving capacity gets incorporated before the next change in the available training signal. This can make the adaptive process less stable and more path-dependent, even when the final retrained mask is competitive with pruning. We isolate this insertion primitive and investigate when GROW and PRUNE differ, asking whether this discrepancy stems from the quality of the final sparse architecture or the adaptive process used to generate it. [1]

- We show that the GROW–PRUNE gap is not monolithic: in small MLPs, growth and pruning produce similarly retrainable masks, while in convolutional feature-learning regimes they can produce similarly strong final subnetworks but differ substantially in how well the model behaves as its architecture is modified (Sec. 4). Then, in residual networks, edit locus further determines future adaptability: head-localized GROW adapts fastest under class-permutation remapping, while broader residual growth reaches a lower bounded endpoint (Sec. 4.3).
- We identify insertion-time optimization disadvantage as a process-level bottleneck for growth, showing that newborn units can be forward-active while remaining backward-starved (Sec. 4.2).
- We use interventions on optimizer state, insertion, selection, and activation-level trainability as probes of this bottleneck (Sec. 5), and show that under continual shift, growth is most effective when new units have time to integrate before the next distributional change (Sec. 6).

2 BACKGROUND & RELATED WORK

Dynamic sparsity and structural operators. Dynamic sparse training (DST) methods maintain a fixed parameter budget while updating sparse connectivity online, combining pruning and regrowth as intertwined structural operators (Mocanu et al., 2018; Bellec et al., 2017; Dettmers & Zettlemoyer, 2019). This line of work views pruning and growth as search moves in architecture space, and highlights that the *allocation rule*—which connections or units receive structure—interacts strongly with learning dynamics (Evci et al., 2020). We do not focus on the allocation rule, but on the stability of the *insertion* event itself.

Insertion-compatible growth. Architectural expansion can disrupt a learned function and introduce optimization asymmetries between incumbent and newborn parameters. Net2Net and Network Morphism widen networks while approximately preserving the current input–output mapping (Chen et al., 2016; Wei et al., 2016), while MorphNet couples structural adaptation to resource constraints (Gordon et al., 2018). Other methods use activations or gradients to select expansion locations (Wu et al., 2019; Evci et al., 2022), or modify the optimization of late-arriving parameters through variance transfer and age-dependent learning rates (Yuan et al., 2023). These approaches target distinct parts of the insertion process. More broadly, warm-up, layer-wise modulation, and adaptive optimizer state all point to the same issue: a newly inserted unit may be disadvantaged not only by where it is placed, but by *when* it enters training (You et al., 2019; Xiong et al., 2020; Mosbach et al., 2020; Kingma & Ba, 2014; Reddi et al., 2019; Zhuang et al., 2020). We study these mechanisms jointly through event-aligned unit diagnostics and targeted interventions.

Plasticity and plasticity loss under continual shift. Two distinct concepts are relevant. Following the continual-learning literature, *plasticity* refers to a network's capacity, from its current state, to continue learning—that is, to reduce loss and acquire new mappings when the learning signal changes (Lyle et al., 2023; Abbas et al., 2023). *Loss of plasticity* refers to the decline of this ability over the course of training (Dohare et al., 2024; Klein et al., 2024), and is different from forgetting, which concerns the deterioration of performance on information that was previously learned (McCloskey & Cohen, 1989; Kirkpatrick et al., 2017). In contrast, *structural plasticity* denotes the capability

[1] Code available at: `https://anonymous.4open.science/r/structural_plasticity-1544`

to modify the network architecture itself during training. Consequently, architectural expansion increases the structural capacity of the model, but it enhances plasticity only if the newly introduced units are functionally integrated into the network and remain effectively utilizable.

Growth in continual learning. Many continual-learning methods rely on architectural expansion to accommodate new tasks, with design choices centered on *when*, *where*, and *what* to grow (Rusu et al., 2016; Fernando et al., 2017; Yoon et al., 2017; Li et al., 2019; Yang et al., 2021). Growth is appealing because it allocates fresh capacity for new information, potentially easing the stability–plasticity dilemma. At the same time, strong isolation-based approaches such as PackNet, Piggyback, and winning-subnetwork methods achieve low forgetting by assigning task-specific subnetworks within shared weights (Mallya & Lazebnik, 2018; Mallya et al., 2018; Kang et al., 2022), often at the cost of task identity, routing, or selection at inference time. Recent work further shows that poorly controlled expansion can itself induce forgetting in task-agnostic settings (Zhao et al., 2024). Our work instead asks whether newly added units can integrate stably within a shared, fully plastic representation, rather than through task-isolated or frozen modules.

3 Experimental Setup

We compare three model families under matched data streams, optimizers, and compactness targets:

- **Dense** uses the full architecture without structural edits.
- **Prune** applies Iterative Magnitude Pruning (IMP) to editable layers and rewinds surviving weights after each pruning event. This canonical lottery-ticket baseline separates the quality of the selected sparse mask from the trajectory used to obtain it (App. A.6.2).
- **Grow** starts from a sparse seed mask and iteratively *activates* dormant units until reaching the target compactness. Candidates are ranked by how frequently their post-activations would exceed a small threshold if unmasked, estimating whether they would participate meaningfully if recruited (App. A.6.1). Growth updates the mask in place without rewind: dormant parameters remain at initialization until activation, newborn units do not inherit optimizer state, and all active parameters remain jointly trainable rather than frozen or task-isolated. Gradient-based recruitment gives similar qualitative conclusions (App. B.2.2).

We begin with a three-layer MLP containing two editable hidden layers and an unmasked classifier, so structural edits change hidden capacity without altering the output mapping. Grow and Prune follow opposite compactness trajectories, with Grow increasing from 5% and Prune decreasing from 100% until reaching retained compactness $c \in \{20, 30, 40, 50\}\%$. During the first $T - 1$ cycles, the method adds or removes units and then trains; the final cycle trains at the target compactness.

Each procedure produces a final binary mask, which we evaluate in two ways:

1. **In-place structural-edit**: performance measured during the structural-editing trajectory itself, measured at the end of each cycle. It captures how well the model performs while architecture changes are being made online during training.
2. **Winning-Ticket (WT)**: performance measured after freezing the final discovered mask, rewinding the model, and retraining it from scratch (Frankle & Carbin, 2018). This tests the sparse architecture independently of the path used to discover it.

For both evaluations, we report final cumulative accuracy after the last task (**ACC**) reflecting end-of-stream performance.

4 Feature Learning Exposes a Newborn-Integration Bottleneck

Minimal MLPs do not expose a stable growth gap. On IID MNIST (Lecun et al., 1998), all methods achieve similarly high accuracy. Under class-incremental Split-MNIST with a shared 10-way head, all methods collapse near chance without stabilization, indicating that forgetting dominates the architectural comparison. Therefore, we add a small experience replay buffer (+ER; 50 samples per class) to make structural differences interpretable, and use the SGD optimizer with learning rate $\eta = 0.01$. Across compactness levels, Grow, Prune, and Dense remain closely matched in both In-place structural-edit and Winning-Ticket (WT) metrics, with no consistent ordering; the same holds for Split-Fashion-MNIST (Xiao et al., 2017). In particular, Grow and Prune produce final masks that retrain to similar winning-ticket accuracy and TAA. Thus, neither sparsity nor class-incremental training alone produces a stable Grow–Prune asymmetry. Complete results are reported in Table B.1.

4.1 NEWBORN UNITS ARE FORWARD-ACTIVE BUT BACKWARD-STARVED

We next move to CIFAR-100 (Krizhevsky, 2009) to increase representation-learning and optimization difficulty while retaining the same unit-level structural abstraction. We use a hybrid ConvNet with a dense two-layer convolutional trunk and a four-layer fully connected head. Only the hidden layers of the head are editable; the convolutional trunk remains dense. Models use SGD with $\eta = 0.1$, selected independently for each treatment, and the neutral allocation schedule described in App. B.3.

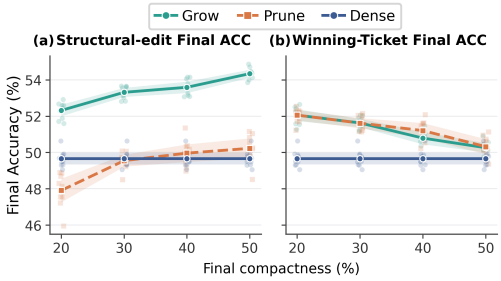

Figure 1: **GROW reaches a substantially stronger structural-edit endpoint despite discovering no better final architecture.** **(a)** During structural adaptation, GROW consistently exceeds PRUNE, reaching 54.3% accuracy at 50% compactness versus 50.2% for PRUNE and 49.7% for DENSE. **(b)** After rewinding and retraining the discovered masks, however, GROW and PRUNE become indistinguishable. Thus, the advantage comes from the growth trajectory, not a better final sparse architecture, prompting our analysis of how newborn capacity integrates during training.

Edit-time learning differs from final mask quality. Figure 1 shows that GROW reaches the strongest final accuracy during in-place structural training across compactness levels, outperforming both PRUNE and DENSE. To separate this trained-state advantage from the quality of the discovered compact architecture, we use the standard winning-ticket test: when final masks are rewound and retrained from scratch, GROW and PRUNE are statistically indistinguishable. This comparison is conservative with respect to GROW: PRUNE begins at full capacity and subsequently decreases capacity, thereby experiencing substantially greater exposure to active parameters over the course of the structural trajectory, whereas GROW increases capacity from an initially sparse seed; excluding IMP rewinding reduces this discrepancy only modestly (App. B.4.2). Thus, the superior in-place endpoint achieved by GROW cannot be attributed solely to a higher-quality final mask, motivating a study of how newly added capacity is integrated across training.

4.2 EDIT-TIME INTEGRATION OF NEWBORN UNITS

The gap between final performance and the warm-started training trajectory in Sec. 4.1 suggests that GROW successfully identifies high-quality final subnetworks, but is less effective at integrating them during the warm-start editing process. To pinpoint this bottleneck, we analyze each growth event, comparing newly activated units to previously active ones. We measure forward participation by post-activation magnitude and backward credit by pre-activation gradient magnitude. For each newborn cohort, we compare activation and gradient signals to already-active units. A value of 100% means equal signal; below 100% means newborn units receive less activation or gradient signal than incumbent units.

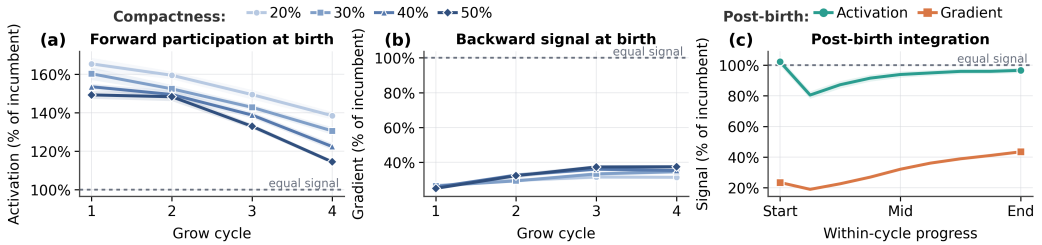

Figure 2: **Newborn units become forward-active but remain gradient-disadvantaged.** Event-aligned diagnostics for the editable ConvNet head on CIFAR-100. **(a)** Newborn activation is near or above the incumbent level immediately after insertion. **(b)** In contrast, newborn units receive only a fraction of the incumbent gradient signal. **(c)** During subsequent training, forward participation approaches parity while backward signal remains substantially weaker. Lines and shaded regions show means and 95% confidence intervals; dashed references denote equal signal.

Growth is limited by backward integration. Figure 2 rules out a simple dead-unit explanation. Forward participation is partly encouraged by the default activation-based selector, but the resulting backward disadvantage is not: newborn units receive substantially weaker gradients despite being active. Newborn units are forward-active at birth and approach incumbent activation levels, yet

receive substantially weaker gradients throughout the cycle, indicating that architectural capacity appears before comparable credit assignment. This helps explain why GROW can discover winning tickets comparable to PRUNE while exhibiting weaker edit-time performance. By contrast, PRUNE initially removes less-active units, though this selectivity weakens and survivor readjustment increases across cycles (App. B.4.2). The bottleneck also persists under gradient-based recruitment and worsens with more frequent growth under a fixed training budget (App. B.2.2). Because these diagnostics edit only the classifier head, Sec. 4.3 tests whether the same limitation strengthens when growth enters the learned feature hierarchy.

4.3 WHERE CAN NEW STRUCTURE INTEGRATE? GROWTH INSIDE A FEATURE HIERARCHY

Our previous experiments isolate structural allocation in a fully connected classifier head, leaving open whether GROW–PRUNE asymmetry persists when edits modify the learned feature hierarchy. We study this on CIFAR-100 using a CIFAR-style ResNet where structural units are channels inside residual branches. This lets us distinguish *initial structural plasticity* from *realized structural plasticity*: activating a dormant channel changes the architecture immediately, but yields useful plasticity only if the channel is integrated into the learned representation and aids further optimization.

Residual-channel editing. For residual block b, let

$$u_b = \phi(\mathrm{BN}_{b,1}(W_{b,1}h_b)), \qquad \tilde{u}_b = m_b \odot u_b, \qquad \text{and} \qquad h_{b+1} = \phi(\mathrm{BN}_{b,2}(W_{b,2}\tilde{u}_b) + S_b(h_b)),$$

where $m_b \in \{0,1\}^{d_b}$ is a binary channel mask applied after the first convolution and nonlinearity, and S_b denotes the identity or projection shortcut. Therefore, a masked structural unit comprises an output filter of $W_{b,1}$ and the corresponding input-channel slice of $W_{b,2}$. The shortcut pathway remains intact, so edits modify the residual transformation without deleting the skip connection. We compare four edit loci: (i) HEAD, which edits only the classifier head; (ii) LATE, which edits residual channels in the final stage; (iii) MID+LATE, which edits the final two stages; and (iv) ALL, which edits every residual stage. All methods use the same optimization budget and per-mask active-channel target. Thus, compactness is matched within editable masks, not in whole-network parameter count: broader loci place structural pressure on more of the feature hierarchy (App. A.5). Accordingly, cross-locus trends reflect both edit depth and the amount of feature-extractor capacity being edited.

Current-task performance degrades as editing broadens. At 50% compactness, extending structural edits through more of the residual hierarchy lowers in-place ACC for both methods: GROW falls from 61.2% under HEAD to 58.5% under ALL, while PRUNE falls from 61.9% to 58.0%. Within each locus, however, their endpoints remain close, and the same holds after retraining the final masks from scratch; under ALL, WT accuracy is 59.2% for GROW and 59.0% for PRUNE. Thus, broader editing reduces current-task performance, but endpoint accuracy alone reveals little GROW–PRUNE separation. We next ask whether similarly performing models nevertheless realize their structural capacity differently in the learned representation. Complete results are reported in App. B.4.1.

Realized representations: scale, support, and utility. Adding channels increases architectural capacity, but it need not produce a broader or more useful representation. Therefore, we ask whether growth at different edit loci changes (i) how much the learned features vary, (ii) how broadly that variation is distributed across feature directions, and (iii) how easily current-task information can be read out. At the final structural cycle, we spatially average `stage3_out`—the output of the final residual stage—to obtain one 64-dimensional feature vector per example (Apps. A.4, B.4.4). Let $H \in \mathbb{R}^{n\times d}$ contain held-out feature vectors, let $\bar{H}$ denote their centered form, and define $\Sigma = \frac{1}{n-1}\bar{H}^\top\bar{H}$. First, total covariance variance, $V(H) = \mathrm{tr}(\Sigma) = \sum_i \lambda_i$, is the variance summed across all feature directions, and therefore measures the overall spread of the representation across examples. Second, normalized effective rank measures how broadly that variation is distributed across directions, following entropy-based effective dimensionality measures used in representation and plasticity analyses (Roy & Vetterli, 2007; Dohare et al., 2024). With $p_i = \frac{\lambda_i}{\sum_j \lambda_j + \epsilon}$, we compute $\widetilde{r}_{\mathrm{eff}}(H) = \frac{1}{r_{\max}}\exp(-\sum_i p_i \log(p_i + \epsilon))$, where $r_{\max}$ is the maximum supported covariance rank. Unlike total variance, this quantity is insensitive to uniform rescaling: high values indicate variation spread across many directions, low values indicate concentration into a smaller feature subspace. Third, we train a linear classifier on frozen features and report held-out accuracy, measuring whether current-task information is accessible through a simple readout independent of the original classifier.

Figure 3 reveals two growth regimes. Under HEAD editing, GROW raises total variance from 35.4 to 53.5 and linear-probe accuracy from 55.1% to 57.0%, despite a lower normalized effective rank than

DENSE (0.516 versus 0.588). Therefore, HEAD growth amplifies a concentrated set of task-relevant directions rather than creating a uniformly broader feature basis. Growth within the residual hierarchy behaves differently. Under ALL, GROW retains more variance than PRUNE (17.7 versus 8.3), but has lower effective rank (0.405 versus 0.528) and probe accuracy (45.4% versus 50.5%), with the same pattern under LATE and MID+LATE. Thus, GROW produces a larger but more directionally concentrated feature cloud than PRUNE: added channels increase architectural capacity without yielding broadly usable feature support.

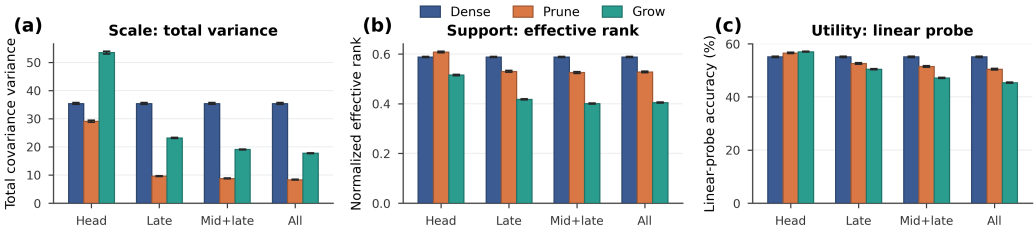

Figure 3: **Does added structural capacity become useful representational capacity?** We examine the final-cycle `stage3_out` representation at 50% compactness along three complementary axes: (a) *scale*, how much the features vary across examples; (b) *support*, how broadly that variation is distributed across feature directions; and (c) *utility*, how well current-task information can be recovered with a frozen linear probe. HEAD-localized GROW increases scale and linear utility despite concentrating variation into fewer directions, whereas growth through the residual hierarchy yields narrower support and weaker linear utility. Error bars show 95% confidence intervals.

4.3.1 BOUNDED FUTURE REACHABILITY.

We use *bounded future reachability* to mean how readily a trained checkpoint can be adapted to a changed learning signal within a fixed optimization budget. Specifically, current-task representation quality does not necessarily determine whether a checkpoint remains useful for future learning. Therefore, we ask whether structural growth changes the *speed and endpoint of subsequent adaptation*. Using a controlled *class-permutation remapping probe*, we clone selected checkpoints and fine-tune them on a fixed permutation of CIFAR-100 labels. Inputs and visual classes stay the same while output assignments change, isolating functional remapping—whether learned features can be quickly reused and reorganized without adapting to a new visual domain. All parameters are fine-tuned for the same budget of $B = 2400$ gradient steps with batch size 256. Let $0 = b_0 < \cdots < b_J = B$ denote adaptation steps and a_{b_j} the corresponding probe accuracies. We report the normalized area under the adaptation curve, $\mathrm{AUC}_B(\theta) = \frac{1}{B}\sum_{j=0}^{J-1} \frac{a_{b_j}+a_{b_{j+1}}}{2}(b_{j+1} - b_j)$, which rewards checkpoints that achieve useful performance earlier within the fixed budget. Final probe accuracy separately measures the adaptation endpoint.

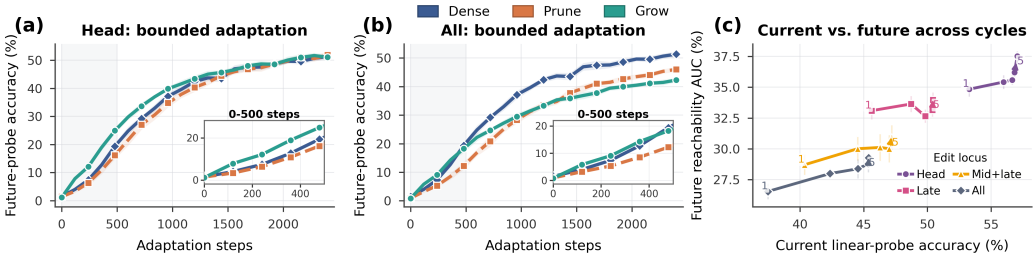

Figure 4: **The effect of growth on future reachability is locus- and budget-dependent.** We clone final-cycle checkpoints at 50% compactness and fully fine-tune them for 2400 steps after a fixed label permutation. **(a)** HEAD-localized GROW adapts fastest during the early probe and reaches approximately the same endpoint as DENSE and PRUNE. **(b)** When all residual stages are editable, DENSE retains a substantially higher adaptation trajectory; GROW has a small early advantage over PRUNE but reaches a lower final accuracy. Insets magnify the first 500 adaptation steps. **(c)** For GROW, each trajectory follows structural cycles 1–5 in the plane defined by current linear-probe accuracy and future reachability AUC. Therefore, higher and farther right indicates greater current utility and future adaptability. Error bars and shaded regions show 95% confidence intervals.

Head-localized growth enables faster remapping. Figure 4(a) reveals a marked early advantage for GROW under HEAD: after 500 steps it reaches roughly 25% accuracy, versus 19% for DENSE and

16% for PRUNE. Yet all three finish near 51% after 2400 steps. Thus, head-localized growth does not reach a better final solution; it reaches a comparable one substantially faster. When edits extend throughout the residual hierarchy, this pattern changes (Fig. 4(b)): GROW retains some early mobility but finishes at 41.9%, below PRUNE (46.2%) and DENSE (51.3%). Thus, GROW retains strong early mobility but broad residual editing lowers its bounded adaptation endpoint.

Current feature utility tracks this locus dependence. Figure 4(c) asks whether checkpoints that are more useful on the current task are also easier to repurpose. Each point represents a GROW checkpoint, with current linear-probe accuracy on the x-axis and future-remapping AUC on the y-axis; connected points trace structural cycles within each edit locus. Therefore, moving up and right means becoming both more useful now and easier to adapt later. HEAD occupies the favorable upper-right region, while broader loci remain progressively lower in both quantities. At the final cycle, AUC declines from 36.5 under HEAD to 32.9 (LATE), 29.5 (MID+LATE), and 28.4 (ALL) (Table B.8). Thus, where growth occurs strongly shapes both the representation available now and how readily it can be reused under a changed learning signal.

5 INTERVENTIONS ON THE NEWBORN INTEGRATION BOTTLENECK

Sections 4.2 and 4.3 identify two linked limitations of growth. Newborn units can participate in the forward computation while receiving weak backward signal, and this integration deficit becomes more consequential when growth modifies the learned feature hierarchy. Therefore, we evaluate four possible sources of this integration bottleneck: (i) *optimizer state* mismatch due to lacking accumulated states, (ii) disruptive *insertion* operations that perturb the current function, (iii) suboptimal *selection* of inactive units, and (iv) restricted gradient flow from *activation functions*. These interventions test which mechanisms improve newborn integration and whether the strongest gains transfer from the controlled ConvNet head to residual-channel growth.

Optimizer-state interventions: Two-Speed and Moment Transplant. Newborn parameters enter training without the optimization history accumulated by incumbent units. We test this cold-start hypothesis with two complementary interventions. TWO-SPEED, inspired by learning-rate adaptation for incrementally grown networks (Yuan et al., 2023), temporarily amplifies updates to the incoming and outgoing weights associated with newborn units. If θ_t denotes these parameters and $\theta_{t+1}^{\text{base}}$ is the update proposed by the base optimizer, we apply $\theta_{t+1} = \theta_t + r(\theta_{t+1}^{\text{base}} - \theta_t)$ with $r > 1$ during a short post-birth warm-up window. MOMENT TRANSPLANT instead initializes the newborn optimizer buffers from an incumbent donor unit. This tests whether missing accumulated state, rather than an insufficient update magnitude alone, limits early integration (Asadi et al., 2023; Behrouz et al., 2025).

Insertion intervention: Net2Wider. A second possibility is that activating a new unit perturbs an already learned function before the unit has acquired a useful role. Therefore, we implement NET2WIDER (Chen et al., 2016), which duplicates an incumbent unit and redistributes its outgoing weights so that the network function is preserved at insertion. This directly tests whether reducing insertion shock improves newborn integration. Function preservation does not, however, guarantee useful differentiation: the duplicate initially shares its donor's representation and may remain redundant or receive similarly weak credit.

Selection intervention: GradMax-style recruitment. The integration deficit may also depend on which dormant units are recruited. Our default activation-based rule selects candidates according to their prospective forward participation. In contrast, the GRADMAX-style selector (Evci et al., 2022) scores dormant candidates by the learning signal they would receive on the mini-batch available at the growth event and activates the top-ranked units within each editable layer. This changes only the *selection policy*; initialization, optimizer handling, and post-insertion training remain unchanged. Therefore, it tests whether weak newborn gradients arise because the default rule recruits candidates that are forward-active but poorly coupled to the current loss.

Activation-function intervention: Rand. Smooth-Leaky. Finally, the bottleneck may reflect restricted gradient transmission rather than optimizer state, insertion shock, or candidate selection. Because newborn units are forward-active yet gradient-disadvantaged, changing the nonlinearity provides a direct intervention on their backward pathway while leaving the structural procedure unchanged. We use RAND. SMOOTH-LEAKY, a randomized smooth leaky activation function previously shown to preserve plasticity by maintaining non-zero negative-side gradients and a gradual transition around the origin (Lillo & Cheney, 2025). This tests whether improving generic activation-

level trainability is sufficient to help newborn units acquire useful roles. Additional activation controls and hyperparameters are reported in Appendix C.1.

Interventions decouple current performance from future adaptability. The birth interventions reveal that improving newborn integration does not necessarily improve the current endpoint. Across the controlled ConvNet and ResNet experiments, changes in newborn backward signal are only weakly reflected in in-place ACC, indicating that better local integration is not sufficient to determine current-task performance (App. B.4.5). The clearest case is RAND. SMOOTH-LEAKY: it improves newborn gradient signal while yielding lower current accuracy, yet produces substantially higher future-reachability AUC, particularly when growth extends through the residual hierarchy. Thus, an intervention can improve the optimization state left by structural growth without improving the solution currently expressed by the network. This distinction motivates Sec. 6, where we test whether such improved trainability becomes useful when the learning signal repeatedly changes.

6 CONTINUAL LEARNING ACROSS SHIFT REGIMES

If interventions increase a grown network's future adaptability without improving its current endpoint, the key test is whether the gain persists under repeated changes in the learning signal. Our earlier experiments showed that growth is limited not only by which units are added, but by whether newborn units can integrate into a trained representation. We use the following benchmarks to compare how the same structural operators behave under continual shift, rather than to benchmark complete continual-learning methods against one another. We study two regimes: (i) sequential class accumulation, with longer intervals between shifts, and (ii) repeated-shift benchmarks that reveal continuing-learning degradation (Dohare et al., 2024).

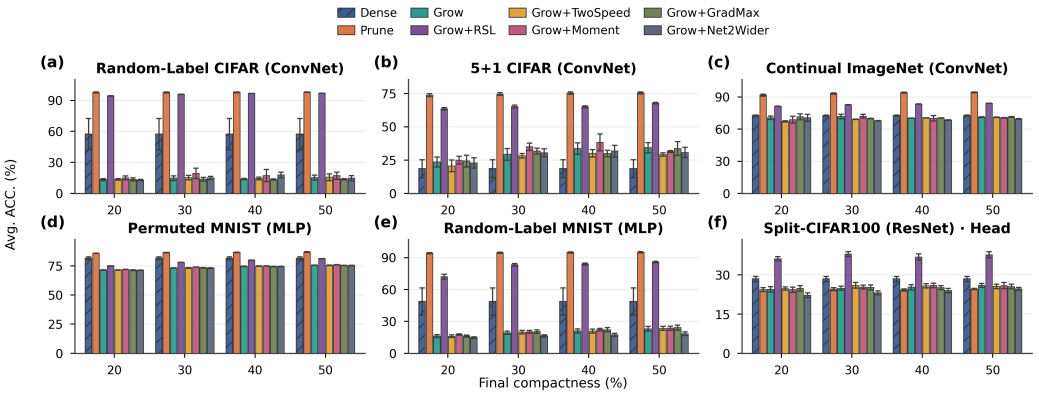

Figure 5: **Structural editing across sequential and repeated-shift continual learning.** We report average online accuracy across five repeated-shift benchmarks and ResNet Split-CIFAR100. PRUNE is the most reliable structural baseline under rapid shifts, whereas GROW + RAND. SMOOTH-LEAKY (RSL) is the most consistent growth variant and strongly improves over vanilla GROW on Split-CIFAR100; activation-matched DENSE and PRUNE controls are reported in App. C.

Growth under sequential class accumulation. We evaluate Split-CIFAR100 using the ResNet architecture from Sec. 4.3. Task boundaries are known during training, but tasks share one classifier and task identity is unavailable at test time. At each boundary, the model either stays dense or is structurally edited before learning the next subset. Since Split-CIFAR100 provides longer intervals between shifts, it tests whether newborn units become useful when given more time to integrate. Figure 5 shows that GROW+RAND. SMOOTH-LEAKY substantially improves average online accuracy over vanilla GROW and the other birth interventions. Although Rand. Smooth-Leaky also benefits non-growing models (App. C), its gain is largest for GROW, consistent with improved newborn integration when sufficient adaptation time is available.

Growth under repeated non-stationary shifts. To distinguish this effect from changes in capacity or interference alone, we also evaluate five benchmarks designed to stress repeated-shift plasticity (Kumar et al., 2023). **Permuted MNIST** (Goodfellow et al., 2013) applies a new fixed pixel permutation at each task; **Random-Label MNIST** and **Random-Label CIFAR** (Lyle et al., 2023) repeatedly assign random labels to a fixed data subset; **CIFAR 5+1** alternates harder five-class and easier single-class tasks drawn from CIFAR-100; and **Continual ImageNet** (Dohare et al., 2024; Russakovsky et al., 2015) presents non-repeating binary classification tasks drawn from distinct

ImageNet classes. Together, these benchmarks span input and concept shift while repeatedly testing whether the current network remains capable of learning the next task.

Growth is competitive but more regime-sensitive under continual shift. Across the repeated-shift benchmarks, PRUNE is generally the strongest or near-strongest structural method, consistent with the advantage of retaining units that are already integrated into the representation and optimization state. Vanilla GROW can match or exceed DENSE in some cases, including CIFAR 5+1, but its gains are less stable. Among the growth variants, GROW+RAND. SMOOTH-LEAKY is the most reliable. Because this activation also improves DENSE and PRUNE in several benchmarks, App. C compares its effect within each structural method; the gains are often largest for GROW. Figure 6 helps explain this advantage on sequential class accumulation: on Split-CIFAR100, Rand. Smooth-Leaky keeps newborn activations near incumbent levels while substantially improving newborn gradient signal across repeated growth cycles, making new units more trainable under repeated structural edits. Moment Transplant and GradMax provide occasional gains, while TwoSpeed and Net2Wider are less effective, consistent with Fig. B.5. In this controlled comparison, growth can match or beat pruning in some regimes, but works best when newborn units stabilize before the next shift.

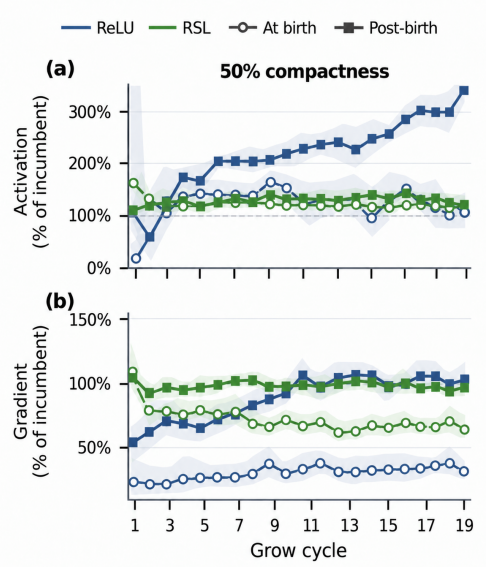

Figure 6: **Rand. Smooth-Leaky stabilizes repeated birth events on Split-CIFAR100.** At 50% compactness, we compare newborn vs. incumbent activations and gradient signal relative to incumbent units across 20 growth cycles. With ReLU, newborn gradients are low at birth and their activations rise over cycles. Rand. Smooth-Leaky (RSL) keeps newborn activations near incumbent levels and boosts post-birth gradients toward parity, improving usability under sequential growth.

7 CONCLUSION & FUTURE WORK

Conclusion Growth and pruning are complementary forms of structural plasticity but not optimization-symmetric. Given similar compactness and compute, both can yield competitive sparse architectures; they mainly differ in the trajectories that produce them. New units are added to an already specialized network and may activate before receiving comparable backward credit, creating a lag between architectural expansion and functional integration. The consequences of this lag are weak in small MLPs, become visible in convolutional feature learning, and become increasingly consequential when edits enter the residual hierarchy. Head-localized growth can produce concentrated, task-aligned representations and enable rapid functional remapping, whereas broader residual growth yields narrower feature support and lower bounded future adaptability despite similar current endpoints. Our interventions further show that improving newborn trainability need not improve immediate accuracy: it can instead change the optimization state left by growth and make future solutions more reachable. Under continual shift, GROW becomes more competitive when newborn units are sufficiently trainable and have enough time to integrate. Overall, structural growth should be judged not only by the architecture it produces, but by whether the capacity it adds becomes usable for subsequent optimization.

Future Work These findings suggest that growth rules should optimize for *realized*, not merely added, capacity. A natural next step is to replace fixed growth schedules with integration-aware policies that decide when and where to add structure using signals such as newborn-to-incumbent gradient balance, representational support, or short-horizon bounded reachability. Such policies could delay subsequent edits until newly added capacity has stabilized, or redirect growth toward loci where additional structure remains readily trainable. A second direction is to test whether these principles persist in larger residual and attention-based architectures and against end-to-end continual-learning and modular expansion methods. More broadly, structural adaptation should be evaluated jointly by the architecture it produces, its immediate task utility, and the optimization state it leaves behind.

REFERENCES

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. In *Conference on lifelong learning agents*, pp. 620–636. PMLR, 2023.

Kavosh Asadi, Rasool Fakoor, and Shoham Sabach. Resetting the optimizer in deep rl: An empirical study. *Advances in Neural Information Processing Systems*, 36:72284–72324, 2023.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. *arXiv preprint arXiv:2512.24695*, 2025.

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

Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8281–8290, 2021.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

Nicholas Cheney, Martin Schrimpf, and Gabriel Kreiman. On the robustness of convolutional neural networks to internal architecture and weight perturbations. *arXiv preprint arXiv:1703.08245*, 2017.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

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

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632: 768—774, 2024.

Ufuk Evci, Trevor Gale, Jacob Menick, Pablo Sampedro, Eliana Lorch, and Jascha Sohl-Dickstein. Rigging the lottery: Making all tickets winners. In *NeurIPS*, 2020.

Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. Gradmax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=qjN4h_wwUO.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv: Learning*, 2019. URL https://api.semanticscholar.org/CorpusID:182952565.

Yasir Ghunaim, Adel Bibi, Kumail Alhamoud, Motasem Alfarra, Hasan Abed Al Kader Hammoud, Ameya Prabhu, Philip HS Torr, and Bernard Ghanem. Real-time evaluation in online continual learning: A new hope. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11888–11897, 2023.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

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

Haeyong Kang, Rusty John Lloyd Mina, Sultan Rizky Hikmawan Madjid, Jaehong Yoon, Mark Hasegawa-Johnson, Sung Ju Hwang, and Chang D. Yoo. Forget-free continual learning with winning subnetworks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 10734–10750. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/kang22b.html.

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

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114 (13):3521–3526, 2017.

Timo Klein, Christoph Luther, Manus McAuliffe, Lukas Miklautz, Claudia Plant, and Sebastian Tschiatschek. Plasticity loss in deep reinforcement learning: A survey. *arXiv preprint arXiv:2411.04832*, 2024.

Alex Krizhevsky. Learning multiple layers of features from tiny images. In *University of Toronto Technical Report*, 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity in continual learning via regenerative regularization. *arXiv preprint arXiv:2308.11958*, 2023.

Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic sparse training with structured sparsity. *arXiv preprint arXiv:2305.02299*, 2023.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International conference on machine learning*, pp. 3925–3934. PMLR, 2019.

Lute Lillo and Nick Cheney. Activation function design sustains plasticity in continual learning. *arXiv preprint arXiv:2509.22562*, 2025.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pp. 23190–23211. PMLR, 2023.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 67–82, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Thomas Miconi. Neural networks with differentiable structure. *arXiv preprint arXiv:1606.06216*, 2016.

Decebal Constantin Mocanu et al. Scalable training of artificial neural networks with adaptive sparse connectivity. In *AAAI*, 2018.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2020.

Ameya Prabhu, Zhipeng Cai, Puneet Dokania, Philip Torr, Vladlen Koltun, and Ozan Sener. Online continual learning without the storage constraint. *arXiv preprint arXiv:2305.09253*, 2023.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. *2007 15th European Signal Processing Conference*, pp. 606–610, 2007. URL `https://api.semanticscholar.org/CorpusID:12184201`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016. URL `https://proceedings.mlr.press/v48/wei16.html`.

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

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

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

Li Yang, Sen Lin, Junshan Zhang, and Deliang Fan. Grown: Grow only when necessary for continual learning. *arXiv preprint arXiv:2110.00908*, 2021.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Xin Yuan, Pedro Savarese, and Michael Maire. Accelerated training via incrementally growing neural networks using variance transfer and learning rate adaptation. *Advances in Neural Information Processing Systems*, 36:16673–16692, 2023.

Yuqing Zhao, Divya Saxena, Jiannong Cao, Xiaoyun Liu, and Changlin Song. Overcoming growth-induced forgetting in task-agnostic continual learning. *arXiv preprint arXiv:2408.10566*, 2024.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020.

A Datasets, Benchmarks and Hyperparameters

A.1 Datasets and Benchmarks

We evaluate structural adaptation under three data regimes: (i) stationary i.i.d. supervised learning, (ii) non-stationary single-head class-incremental streams, and (iii) continual-learning benchmarks designed to stress plasticity under repeated shift. In all cases, we use the standard train/test splits provided with each dataset unless otherwise specified, and the test set is used only for evaluation.

A.1.1 Independently and identically distributed (i.i.d.)

MNIST MNIST is treated as a stationary 10-class classification problem. We train on the full training set and evaluate on the standard MNIST test set. Each run uses a fixed budget of 5 Grow or Prune cycles, with 20 epochs per cycle. Winning-ticket retraining runs for 100 epochs to match the total cycle-training budget.

CIFAR-100 CIFAR-100 is treated as a stationary 100-class classification problem. We train on the full training set and evaluate on the standard CIFAR-100 test set. Each run uses 200 epochs total for dense training and winning-ticket retraining. Cycle training is divided into 5 Grow or Prune cycles, with 40 epochs per cycle. This setting does not use experience replay.

A.1.2 Class-Incremental Continual Learning

Class-incremental Split-MNIST and Split-FashionMNIST MNIST and FashionMNIST each contain 10 classes. We construct a stream of $K = 5$ tasks by partitioning the classes into five disjoint class pairs. We use a single shared 10-way classifier head throughout training, with no task-id routing and no multi-head evaluation. At task $t \in \{1, \ldots, K\}$, training uses only the two classes assigned to task t, while evaluation uses the cumulative test set containing all classes observed up to task t. Unless otherwise noted, the class-pair ordering is randomized per run using a fixed seed, and the same ordering is shared across methods within each seed.

Experience replay To avoid forgetting-dominated collapse in the single-head class-incremental setting, we optionally use experience replay. The replay buffer stores up to $M = 50$ examples per class, with a maximum total size of 200 examples. During training on tasks $t > 1$, each mini-batch mixes current-task samples with replay samples. After finishing task t, we add up to M examples for each newly observed class. Unless explicitly stated as "no replay," Split-MNIST and Split-FashionMNIST use TinyER with a replay fraction of 0.5 and a fixed budget of 20 epochs per task (100 epochs total across 5 tasks), aligned with the structural-edit cycles.

Split-CIFAR100 (class-incremental) We also evaluate class-incremental Split-CIFAR100 as a sequential accumulation benchmark. We partition CIFAR-100 into 20 disjoint 5-class tasks and train for 10 epochs per task with batch size 64, without experience replay, and the learner is trained without task-identity information at test time. As in the other single-head stream settings, we use a shared 100-way classifier head throughout training and evaluate cumulatively over all classes observed so far where identity is unavailable at test time. At each task boundary, the model either remains dense or undergoes a structural edit before continuing optimization on the new task. Results for this setting are reported in the main text continual-learning section.

A.1.3 Plasticity-stressing continual-learning benchmarks

Following Kumar et al. (2023), we evaluate five supervised continual image-classification benchmarks spanning two shift types: *input-distribution shift* (Permuted MNIST, 5+1 CIFAR, and Continual ImageNet) and *concept shift* (Random Label MNIST and Random Label CIFAR). Across all settings, training proceeds as a sequence of tasks without task-identity signals: the model is never told when a task switch occurs. Within each task, the learner receives mini-batches for a fixed duration and is updated incrementally with cross-entropy on the arriving batches. Summary hyperparameters are reported in Table A.1.

Permuted MNIST Permuted MNIST (Goodfellow et al., 2013) is used as an input-shift benchmark. We first sample a fixed subset of 10,000 images from the MNIST training set. Each task is defined by drawing a new fixed random permutation over pixel indices and applying it to every image in the subset. The permutation remains constant within a task and is independent across tasks. Each task presents exactly one pass over its 10,000 permuted images in mini-batches of size 16, after which the next task begins with a new permutation. We train for 500 tasks in total.

Random Label MNIST Random Label MNIST (Lyle et al., 2023) is used as a concept-shift benchmark. We fix a subset of $1{,}200$ MNIST images once, and for each task generate a fresh random label for every image in the subset. The inputs are unchanged across tasks, but the input-label mapping changes completely. To encourage memorization under an arbitrary target function, the model is trained for 400 epochs per task with batch size 16. After each task, a new independent random labeling is sampled. We run 50 tasks in sequence.

Random Label CIFAR Random Label CIFAR follows the same protocol as Random Label MNIST, but uses images drawn from CIFAR-10. We again fix a subset of $1{,}200$ images, reassign random labels independently for each task, and train for 400 epochs per task with batch size 16 over 50 tasks.

5+1 CIFAR 5+1 CIFAR is an input-shift benchmark with alternating task difficulty. Tasks are constructed from CIFAR-100 and alternate between *hard* tasks containing 5 classes ($2{,}500$ images total, 500 per class) and *easy* tasks containing a single class (500 images). Classes do not repeat across the sequence. Each task lasts 780 parameter-update steps. With batch size 32, this corresponds to approximately 10 epochs on hard tasks and approximately 50 epochs on easy tasks. We report performance on the hard tasks only, since single-class tasks are near ceiling for all methods.

Continual ImageNet Continual ImageNet (Dohare et al., 2024; Russakovsky et al., 2015) is used as an input-shift benchmark. Each task is a binary classification problem between two distinct ImageNet classes. For every task, we draw $1{,}200$ images total (600 per class) and downsample them to 32×32, following Dohare et al. (2024), to reduce compute while preserving semantic variability. Classes do not repeat across tasks. We train for 10 epochs per task with batch size 100 and report task accuracy.

Benchmark	Per-Task Data Size	Batch	Epochs	Timesteps	# Tasks
Permuted MNIST	10,000 images	16	1	625	500
Random Label MNIST	1,200 images	16	400	30,000	50
Random Label CIFAR	1,200 images	16	400	30,000	50
5+1 CIFAR	Hard: 2,500 images (5 classes, 500/class)	32	Hard: ≈ 10	780	15
	Easy: 500 images (1 class)		Easy: ≈ 50		15
Continual ImageNet	1,200 images/task (600/class)	100	10	120	500

Table A.1: Hyperparameters and schedule per benchmark. *Timesteps* denote parameter-update steps (i.e., mini-batches) within a task. For 5+1 CIFAR, a fixed timestep budget per task implies approximate epochs depending on data size.

Notes. (i) In 5+1 CIFAR, classes do not repeat across tasks; tasks alternate easy/hard. 780 timesteps ≈ 10 epochs on the hard set (since $2{,}500/32 \approx 78.125$ batches/epoch) and ≈ 50 epochs on the easy set (since $500/32 \approx 15.625$). (ii) In Continual ImageNet, images are downsampled to 32×32 to reduce compute; classes do not repeat across tasks. (iii) Timesteps are computed as the number of mini-batches per task.

A.2 Hyperparameters, Optimizers and Learning Rates

For each MLP, ConvNet, and ResNet dataset–architecture setting, we sweep optimizers in $\{\text{SGD}, \text{Adam}, \text{AdamW}\}$ and learning rates $\eta \in \{0.1, 0.01, 0.001, 0.0001\}$ independently for each treatment. For each method, we select the configuration with the highest mean final accuracy. Thus, all methods receive the same hyperparameter search space and tuning budget, rather than being evaluated under a configuration favored by another treatment. Within each selected configuration, the remaining optimizer hyperparameters, training budget, and schedule are fixed; they differ only through their prescribed structural-edit procedure and resulting active units (App. A.4). Unless stated otherwise, all models use ReLU activations, and all `Conv2d` and `Linear` layers use Kaiming uniform initialization, except ResNet `Linear` layers which use Xavier uniform initialization.

The continual-learning benchmarks in Sec. 6 use a separate tuning protocol: Dense uses the dataset-specific reference learning rate, while non-dense methods select the best learning rate by average accuracy from $\{0.01, 0.001, 0.0001\}$. For the activation-control analysis in App. C, learning rates are selected independently for ReLU and Rand. Smooth-Leaky before computing activation-specific differences.

Learning-rate schedule For all experiments in Sections 4 and 5 we apply cosine annealing over the full training horizon with $T_{\max} = N_{\text{epochs}}$ and $\eta_{\min} = 0$, so the learning rate decays smoothly from its initial value to zero over 200 epochs.

A.3 Performance Metrics: ACC, trajectory-average accuracy (TAA), and TAOA

We use the same metric names and evaluation conventions as in the main text. Our primary evaluation index is defined by GROW/PRUNE checkpoints: in class-incremental streams these checkpoints coincide with task boundaries, while in i.i.d. settings they simply mark successive structural-edit cycles.

Checkpoint indexing Let $t \in \{1, \dots, T\}$ index evaluation checkpoints along training. In our implementation, checkpoints occur at the end of each GROW/PRUNE cycle. In class-incremental streams, this aligns t with task time; in i.i.d. settings, t indexes successive structural-edit events.

Class-incremental settings When tasks exist, let K_t be the number of tasks encountered up to checkpoint t, and let $\mathrm{Acc}_{t,k}$ denote test accuracy on task $k \in \{1, \dots, K_t\}$ measured at checkpoint t. We define cumulative accuracy at checkpoint t as

$$\mathrm{Acc}_t^{\mathrm{cum}} \;=\; \frac{1}{K_t} \sum_{k=1}^{K_t} \mathrm{Acc}_{t,k}. \tag{1}$$

We then report

$$\mathrm{ACC} \;=\; \mathrm{Acc}_T^{\mathrm{cum}}, \tag{2}$$

$$\mathrm{TAA} \;=\; \frac{1}{T} \sum_{t=1}^{T} \mathrm{Acc}_t^{\mathrm{cum}}. \tag{3}$$

Thus, ACC summarizes end-of-stream performance, while TAA summarizes performance over the full trajectory.

I.i.d. settings When there are no tasks, let Acc_t be the test accuracy at checkpoint t. We report

$$\mathrm{ACC} \;=\; \mathrm{Acc}_T, \tag{4}$$

$$\mathrm{TAA} \;=\; \frac{1}{T} \sum_{t=1}^{T} \mathrm{Acc}_t. \tag{5}$$

In this case, ACC is the standard final test accuracy and TAA is the average test accuracy over the checkpoint-defined training trajectory.

In-place vs. Winning-Ticket evaluation Both *In-place* and *Winning-Ticket* results report ACC and TAA using the definitions above. In-place metrics are measured during the GROW/PRUNE procedure while the mask changes across checkpoints. Winning-Ticket metrics are measured after freezing the discovered mask, reinitializing the model, and retraining it from scratch under the same data stream and replay setting, following standard Lottery Ticket evaluation (Frankle & Carbin, 2018).

Total Average Online Accuracy (TAOA) On the continual-learning benchmarks, we additionally use Total Average Online Accuracy (TAOA), following prior online continual-learning work (Cai et al., 2021; Ghunaim et al., 2023; Prabhu et al., 2023; Kumar et al., 2023). Unlike ACC and TAA, which are checkpoint-based, TAOA aggregates online accuracy over all mini-batches seen so far. Let $B_{\le T} = \sum_{i=1}^{T} M_i$ be the total number of processed mini-batches up to task T, and let a_t denote online accuracy at global batch index t. We define

$$\mathrm{TAOA}_{\le T} = \frac{1}{B_{\le T}} \sum_{t=0}^{B_{\le T}-1} a_t. \tag{6}$$

If all tasks have equal length $M_i \equiv M$, this reduces to

$$\mathrm{TAOA}_{\le T} = \frac{1}{MT} \sum_{t=0}^{MT-1} a_t. \tag{7}$$

We use TAOA in Section 6 to capture how quickly the agent learns the current task (plasticity) and distinguish it from the checkpoint-based ACC and TAA reported in the rest of the main paper.

A.4 NEURAL NETWORK ARCHITECTURES

All methods (DENSE, GROW, PRUNE) share the same underlying parameterization and differ only in the binary unit masks that determine which hidden units are active during forward and backward passes. This ensures that comparisons isolate the effect of structural adaptation rather than changes in the base architecture.

Unit-wise masking For a hidden layer ℓ with pre-activation

$$z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}, \tag{8}$$

and activation function $\phi(\cdot)$, we define

$$h^{(\ell)}_{\text{raw}} = \phi\Big(z^{(\ell)}\Big), \qquad h^{(\ell)} = h^{(\ell)}_{\text{raw}} \odot m^{(\ell)}, \tag{9}$$

where $m^{(\ell)} \in \{0,1\}^{d_\ell}$ is a unit-wise binary mask and $\odot$ denotes element-wise multiplication. Mask entries set to zero fully deactivate the corresponding hidden units, affecting both activations and gradients. Masks are stored as non-trainable buffers and are therefore serialized with the model state.

A.4.1 MLP FOR MNIST AND FASHIONMNIST

Our MLP backbone is a 3-layer fully connected network:

$$\texttt{fc1}: \ 784 \to H, \tag{10}$$
$$\texttt{fc2}: \ H \to H, \tag{11}$$
$$\texttt{fc3}: \ H \to 10, \tag{12}$$

with two masked hidden layers (`fc1`, `fc2`) and an unmasked output layer (`fc3`). Unless otherwise noted, $H = 256$.

Dense training corresponds to $m^{(1)} = \mathbf{1}$ and $m^{(2)} = \mathbf{1}$ throughout. Growth begins from a small active fraction (default 10%) and expands masks over time by flipping selected zeros to ones. Pruning starts fully active and removes units by setting mask entries to zero.

A.4.2 CONVNET FOR CIFAR-10 & CIFAR-100

For both CIFAR-10 and CIFAR-100 we use a fully active convolutional trunk and a growable/prunable MLP head.

Convolutional trunk The trunk is

$$\texttt{conv1}: \ 3 \to 32, \ k=3, \ p=1 \ \to \ \phi \ \to \ \texttt{maxpool}(2), \tag{13}$$
$$\texttt{conv2}: \ 32 \to 64, \ k=3, \ p=1 \ \to \ \phi \ \to \ \texttt{maxpool}(2), \tag{14}$$

followed by flattening to $8 \times 8 \times 64 = 4096$ features. No structural adaptation is applied in the convolutional trunk.

Masked fully connected head The head has four fully connected layers:

$$\texttt{fc1}: \ 4096 \to H_1 \ \to \ \phi \ \to \ \odot m^{(1)}, \tag{15}$$
$$\texttt{fc2}: \ H_1 \to H_2 \ \to \ \phi \ \to \ \odot m^{(2)}, \tag{16}$$
$$\texttt{fc3}: \ H_2 \to H_3 \ \to \ \phi \ \to \ \odot m^{(3)}, \tag{17}$$
$$\texttt{fc4}: \ H_3 \to C, \tag{18}$$

where C is the number of output classes. By default, $(H_1, H_2, H_3) = (512, 512, 256)$. In all cases, structural adaptation is confined to the fully connected head.

A.4.3 RESNET FOR CIFAR-100

For the CIFAR-100 feature-hierarchy experiments, we use a CIFAR-style ResNet with a 3×3 convolutional stem followed by three residual stages. The stage widths are $(16, 32, 64)$, with three basic blocks per stage. The first block of `stage2` and `stage3` uses stride two; all remaining convolutions use stride one. Projection shortcuts are used whenever the spatial resolution or channel dimension changes.

The resulting feature hierarchy is

$$\texttt{stem}: \quad 3 \times 32 \times 32 \to 16 \times 32 \times 32, \tag{19}$$
$$\texttt{stage1}: \quad 16 \times 32 \times 32 \to 16 \times 32 \times 32, \tag{20}$$
$$\texttt{stage2}: \quad 16 \times 32 \times 32 \to 32 \times 16 \times 16, \tag{21}$$
$$\texttt{stage3}: \quad 32 \times 16 \times 16 \to 64 \times 8 \times 8. \tag{22}$$

After `stage3`, global average pooling produces a 64-dimensional vector. The classifier head is

$$\texttt{fc1}: \quad 64 \to 256 \to \phi, \tag{23}$$
$$\texttt{fc2}: \quad 256 \to C, \tag{24}$$

where $C = 100$ for CIFAR-100.

Masked residual units. Each basic block contains two 3×3 convolutions. Writing $S_b(\cdot)$ for the identity or projection shortcut, the block computes

$$u_b = \phi(\mathrm{BN}_{b,1}(W_{b,1} h_b)), \tag{25}$$
$$\widetilde{u}_b = m_b \odot u_b, \tag{26}$$
$$h_{b+1} = \phi(\mathrm{BN}_{b,2}(W_{b,2} \widetilde{u}_b) + S_b(h_b)), \tag{27}$$

where $m_b \in \{0,1\}^{d_b}$ is broadcast across the spatial dimensions. The mask is applied after the first batch-normalization and activation and before the second convolution.

A masked residual unit consists of one output channel of $W_{b,1}$, its associated batch-normalization channel, and the corresponding input-channel slice of $W_{b,2}$. Setting its mask entry to zero removes that channel from the residual computation and blocks its forward contribution and gradient signal. The skip pathway remains active, so masking changes the learned residual transformation without removing the block's identity connection. The stem, second-convolution output channels, shortcut pathways, and final output layer are never masked.

Structural edit loci. We control where structural changes occur through four mutually exclusive edit-locus configurations:

HEAD: only the 256 hidden units of `fc1` are masked; all residual stages remain fully active.

LATE: the hidden residual channels in every block of `stage3` are masked; the classifier head and earlier stages remain fully active.

MID+LATE: the hidden residual channels in every block of `stage2` and `stage3` are masked; `stage1` and the classifier head remain fully active.

ALL: the hidden residual channels in every block of `stage1`, `stage2`, and `stage3` are masked; the classifier head remains fully active.

Thus, the loci are not cumulative with respect to HEAD: feature-level edit loci modify residual channels but do not additionally mask `fc1`.

Diagnostic feature readouts. The model records representations after the stem and after the final block of each residual stage:

$$\texttt{stem_out} \in \mathbb{R}^{16\times32\times32}, \tag{28}$$
$$\texttt{stage1_out} \in \mathbb{R}^{16\times32\times32}, \tag{29}$$
$$\texttt{stage2_out} \in \mathbb{R}^{32\times16\times16}, \tag{30}$$
$$\texttt{stage3_out} \in \mathbb{R}^{64\times8\times8}. \tag{31}$$

A.5 COMPACTNESS AND SPARSITY

Our structural interventions operate through unit-wise binary masks applied to selected hidden layers. Therefore, we define compactness budgets in terms of active units and match these budgets across DENSE, GROW, and PRUNE.

Layer compactness For a masked layer ℓ with d_ℓ units and mask $m^{(\ell)} \in \{0,1\}^{d_\ell}$, the number of active units is

$$a_\ell = \|m^{(\ell)}\|_0 = \sum_{j=1}^{d_\ell} m_j^{(\ell)}, \tag{32}$$

and the layer compactness is

$$c_\ell = \frac{a_\ell}{d_\ell} \in (0,1]. \tag{33}$$

Global weight-space compactness for MLPs and ConvNets. For the MLP and ConvNet experiments, the final compactness target $c \in (0,1]$ specifies the fraction of structured weights retained or activated across all masked layers. We allocate this global weight budget across layers and convert the resulting allocations into integer unit targets. Consequently, c is primarily a weight-space budget and need not produce the same active-unit fraction in every masked layer.

Per-mask channel compactness for ResNets. The ResNet edit-locus experiment instead uses a per-mask channel target. Let $\mathcal{G}_\mathcal{L}$ denote the collection of masks editable under locus $\mathcal{L}$. For each mask $g \in \mathcal{G}_\mathcal{L}$ with d_g structural units, the target compactness c determines

$$a_g^\star = \lfloor c\, d_g \rceil\,, \qquad c_g = \frac{\|m_g\|_0}{d_g} \approx c, \tag{34}$$

up to integer rounding. The same target fraction is therefore applied independently to every editable residual block, or to `fc1` in the HEAD condition.

For example, at $c = 0.5$:

- HEAD retains approximately 128 of the 256 `fc1` units;
- LATE retains approximately 32 of 64 hidden channels in each `stage3` block;
- MID+LATE retains approximately 16 of 32 channels in each `stage2` block and 32 of 64 in each `stage3` block;
- ALL additionally retains approximately 8 of 16 channels in each `stage1` block.

The corresponding mask sparsity is $s_g = 1 - c_g$. Because each locus exposes a different number and width of masks, the same reported compactness does *not* imply the same whole-network parameter sparsity across HEAD, LATE, MID+LATE, and ALL. Instead, it matches the fraction of active structural units within every editable block. This avoids confounding edit depth with a global allocator that could redistribute the budget toward particular stages.

Matched structural budgets. Within a dataset–architecture setting and edit locus, GROW and PRUNE are budget-matched when they reach the same integer targets $\{a_g^\star\}_{g \in \mathcal{G}_\mathcal{L}}$. Growth approaches these targets from a sparse initial mask, whereas pruning approaches them from a fully active mask. Dense keeps all units active and is locus-agnostic.

Masking induces *effective structured sparsity*: inactive producer filters and consumer slices remain allocated in the parameter tensors but do not participate in the forward or backward computation. Reported compactness therefore describes active computational structure rather than compressed model-storage size.

A.6 STRUCTURAL ADAPTATION PROCEDURE

Both GROW and PRUNE operate on the same masked backbone (App. A.4) and train for T cycles. Structural edits occur during cycles $t = 1, \ldots, T-1$, reaching the target compactness before the final cycle; cycle T therefore trains the fixed target architecture without an additional edit. Each cycle consists of (i) selecting units to activate or deactivate, (ii) updating the masks and any associated bookkeeping, and (iii) training for a fixed budget before the next cycle.

A.6.1 GROW

Selection rule At cycle t, for each masked layer ℓ, we compute a score $s_j^{(\ell)}$ for each currently inactive unit j with $m_j^{(\ell)} = 0$. Then, we activate the top-$n_{t,\ell}$ inactive units in that layer. Let $a_\ell = \|m^{(\ell)}\|_0$ be the current active count and $a_\ell^\star$ the target.

For $t \in \{1, \dots, T-1\}$, the remaining number of units to add is

$$q_{t,\ell}^{\text{grow}} = \max(0, a_\ell^\star - a_\ell).$$

and we activate

$$n_{t,\ell} = \min\left(q_{t,\ell}^{\text{grow}}, \left\lceil \frac{q_{t,\ell}^{\text{grow}}}{T-t} \right\rceil, d_\ell - a_\ell\right)$$

highest-scoring inactive units.

For the MLP and ConvNet experiments, the layer targets are obtained from the global structured-weight allocator; for the ResNet, the same active-channel fraction is applied independently to every mask in the selected edit locus (App. A.5).

Activation-frequency heuristic Our default growth heuristic scores an inactive unit by the fraction of post-activation values exceeding a threshold τ (i.e. $\tau = 0.05$). Given a mini-batch B, we define

$$s_j^{\text{Act}} = \begin{cases} \frac{1}{|B|} \sum_{x \in B} \mathbf{1}\{A_j(x) > \tau\}, & \text{fully connected,} \\ \frac{1}{|B|} \sum_{x \in B} \left(\frac{1}{HW} \sum_{u \in [H] \times [W]} \mathbf{1}\{A_j(x,u) > \tau\} \right), & \text{convolutional.} \end{cases} \tag{35}$$

Here A_j denotes the *raw* post-activation value computed before the binary mask is applied. This permits dormant units to be evaluated even though their masked contribution to the network output is zero. For convolutional units, the score averages over both examples and spatial locations. Intuitively, the score, s_j^{Act}, estimates how often an inactive unit would produce a non-negligible response if activated.

A.6.2 PRUNE

Selection rule. PRUNE starts from a fully active network and removes units over T cycles until reaching the target per-layer unit counts.

For $t \in \{1, \dots, T-1\}$, the remaining number to remove is

$$q_{t,\ell} = a_\ell - a_\ell^\star,$$

and we prune

$$k_{t,\ell} = \min\left(q_{t,\ell}, \left\lceil \frac{q_{t,\ell}}{T-t} \right\rceil, a_\ell\right)$$

units from the active set by selecting the lowest-scoring units.

Magnitude score. Our default pruning score is the mean absolute weight magnitude per unit. For a `Linear` layer,

$$s_j^{\text{Mag}} = \frac{1}{d_{\text{in}}} \sum_{i=1}^{d_{\text{in}}} |W_{j,i}|, \tag{36}$$

and for a `Conv2d` layer,

$$s_j^{\text{Mag}} = \frac{1}{C_{\text{in}} k_H k_W} \sum_{c,u,v} |W_{j,c,u,v}|. \tag{37}$$

We prune the smallest-magnitude units among the currently active set.

IMP rewind. Our default pruning procedure uses IMP-style rewinding. After updating the mask, we rewind surviving parameter slices to their initialization (or stored rewind snapshot) before retraining, using the current mask to select the surviving rows, filters, and input columns as needed. This isolates the effect of subnetwork selection from continued fine-tuning dynamics.

B ADDITIONAL EXPERIMENTAL STUDIES AND ABLATIONS

B.1 COMPACTNESS-RESOLVED MLP WINNING-TICKET RESULTS

Table B.1 expands the MLP results from Sec. 4 across compactness levels. On IID MNIST, Split-MNIST+ER, and Split-Fashion+ER, GROW and PRUNE achieve closely matched winning-ticket ACC and TAA with no stable ordering. The $\Delta_{\text{WT}-\text{C}}$ values vary by dataset, but overall these results confirm that small MLP settings do not expose a robust GROW–PRUNE asymmetry.

Method	20%			30%			40%			50%		
	WT Final	**WT TAA**	$\Delta_{\mathrm{WT-C}}$	**WT Final**	**WT TAA**	$\Delta_{\mathrm{WT-C}}$	**WT Final**	**WT TAA**	$\Delta_{\mathrm{WT-C}}$	**WT Final**	**WT TAA**	$\Delta_{\mathrm{WT-C}}$
MNIST (IID)	***Dense (100%)*: WT Final** 95.98 ± 0.09, **WT TAA** 93.13 ± 0.06											
Grow	95.39 ± 0.13	92.94 ± 0.13	-1.16 ± 0.17	95.96 ± 0.07	93.71 ± 0.07	-0.57 ± 0.10	96.19 ± 0.08	94.12 ± 0.07	-0.38 ± 0.11	96.40 ± 0.07	94.40 ± 0.06	-0.22 ± 0.22
Prune	95.33 ± 0.12	92.87 ± 0.12	-0.63 ± 0.20	95.87 ± 0.14	93.66 ± 0.11	-0.58 ± 0.15	96.16 ± 0.13	94.05 ± 0.10	-0.62 ± 0.14	96.39 ± 0.09	94.39 ± 0.08	-0.60 ± 0.10
Split-MNIST (+ER)	***Dense (100%)*: WT Final** 84.36 ± 0.75, **WT TAA** 90.44 ± 0.77											
Grow	84.53 ± 0.77	90.19 ± 0.62	11.15 ± 1.66	84.47 ± 0.77	91.00 ± 0.52	11.93 ± 1.16	84.88 ± 1.09	91.30 ± 0.66	12.21 ± 1.54	85.81 ± 0.49	91.41 ± 0.75	10.85 ± 1.48
Prune	84.53 ± 0.72	90.33 ± 0.48	8.52 ± 2.14	85.14 ± 1.17	90.70 ± 0.84	7.92 ± 1.61	85.69 ± 0.71	90.98 ± 0.51	8.47 ± 1.51	85.20 ± 0.79	90.91 ± 0.36	7.29 ± 0.98
Split-Fashion (+ER)	***Dense (100%)*: WT Final** 75.01 ± 1.58, **WT TAA** 84.34 ± 1.53											
Grow	74.88 ± 1.07	83.34 ± 2.09	8.13 ± 1.07	74.96 ± 1.66	85.84 ± 1.34	7.21 ± 0.92	75.70 ± 1.11	86.02 ± 1.11	8.49 ± 1.02	75.11 ± 2.03	85.02 ± 2.16	7.19 ± 2.12
Prune	75.53 ± 1.57	86.32 ± 0.76	5.01 ± 0.92	75.70 ± 1.52	85.72 ± 1.67	5.74 ± 2.15	75.76 ± 1.08	84.08 ± 2.21	4.92 ± 0.97	75.00 ± 1.48	84.92 ± 3.04	5.71 ± 1.54

Table B.1: **Compactness-resolved winning-ticket performance for the small MLP benchmarks.** Entries are mean $\pm$ 95% CI over 10 seeds. For each compactness budget, we report Winning-Ticket final accuracy (**WT Final**), Winning-Ticket trajectory-average accuracy (**WT TAA**), and $\Delta_{\mathrm{WT-C}} = \mathrm{Final}_{\mathrm{WT}} - \mathrm{Final}_{\mathrm{C}}$. Negative $\Delta_{\mathrm{WT-C}}$ values indicate that the in-place structural-edit *cycle* reached a higher endpoint than the retrained ticket, while positive values indicate that the frozen mask retrained from scratch to a higher endpoint than the in-place structural-edit *cycle* procedure achieved. Dense is invariant to compactness and is therefore reported once per dataset header.

B.2 CIFAR-100 ConvNet: Additional Results and Ablations

Table B.2 makes the CIFAR-100 in-place–vs.–ticket dissociation explicit across compactness. Grow consistently attains higher in-place final ACC than Prune, yet their Winning-Ticket performance is statistically indistinguishable after retraining from scratch. As compactness increases, Grow shows increasingly negative ΔACC, indicating stronger dependence of its cycle endpoint on the adaptive path, whereas Prune remains near-zero or positive. Thus, the main separation lies in the structural-editing process rather than final retrainable architecture quality.

		Cycle Eval.		**Winning-ticket Eval.**		**Ticket − Cycle**	
Comp. (%)	**Method**	**ACC**	**TAA**	**ACC**	**TAA**	**ΔACC**	**ΔTAA**
100	**Dense**	**49.660±0.315**	**48.062±0.284**	**49.660±0.315**	**45.998±0.262**	0.000	−2.065
20	Grow	**52.316±0.317**	47.288±0.269	52.055±0.324	46.257±0.188	−0.261	−1.031
20	Prune	47.911±0.644	**48.519±0.348**	**52.057±0.283**	**46.298±0.162**	+4.146	−2.221
30	Grow	**53.312±0.247**	47.871±0.374	**51.626±0.232**	**46.909±0.220**	−1.686	−0.963
30	Prune	49.549±0.325	**48.732±0.358**	51.615±0.280	46.880±0.193	+2.066	−1.852
40	Grow	**53.588±0.313**	48.338±0.291	50.791±0.338	46.886±0.239	−2.797	−1.452
40	Prune	49.963±0.481	**48.869±0.251**	**51.205±0.415**	**47.195±0.234**	+1.242	−1.675
50	Grow	**54.338±0.221**	48.990±0.245	50.260±0.257	**46.776±0.174**	−4.078	−2.213
50	Prune	50.227±0.548	**48.997±0.240**	**50.320±0.415**	46.749±0.295	+0.093	−2.247

Table B.2: **CIFAR-100 (i.i.d.) ConvNet, ReLU, 200 epochs (SGD, η=0.1).** Mean $\pm$ CI95 over $n = 10$ seeds. We report *cycle* evaluation metrics (measured during in-place structural adaptation) and *winning-ticket* evaluation metrics (mask frozen, weights reinitialized, retrained from scratch). Deltas are computed *within method and compactness* as $\Delta = (\text{ticket} - \text{cycle})$ (%). Bold highlights the higher value between Grow and Prune within each compactness for each column.

Statistical comparisons. Welch two-sample tests shown in Table B.3 and Table B.4 confirm that the Grow–Prune difference is strongest during structural adaptation. In-place final ACC differs significantly at every compactness, while in-place final TAA differs at 20–40%. In contrast, neither Winning-Ticket ACC nor Winning-Ticket TAA differs significantly between the two methods at any compactness. Thus, the statistically reliable separation concerns the adaptive process rather than the retrainability of the final masks.

B.2.1 Cohort-level integration diagnostics

Structural plasticity changes *who participates* in the computation and *who receives credit* during optimization. Section 4.2 identifies a consistent asymmetry between structural growth and pruning: newly activated units can participate in the forward pass while remaining weakly integrated into the optimization process. The following ablations test whether this result depends on particular design choices, including the growth-selection heuristic, the allocation of the structural budget across layers, and the location at which structural edits are introduced.

Comparison	20%		30%		40%		50%	
	In-place	WT	In-place	WT	In-place	WT	In-place	WT
GROW vs DENSE	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0038**
GROW vs PRUNE	**0.0000**	0.9917	**0.0000**	0.9461	**0.0000**	0.0980	**0.0000**	0.7846
DENSE vs PRUNE	**0.0001**	**0.0000**	0.5862	**0.0000**	0.2514	**0.0000**	0.0615	**0.0108**

Table B.3: Welch two-sample t-test p-values for final accuracy (ACC), comparing methods across compactness levels for in-place structural adaptation and Winning-Ticket (WT) evaluations. Bold indicates statistical significance at $p < 0.05$.

Comparison	20%		30%		40%		50%	
	In-place	WT	In-place	WT	In-place	WT	In-place	WT
GROW vs DENSE	**0.0003**	0.0876	0.3701	**0.0000**	0.1425	**0.0000**	**0.0000**	**0.0000**
GROW vs PRUNE	**0.0000**	0.7154	**0.0014**	0.8277	**0.0059**	0.0517	0.9637	0.8601
DENSE vs PRUNE	**0.0340**	**0.0437**	**0.0040**	**0.0000**	**0.0001**	**0.0000**	**0.0000**	**0.0004**

Table B.4: Welch two-sample t-test p-values for TAA, comparing methods across compactness levels for in-place structural adaptation and Winning-Ticket (WT) evaluations. Bold indicates statistical significance at $p < 0.05$.

Cohorts and measurement snapshots. Both are computed from the same per-cycle cohort snapshots, but they answer different questions. Absolute diagnostics ask whether newborn, kept, or pruned units exhibit nontrivial forward participation and learning signal; parity diagnostics ask how those cohorts compare *relative* to one another.

For GROW, at cycle t we define the newborn cohort

$$\mathcal{N}_t = \{\text{units activated at the GROW step of cycle } t\},$$

and the incumbent cohort

$$\mathcal{O}_t = \{\text{units already active before that GROW step}\}.$$

For PRUNE, we define the kept cohort

$$\mathcal{K}_t = \{\text{units kept by the PRUNE decision at cycle } t\},$$

and the pruned cohort

$$\mathcal{P}_t = \{\text{units removed by the PRUNE decision at cycle } t\}.$$

We use three time-points: (i) *Post*, the immediate snapshot after a structural edit and before further training; (ii) *Exit*, the end-of-cycle checkpoint at which prune decisions are made; and (iii) *End*, the end of the subsequent training segment.

Absolute activation and gradient signals. For a layer ℓ and a unit j, we measure:

- **Activation rate** $\mathrm{act}(j)$: the fraction of post-activation values exceeding a small threshold τ on a single mini-batch.
- **Per-unit gradient magnitude** $\mathrm{grad}(j)$: the mean absolute pre-activation gradient $|\partial \mathcal{L}/\partial z_j|$ on a single mini-batch, where z_j denotes the unit pre-activation.

For convolutional layers, activation rates are averaged over (B, H, W); for linear layers, over B. We aggregate within cohort by averaging over units and seeds, and in the main-paper figures we additionally average across layers unless otherwise stated.

For GROW, we report newborn absolute vitality through the post-growth cohort means

$$A^{\mathrm{abs}}_{\mathrm{grow,act}}(t,\ell) = \mathbb{E}_{j\in\mathcal{N}_t(\ell)}[\mathrm{act}(j)], \qquad A^{\mathrm{abs}}_{\mathrm{grow,grad}}(t,\ell) = \mathbb{E}_{j\in\mathcal{N}_t(\ell)}[\mathrm{grad}(j)].$$

These quantities serve as sanity checks: they ask whether newborn units are active at birth and whether they receive nonzero learning signal. For PRUNE, we analogously report absolute contrasts such as $\mathbb{E}[\mathrm{act}]$ for kept and pruned cohorts at exit, together with post-prune changes in survivor activation.

Relative cohort signal and log-parity. Absolute activation and gradient magnitudes establish whether structural cohorts receive nontrivial forward and backward signal, but their scales vary with training time, loss magnitude, optimizer state, and representation maturity. Therefore, we compare cohorts measured at the same snapshot. For a focal cohort A, reference cohort B, and signal $x \in \{\mathrm{act}, \mathrm{grad}\}$, we define the relative cohort signal

$$R_x(A, B) = \frac{\mathbb{E}_{j \in A}[x(j)]}{\mathbb{E}_{j \in B}[x(j)] + \epsilon}. \tag{38}$$

In figures intended for direct interpretation, we report $100R_x$: 100% denotes equal per-unit signal, values below 100% indicate that the focal cohort receives less signal, and values above 100% indicate more. For analyses where a symmetric scale around parity is useful, we additionally use the log-ratio

$$\Delta_x(A, B) = \log R_x(A, B), \tag{39}$$

for which parity corresponds to $\Delta_x = 0$.

For GROW, we compare newborn units $N_t^{(\ell)}$ with previously active incumbents $O_t^{(\ell)}$ at the post-growth snapshot:

$$R_{\mathrm{grow},x}(t, \ell) = \frac{\mathbb{E}_{j \in N_t^{(\ell)}}[x(j)]}{\mathbb{E}_{j \in O_t^{(\ell)}}[x(j)] + \epsilon}. \tag{40}$$

When log-parity is required, we use $\Delta_{\mathrm{grow},x}(t, \ell) = \log R_{\mathrm{grow},x}(t, \ell)$.

For PRUNE, we compare retained units $K_t^{(\ell)}$ with units selected for removal $P_t^{(\ell)}$ using the pre-mask exit snapshot:

$$R_{\mathrm{prune},x}(t, \ell) = \frac{\mathbb{E}_{j \in K_t^{(\ell)}}[x(j)]}{\mathbb{E}_{j \in P_t^{(\ell)}}[x(j)] + \epsilon}. \tag{41}$$

The corresponding log-parity is $\Delta_{\mathrm{prune},x}(t, \ell) = \log R_{\mathrm{prune},x}(t, \ell)$. Using the pre-mask snapshot prevents the removed cohort from being assigned trivial zero activation and gradient after pruning.

Prune survivor stability. To measure how the retained cohort changes after pruning, we compare its immediate post-prune activity with its end-of-cycle activity:

$$R_{\mathrm{surv,act}} = \frac{\mathbb{E}_{j \in K_t}[\mathrm{act}_{\mathrm{post}}(j)]}{\mathbb{E}_{j \in K_t}[\mathrm{act}_{\mathrm{end}}(j)] + \epsilon}. \tag{42}$$

We report $100R_{\mathrm{surv,act}}$, so 100% denotes no change, whereas values below 100% indicate that survivor activity increases during the subsequent training segment.

Prune selection and survivor readjustment. Figure B.1 shows the complementary PRUNE dynamics. At the first event, retained units reach roughly 128–138% of the activity of removed units, but this separation approaches 100% across cycles, indicating increasingly less selective pruning. Survivors also readjust more strongly: by the final event, immediate post-prune activity falls to about 84% and 90% of end-of-cycle activity at 20% and 30% compactness, respectively. Thus, PRUNE initially removes clearly low-activity structure, while later edits become less selective and require greater compensatory adaptation from surviving units.

B.2.2 ROBUSTNESS TO THE GROW SELECTION HEURISTIC

The default GROW rule ranks dormant units by their pre-mask activation frequency. A natural concern is that this criterion may select units that respond strongly to the input but are poorly aligned with the loss gradient, thereby producing the observed combination of forward participation and weak backward credit. We test this explanation by replacing activation-based top-k selection with gradient-based top-k selection in the CIFAR-100 ConvNet, while holding the initialization, optimizer, training budget, compactness schedule, and neutral FC-budget allocation fixed.

Table B.5 shows no systematic performance advantage from gradient-based selection. Across compactness levels, in-place final ACC and in-place TAA remain nearly unchanged, while the winning-ticket results are comparable or slightly lower. Therefore, changing the selection signal improves neither the structural-edit trajectory nor the quality of the final retrainable mask.

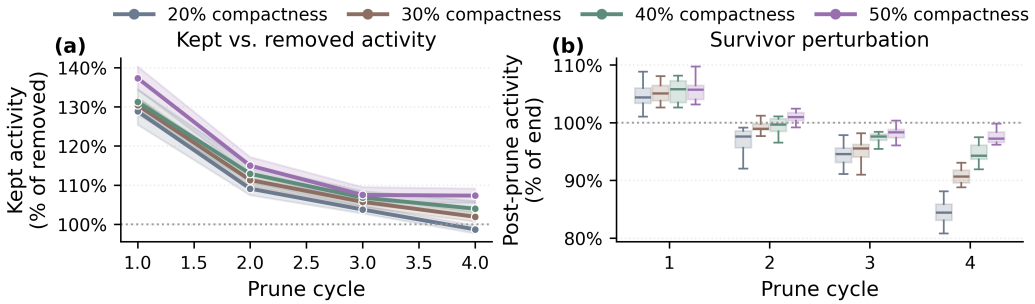

Figure B.1: **Pruning becomes less selective and increasingly perturbs surviving units across structural cycles. (a)** Activity of retained units relative to units selected for removal immediately before each pruning decision, expressed as a percentage of the removed cohort; 100% denotes equal activity. The decreasing separation across cycles shows that later pruning events have fewer clearly low-activity candidates. **(b)** Survivor readjustment, measured as post-prune activity relative to end-of-cycle activity; values below 100% indicate that surviving units increase their activity during the subsequent training segment. Curves, bands, and boxplots summarize means and 95% confidence intervals across runs.

Comp. (%)	GROW heuristic	In-place Final ACC	In-place Final TAA	WT ACC	WT TAA
20	ACTIVATION	52.32 ± 0.27	47.29 ± 0.23	52.05 ± 0.28	46.26 ± 0.16
20	GRADIENT	52.32 ± 0.15	47.38 ± 0.12	51.72 ± 0.26	46.20 ± 0.16
30	ACTIVATION	53.31 ± 0.21	47.87 ± 0.32	51.63 ± 0.20	46.91 ± 0.19
30	GRADIENT	53.22 ± 0.28	48.09 ± 0.16	51.67 ± 0.26	46.99 ± 0.12
40	ACTIVATION	53.59 ± 0.27	48.34 ± 0.25	50.79 ± 0.29	46.89 ± 0.21
40	GRADIENT	53.78 ± 0.26	48.35 ± 0.30	50.89 ± 0.31	46.93 ± 0.26
50	ACTIVATION	54.34 ± 0.19	48.99 ± 0.21	50.26 ± 0.22	46.78 ± 0.15
50	GRADIENT	54.14 ± 0.12	48.95 ± 0.18	50.12 ± 0.38	46.63 ± 0.16

Table B.5: **GROW heuristic ablation.** We compare activation-based and gradient-based top-k GROW under neutral allocation bias. This tests whether the newborn gradient bottleneck is simply caused by selecting growth locations using activation statistics. In-place structural adaptation metrics evaluate the adaptive structural trajectory; WT metrics freeze the final mask and retrain it from scratch. Values are mean $\pm$ CI95 over 20 seeds.

B.3 CONVNET FC-HEAD ALLOCATION-BIAS ABLATION

Because the main CIFAR-100 experiments use a neutral layer-allocation schedule, we test whether the observed GROW behavior depends on how compactness is distributed across the FC head. Table B.6 defines the neutral schedule and alternatives that mildly favor early, late, or both ends of the head. Table B.7 shows only moderate sensitivity: different schedules shift the trade-off between in-place performance and ticket retrainability, but none consistently improves both across compactness levels. Thus, modest FC-head budget redistribution does not remove the newborn-integration limitation.

Schedule	b_{FC1}	b_{FC2}	b_{FC3}
Neutral	1.0	1.0	1.0
FC1-Protect	1.5	1.5	0.6
FC3-Protect	0.6	0.6	1.5
Ends-Skewed	1.2	0.6	1.2

Table B.6: **Bias scalars used to distribute the global kept-weight budget across masked fully connected layers.** The neutral schedule allocates proportionally to layer weight mass, while the remaining schedules mildly favor early layers, late layers, or both ends of the head.

B.4 CIFAR-100 RESNET: ADDITIONAL LOCUS ANALYSES

The main text reports the central ResNet finding: the representation and future adaptability produced by structural growth depend strongly on where new channels are introduced. This section provides the complete compactness sweeps, event-local integration diagnostics, representation decompositions,

		In-place Eval.		Winning-ticket Eval.		Ticket − In-place	
Comp. (%)	**Method**	**ACC**	**TAA**	**ACC**	**TAA**	**ΔACC**	**ΔTAA**
20	GROW	52.316±0.317	47.288±0.269	**52.055±0.324**	**46.257±0.188**	−0.261	−1.031
20	GROW (FC3-Protect)	**53.016±0.393**	**47.717±0.461**	51.287±0.314	46.085±0.175	−1.729	−1.632
20	GROW (Ends-Skewed)	51.926±0.153	46.987±0.196	51.757±0.259	45.758±0.190	−0.169	−1.230
20	GROW (FC1-Protect)	51.794±0.201	46.830±0.247	51.622±0.276	45.721±0.140	−0.172	−1.109
30	GROW	53.312±0.247	47.871±0.374	51.626±0.232	**46.909±0.220**	−1.686	−0.963
30	GROW (FC3-Protect)	**53.564±0.345**	**48.235±0.284**	50.232±0.290	46.295±0.161	−3.332	−1.940
30	GROW (Ends-Skewed)	52.819±0.220	47.650±0.326	**52.076±0.257**	46.876±0.189	−0.743	−0.775
30	GROW (FC1-Protect)	52.647±0.395	47.469±0.172	51.879±0.303	46.648±0.194	−0.768	−0.821
40	GROW	53.588±0.313	48.338±0.291	50.791±0.338	46.886±0.239	−2.797	−1.452
40	GROW (FC3-Protect)	**53.761±0.283**	**48.424±0.257**	49.550±0.369	46.105±0.219	−4.211	−2.319
40	GROW (Ends-Skewed)	53.409±0.257	48.146±0.389	**51.670±0.311**	**47.129±0.231**	−1.739	−1.017
40	GROW (FC1-Protect)	53.305±0.258	48.191±0.101	51.263±0.359	46.795±0.187	−2.042	−1.396
50	GROW	**54.338±0.221**	**48.990±0.245**	50.260±0.257	46.776±0.174	−4.078	−2.213
50	GROW (FC3-Protect)	54.206±0.259	48.941±0.230	49.415±0.258	46.293±0.165	−4.791	−2.647
50	GROW (Ends-Skewed)	53.759±0.225	48.272±0.298	**50.983±0.337**	**47.057±0.253**	−2.776	−1.215
50	GROW (FC1-Protect)	53.791±0.264	48.340±0.246	50.884±0.258	46.887±0.210	−2.907	−1.453

Table B.7: **CIFAR-100 (i.i.d.) ConvNet, ReLU, 200 epochs (SGD, η=0.1).** Mean ± CI95 over $n = 10$ seeds. We report *In-place* evaluation metrics (measured during structural adaptation) and *winning-ticket* evaluation metrics (mask frozen, weights reinitialized, retrained from scratch). Deltas are computed *within method and compactness* as $\Delta = (\text{ticket} - \text{In-place})$ (%). Bold highlights the highest value among the GROW bias schedules within each compactness for each column.

and future-probe robustness analyses underlying that result. Architecture, edit-locus, and compactness definitions are given in Apps. A.4.3 and A.5.

B.4.1 CROSS-METRIC SUMMARY

Table B.8 collects the current-task, representation, and future-adaptation quantities reported in the main text. The table highlights the progressive separation between head-localized and residual-hierarchy growth: broader edit loci reduce in-place final ACC, effective feature support, and the final future-adaptation endpoint, even when final winning-ticket accuracy remains comparatively close to PRUNE.

Method	Locus	In-place ACC	WT ACC	Total variance	Norm. eff. rank	Linear probe	Future AUC	Final future accuracy
DENSE	–	63.1 ± 0.3	63.1 ± 0.3	35.4 ± 0.4	0.588 ± 0.002	55.1 ± 0.2	35.2 ± 0.7	51.3 ± 1.0
PRUNE	Head	61.9 ± 0.3	61.3 ± 0.4	30.8 ± 1.0	0.603 ± 0.004	56.2 ± 0.3	33.6 ± 0.6	50.6 ± 0.9
GROW	Head	61.2 ± 0.3	61.7 ± 0.4	53.6 ± 0.4	0.516 ± 0.002	57.1 ± 0.1	36.5 ± 0.7	50.7 ± 0.6
PRUNE	Late	60.4 ± 0.2	60.5 ± 0.3	13.3 ± 2.0	0.514 ± 0.009	52.1 ± 0.3	32.5 ± 0.9	48.5 ± 0.7
GROW	Late	61.4 ± 0.2	60.2 ± 0.2	23.2 ± 0.1	0.416 ± 0.002	50.4 ± 0.2	32.9 ± 0.4	46.6 ± 0.5
PRUNE	Mid+late	59.2 ± 0.4	59.5 ± 0.4	11.6 ± 1.7	0.510 ± 0.010	50.9 ± 0.4	31.9 ± 0.7	47.9 ± 0.5
GROW	Mid+late	59.5 ± 0.3	59.7 ± 0.3	19.1 ± 0.1	0.400 ± 0.002	47.0 ± 0.2	29.5 ± 0.5	43.3 ± 0.5
PRUNE	All	58.0 ± 0.2	59.0 ± 0.3	11.3 ± 1.6	0.509 ± 0.011	49.7 ± 0.5	30.9 ± 1.0	46.2 ± 0.6
GROW	All	58.5 ± 0.2	59.2 ± 0.4	17.8 ± 0.1	0.403 ± 0.002	45.2 ± 0.2	28.4 ± 0.4	41.9 ± 0.5

Table B.8: **ResNet current-task performance, representation state, and bounded future reachability at 50% final compactness.** Representation scale and effective rank are reported descriptively and are not ranked. Values are means ± 95% confidence intervals. Dense is locus-agnostic and is reported once.

B.4.2 PRUNING PROTOCOL AND CAPACITY-PATH CONTROL.

The main GROW–PRUNE comparison contrasts opposite structural operators, but two procedural asymmetries can affect In-place TAA: the default pruning baseline uses IMP-style rewinding whereas GROW updates in place, and pruning descends from a fully active head while growth ascends from a sparse seed. Therefore, we repeat the `CIFAR-100` ResNet HEAD comparison over the 20–30–40–50% compactness sweep and add PRUNE-FT, which follows the same pruning schedule without

rewinding surviving weights. We report in-place TAA (Cycle-TAA) together with in-place final ACC at the matched target compactness.

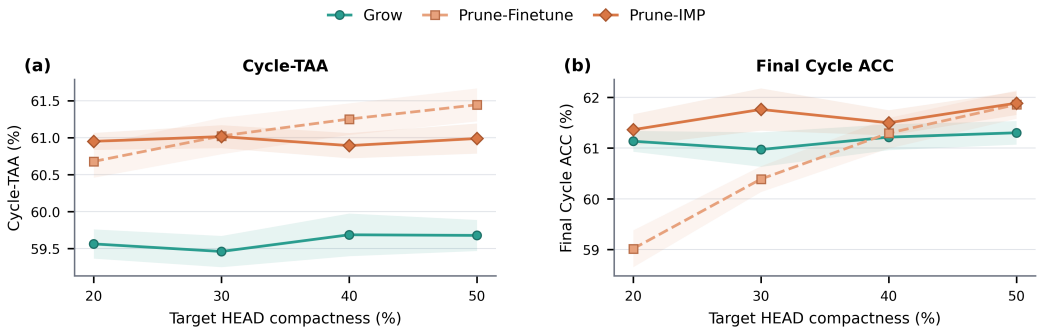

Figure B.2: **Pruning-protocol control across ResNet HEAD compactness.** (a) Cycle-TAA across target compactness for GROW, non-rewind PRUNE-FT, and rewind-based PRUNE-IMP. Because Cycle-TAA averages over the complete structural trajectory, it also reflects the greater active capacity available to pruning before the final target is reached. (b) Final Cycle ACC after each method reaches the same target HEAD compactness. Points show means across experiment indices and shaded bands show 95% confidence intervals. PRUNE-FT removes the IMP reset while preserving the pruning schedule, separating reset-protocol effects from the capacity-path asymmetry.

Results and interpretation. At 50% compactness, removing rewind changes In-place TAA by only +0.46 points and In-place Final ACC by -0.03 points. GROW remains lower in In-place TAA by 1.77/1.31 points relative to PRUNE-FT/IMP, but only 0.56/0.58 points lower at the final cycle. Mean active HEAD capacity over the trajectory is 38.2% for GROW versus 65.0% for both pruning variants. Thus, In-place TAA reflects unequal capacity exposure as well as editing dynamics; PRUNE-FT isolates the rewind convention, while In-place Final ACC and WT-ACC provide matched-target endpoint controls. We retain IMP as the canonical lottery-ticket baseline.

B.4.3 EVENT-LOCAL INTEGRATION DYNAMICS

Table B.9 reports partial Spearman correlations between newborn integration and subsequent representation quality, controlling for final compactness. Associations are strongly locus dependent. At HEAD, stronger newborn activation is associated with greater class separation ($\rho_{\text{partial}} = 0.378$) and lower normalized effective rank ($\rho_{\text{partial}} = -0.480$), with weaker but similar trends for gradient signal. Under MID+LATE, stronger gradient signal likewise tracks greater class separation and lower effective rank, while relationships are weak or absent for LATE and ALL. Neither activation nor gradient integration reliably predicts linear-probe accuracy at any locus. Thus, newborn integration relates more consistently to representation geometry than to downstream feature utility: it constrains how growth is absorbed, but does not by itself determine the usefulness of the resulting representation.

B.4.4 REPRESENTATION ROBUSTNESS ACROSS COMPACTNESS AND STAGES

We focus on final-cycle `stage3_out` at 50% compactness in the main text, where the tradeoff between current utility and bounded future adaptability is clearest. Here we test its robustness across compactness and depth using the same scale–support–utility diagnostics: (1) a compactness sweep at `stage3_out`, and (2) a 50% stagewise analysis across `stage1_out`, `stage2_out`, `stage3_out`, and `prelogit`. Figures B.3 and B.4 show that the main pattern persists across compactness and emerges most strongly in later task-facing stages.

The compactness sweep confirms that the main-text 50% result is not isolated. Under PRUNE, HEAD remains closest to DENSE in support and utility, while broader residual loci reduce variance and linear-probe accuracy but retain relatively high effective rank. Under GROW, HEAD consistently increases feature scale and probe accuracy while concentrating variance into fewer directions, whereas residual-hierarchy growth yields lower effective rank and linear utility than PRUNE, most strongly under ALL. Thus, the contrast between head-localized and deep feature-hierarchy growth persists across compactness levels.

INTEGRATION SIGNAL	REPRESENTATION OUTCOME	LOCUS	PARTIAL SPEARMAN ρ	p
Activation parity	*Class separation*	`Head`	0.378	<.001
		`Late`	0.084	.120
		`Mid+Late`	-0.007	.895
		`All`	-0.013	.810
	Normalized effective rank	`Head`	-0.480	<.001
		`Late`	0.002	.968
		`Mid+Late`	0.037	.506
		`All`	0.053	.331
	Linear probe accuracy (%)	`Head`	-0.083	.127
		`Late`	0.009	.867
		`Mid+Late`	-0.044	.425
		`All`	-0.040	.463
Gradient parity	*Class separation*	`Head`	0.184	<.001
		`Late`	-0.050	.360
		`Mid+Late`	0.126	.022
		`All`	-0.050	.359
	Normalized effective rank	`Head`	-0.191	<.001
		`Late`	0.020	.714
		`Mid+Late`	-0.286	<.001
		`All`	0.042	.441
	Linear probe accuracy (%)	`Head`	-0.040	.464
		`Late`	-0.015	.790
		`Mid+Late`	-0.063	.256
		`All`	-0.086	.115

Table B.9: **Partial Spearman correlations between newborn integration and representation quality.** Integration statistics are first averaged across editable growth layers within each event, and then, over structural cycles within each replicate and compactness, and correlations control for final compactness. For newborn-log diagnostics, integration at birth in cycle c is aligned with representation diagnostics at the subsequent checkpoint in cycle $c + 1$. Reported p-values are descriptive and are not corrected for multiple comparisons.

The stagewise view clarifies how the final `stage3_out` differences emerge. Under PRUNE, HEAD stays close to DENSE, while deeper loci reduce scale and linear utility but largely recover effective support by `stage3_out`. Under GROW, HEAD diverges mainly at later readouts, where scale and probe accuracy increase sharply, whereas residual growth degrades representations earlier and the deficit persists downstream. MID+LATE and ALL already lose support by `stage2_out` and remain below DENSE in effective rank and linear-probe accuracy at `stage3_out` and `prelogit`. Thus, broad growth yields representations that are not only smaller in scale but also less supported and less linearly useful.

B.4.5 INTERVENTIONS DECOUPLE CURRENT PERFORMANCE FROM FUTURE ADAPTABILITY.

Figure B.5 shows that the interventions do not improve newborn integration, current performance, and future adaptability in lockstep. Across ConvNet interventions and compactness levels, early newborn gradient signal is not positively associated with in-place ACC (Spearman $\rho = -0.62$), so better local backward integration does not by itself predict a better current endpoint. The same decoupling appears in ResNet at 50% compactness. Under HEAD, vanilla GROW reaches about 58% in-place ACC and most growth variants remain within roughly one point of it, whereas GROW+RSL drops to about 54%. Under ALL, the pattern is similar: most variants stay near vanilla GROW, while GROW+RSL is again lower in current accuracy. Future adaptability follows a different ordering. Under HEAD, GROW+RSL reaches about 42 AUC, above vanilla GROW (~ 38), PRUNE (~ 36), and the other growth variants. Under ALL, it remains the strongest growth variant at about 37 AUC, compared with roughly 29–30 for vanilla GROW. Thus, interventions can change the optimization state left by growth—especially its later adaptability—without improving the immediate in-place endpoint.

B.4.6 INTERVENTIONS AT INTERMEDIATE EDIT LOCI.

To bridge the HEAD and ALL cases from the main text, we repeat the intervention comparison under LATE and MID+LATE, reporting in-place final ACC and future-reachability AUC. Figure B.6 extends the same decoupling trend. Current performance changes little across most interventions:

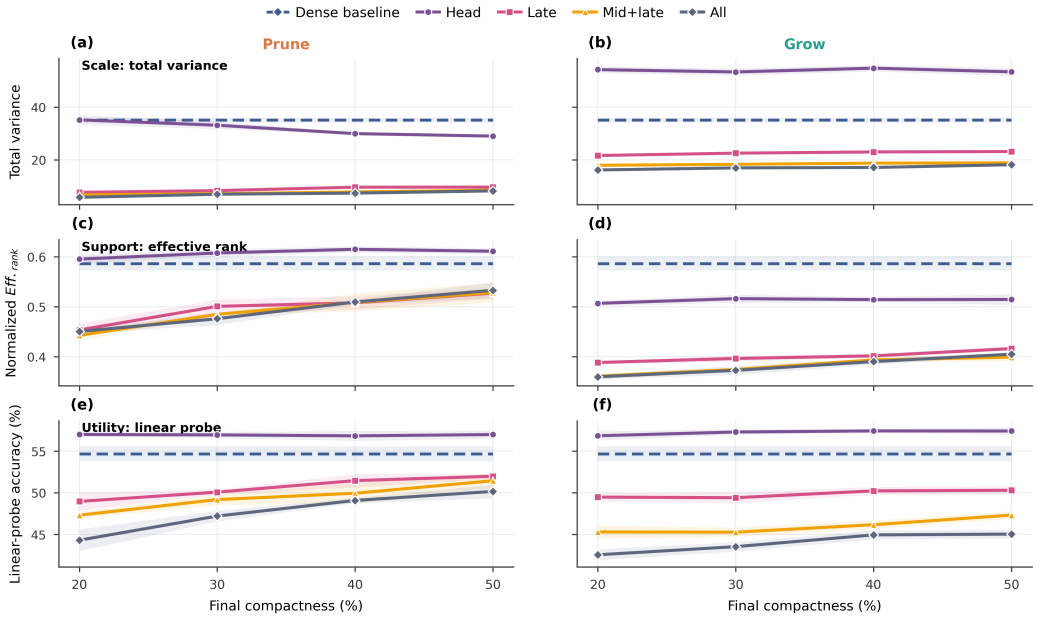

Figure B.3: **Representation robustness across compactness.** Final-cycle representation diagnostics at `stage3_out` for 20–50% compactness. Rows report feature scale (total covariance variance), normalized effective rank, and linear-probe accuracy; columns compare PRUNE and GROW. The dashed line denotes the Dense baseline, which is locus-agnostic. Across compactness values, head-localized growth increases feature scale and probe accuracy while reducing effective support, whereas growth within the residual hierarchy preserves more variance than PRUNE but yields lower effective rank and lower linear readability. Error bands show 95% confidence intervals.

under LATE, vanilla GROW reaches about 58% in-place ACC and the other growth variants remain close to it, while GROW+RSL is lower at roughly 54%. Under MID+LATE, the pattern is similar: most growth variants cluster near 57%, whereas GROW+RSL remains lower at about 53%. Future adaptability differs more strongly. GROW+RSL raises AUC from about 34 to 40 under LATE, and from about 31 to 37–38 under MID+LATE, while the remaining interventions stay closer to vanilla GROW or below it. Together with the HEAD/ALL results, this shows that activation-level trainability can improve bounded future adaptability without improving the immediate endpoint of structural growth.

B.4.7 CLASS-SEPARATION DECOMPOSITION

The main text uses total covariance variance to measure the overall scale of the learned representation, but this quantity does not reveal whether that variation separates classes or instead reflects dispersion within classes. To distinguish these possibilities, we decompose the feature geometry into between- and within-class scatter. Viewing each class as a cloud in feature space, between-class scatter measures the distance among class centers, whereas within-class scatter measures the spread of examples around their own center. We evaluate both components at the final cycle and 50% compactness:

$$S_B = \sum_{c=1}^{C} n_c(\mu_c - \mu)(\mu_c - \mu)^\top, \tag{43}$$

$$S_W = \sum_{c=1}^{C} \sum_{i:y_i=c} (h_i - \mu_c)(h_i - \mu_c)^\top, \tag{44}$$

and report the per-dimension traces $\mathrm{tr}(S_B)/d$ and $\mathrm{tr}(S_W)/d$, together with

$$\mathrm{Sep}(H) = \mathrm{tr}(S_B)/(\mathrm{tr}(S_W) + \epsilon) \tag{45}$$

Figure B.7 shows that similar separation ratios can reflect different geometries. Under HEAD, GROW increases both between- and within-class scatter relative to DENSE, with the larger between-

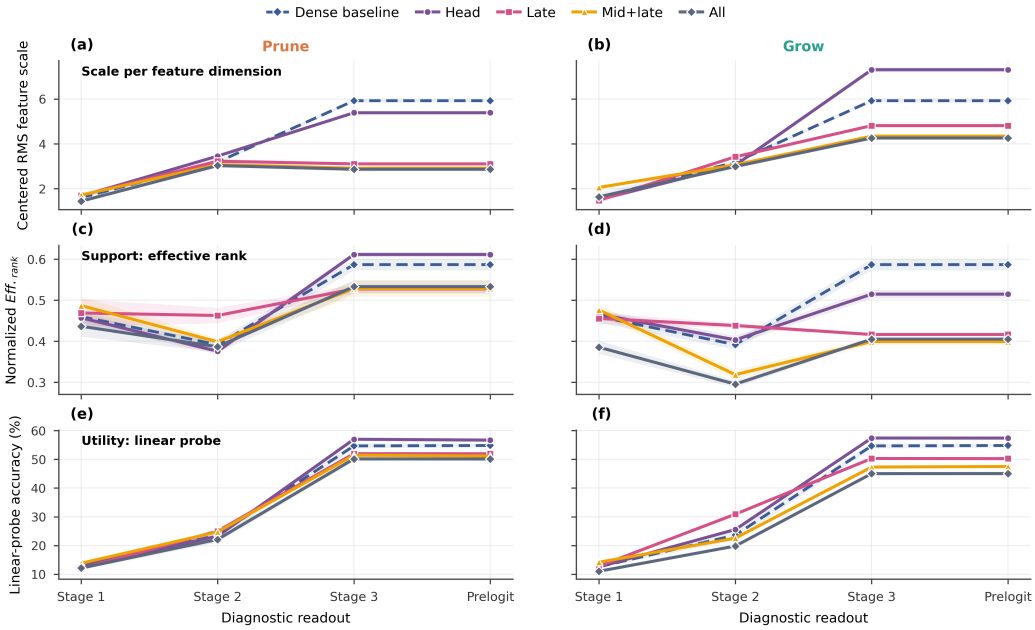

Figure B.4: **Stagewise representation diagnostics at 50% compactness.** Final-cycle diagnostics across `stage1_out`, `stage2_out`, `stage3_out`, and `prelogit`. The dashed line denotes the Dense baseline. Head-localized edits remain close to Dense through the residual hierarchy and diverge mainly at the later readouts, whereas residual-hierarchy edits produce progressively weaker support and utility by `stage3_out` and `prelogit`, especially for GROW. Shaded regions show 95% confidence intervals.

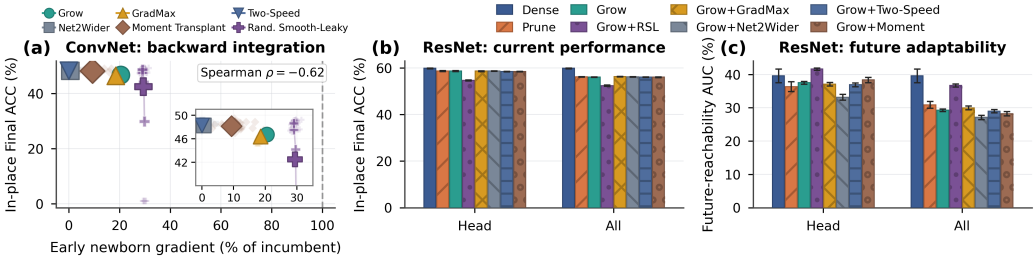

Figure B.5: **Interventions decouple newborn integration, current performance, and future adaptability.** **(a)** Across ConvNet interventions and compactness levels, stronger early newborn gradient signal is not associated with higher in-place ACC (Spearman $\rho = -0.62$); 100% denotes equal backward signal. Large markers denote method means and lighter points individual runs. **(b)** On ResNet, most interventions produce only modest changes in in-place ACC under HEAD and ALL; RAND. SMOOTH-LEAKY improves newborn trainability despite lower current accuracy. **(c)** Future-reachability AUC follows a different ordering: RAND. SMOOTH-LEAKY is the strongest growth variant at both loci, particularly under ALL, despite weaker in-place ACC. Bars show means with 95% confidence intervals.

class increase yielding the highest separation ratio. Residual-hierarchy edits instead reduce both components; GROW can retain a high ratio partly because within-class spread also contracts, without corresponding gains in effective rank or linear-probe accuracy. Thus, head-localized growth amplifies task-aligned class structure, whereas broad growth weakens the feature hierarchy; PRUNE preserves mature pathways but compresses the representation.

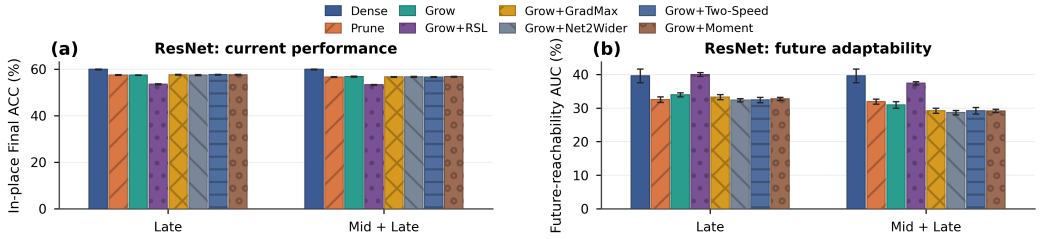

Figure B.6: **Intervention effects persist at intermediate ResNet edit loci.** Results are shown at 50% compactness for edits confined to the final residual stage (LATE) or the final two stages (MID+LATE). **(a)** Most interventions produce only modest changes in in-place final ACC, while RAND. SMOOTH-LEAKY lowers current accuracy relative to vanilla GROW. **(b)** The ordering changes for bounded future adaptation: GROW+RSL achieves the highest future-reachability AUC among the growth variants at both loci, approximately matching DENSE under LATE and substantially narrowing the gap under MID+LATE. The remaining interventions provide smaller or less consistent gains over vanilla GROW. Bars show means with 95% confidence intervals.

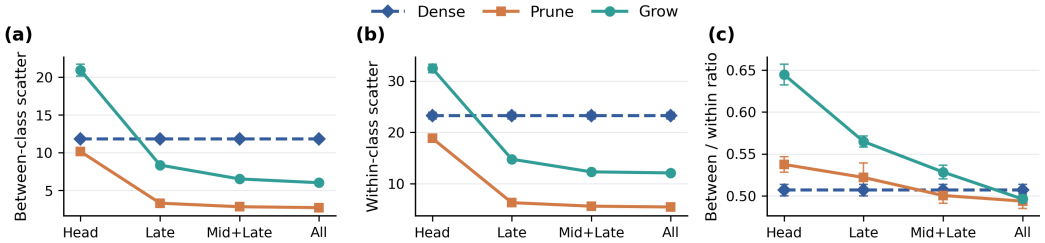

Figure B.7: **Decomposition of class-separation geometry.** Final-cycle `stage3_out` features at 50% compactness. **(a)** Between-class scatter per feature dimension. **(b)** Within-class scatter per feature dimension. **(c)** Their ratio. HEAD-localized GROW expands both components, with a larger proportional increase in between-class scatter. Residual-hierarchy edits instead contract the feature geometry, and apparently favorable separation ratios can partly result from reduced within-class spread rather than stronger absolute class structure. The dashed line denotes the locus-agnostic DENSE baseline; error bars show 95% confidence intervals.

B.4.8 SENSITIVITY TO GROWTH-EVENT FREQUENCY

We vary growth frequency while fixing the total structural-training budget at 200 epochs, comparing $K \in \{5, 10, 20\}$ events with 40, 20, and 10 epochs between edits, respectively.

Figure B.8 shows that more frequent growth lowers in-place TAA (Cycle-TAA in the figure), increasingly so as edits extend through the residual hierarchy. The effect is modest for HEAD and LATE, but stronger for MID+LATE and ALL. Newborn gradient signal shows the corresponding integration pressure: head-level newborns approach incumbent levels with more frequent edits, whereas residual-channel newborns become increasingly disadvantaged at lower compactness. Thus, frequent growth is most disruptive when edits are broad and channel budgets are small, indicating that residual insertion requires sufficient optimization time to integrate.

B.4.9 BOUNDED FUTURE REACHABILITY

As in Sec. 4.3, we clone each checkpoint and fully fine-tune it under the CIFAR-100 class-permutation remapping probe, preserving inputs and visual classes while changing their output assignments. We summarize bounded adaptation with normalized AUC over budget B and final probe accuracy. We select the probe protocol independently of the structural-training optimizer by sweeping $\{\text{SGD}, \text{ADAM}, \text{ADAMW}\}$, learning rates $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$, and $B \in \{600, 1200, 2400, 4800\}$. ADAMW with $\eta = 10^{-3}$, $B = 2400$, and batch size 256 yields the most stable and discriminative trajectories and is used throughout.

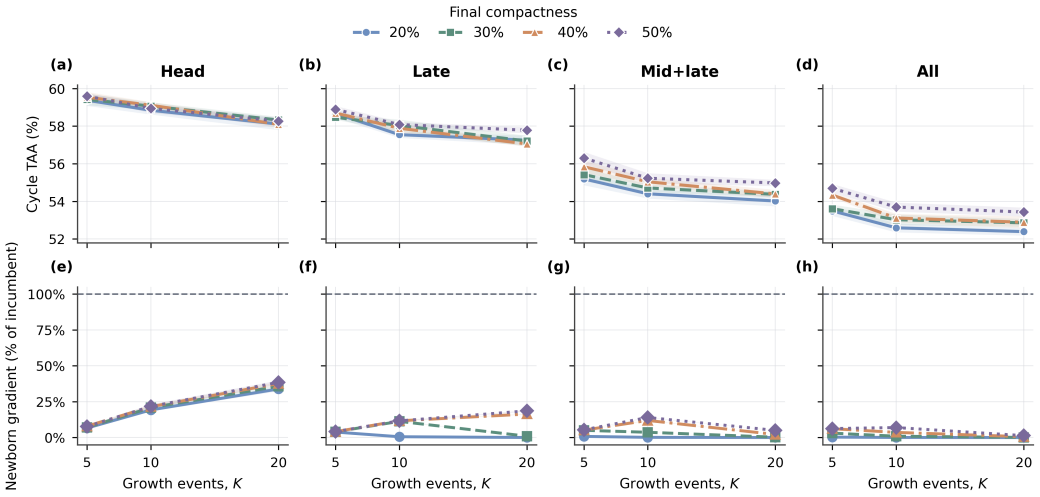

Figure B.8: **ResNet growth-frequency stress test.** We vary the number of growth events $K \in \{5, 10, 20\}$ while fixing the total structural-training budget at 200 epochs. Columns correspond to the HEAD, LATE, MID+LATE, and ALL edit loci; curves denote final compactness targets. **Top:** Cycle-TAA measures performance throughout the repeated structural-edit trajectory. **Bottom:** newborn gradient signal relative to incumbent units at birth, reported as a percentage of incumbent signal; 100% denotes equal backward credit. Bands show 95% confidence intervals

Intermediate-loci adaptation trajectories. Figure B.9 completes the edit-locus comparison at 50% compactness using the same class-permutation shift and 2400-step fine-tuning budget. Under LATE, GROW adapts faster than PRUNE initially, but the trajectories converge and it finishes slightly lower (47.2% vs. 48.1%). Under MID+LATE, this early advantage weakens further and GROW finishes at 44.8%, compared with 48.1% for PRUNE and 51.3% for DENSE (Table B.8). Thus, broader feature-hierarchy growth retains some early mobility but reaches a progressively lower endpoint within the fixed adaptation budget.

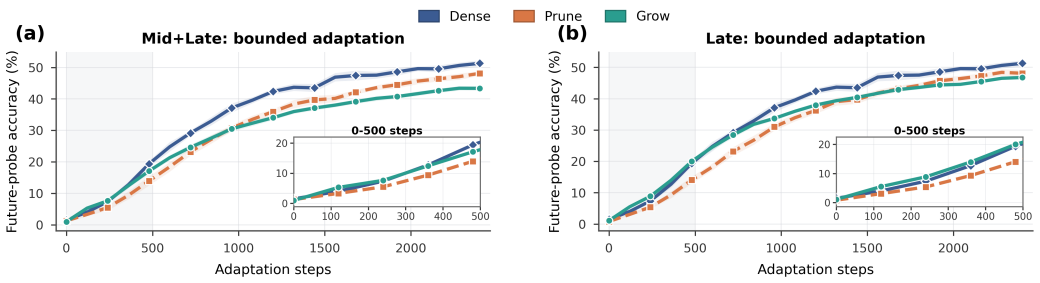

Figure B.9: **Bounded adaptation at intermediate ResNet edit loci.** Under both LATE and MID+LATE editing, GROW initially matches or exceeds PRUNE, but PRUNE subsequently catches up and reaches a higher endpoint. The separation is larger under MID+LATE, while DENSE retains the strongest overall trajectory. Insets magnify the first 500 adaptation steps; shaded regions show 95% confidence intervals.

C ACTIVATION-CONTROL BENCHMARK ANALYSIS

The activation-function intervention in Sec. 5 targets a specific mechanistic question: does improving activation-level trainability reduce the newborn integration disadvantage of growth, rather than just boosting performance via changes in insertion, selection, or optimizer-state handling? In the main text, Rand. Smooth-Leaky was introduced as exactly this intervention, since it modifies the gradient pathway by which new units integrate into a mature network while leaving structural adaptation unchanged.

Protocol For each benchmark and method, we apply the same procedure with two activations: standard ReLU and Rand. Smooth-Leaky (RSL). We report the signed difference

$$\Delta \;=\; \mathrm{RSL} - \mathrm{ReLU},$$

so positive values mean RSL performs better. The comparison is done separately for each method, benchmark, and compactness level, then aggregated over the compactness levels shown. To isolate activation effects from learning-rate choice, we first select the best learning rate independently for each activation, then compute Δ. The full suite spans seven datasets: the five repeated-shift plasticity benchmarks, Split-CIFAR100 as the sequential-accumulation continual-learning setting, and the i.i.d. CIFAR-100 control.

Rand. Smooth-Leaky hyperparameters. For the continual-learning plasticity benchmarks (Fig. 5), we used the default Rand. Smooth-Leaky hyperparameters from Lillo & Cheney (2025). For Split-CIFAR100 and the i.i.d. CIFAR-100 control, we instead ran a dedicated sweep over the Rand. Smooth-Leaky shape parameters, varying c and p over

$$\{0.1,\ 0.3,\ 0.5,\ 0.8,\ 1,\ 2,\ 3,\ 4,\ 5\},$$

and combining them with the following lower/upper-bound pairs:

$$(0.01, 0.05),\ (0.01, 1.00),\ (0.125, 0.333),\ (0.40, 1.00),\ (0.50, 1.00),$$
$$(0.60, 0.80),\ (0.60, 1.00),\ (0.70, 1.00),\ (0.673, 2.673).$$

Sweeps for Split-CIFAR100 and i.i.d. CIFAR-100 were performed under the DENSE configuration. Then, we fixed the best-performing Rand. Smooth-Leaky configuration found under DENSE for both GROW and PRUNE avoiding any type of method-specific over-tuning and ensuring that differences reflect the structural operators themselves. Table C.1 summarizes the Rand. Smooth-Leaky settings used in each benchmark.

Benchmark	c	p	Lower	Upper	LR
Permuted MNIST	0.8	1.0	0.3	0.6	0.001
Random-Label MNIST	2.0	0.8	0.3	0.6	0.001
Random-Label CIFAR	0.8	3.0	0.5	0.5	0.001
5+1 CIFAR	0.5	0.5	0.673	2.673	0.001
Continual ImageNet	0.5	0.5	0.3	0.3	0.001
Split-CIFAR100	0.3	0.1	0.673	2.673	0.001
CIFAR-100 (i.i.d.)	0.1	0.1	0.673	2.673	0.001

Table C.1: **Rand. Smooth-Leaky hyperparameters used in the activation-control analysis.** For the main continual-learning plasticity benchmarks, settings are inherited from Lillo & Cheney (2025). For Split-CIFAR100 and the i.i.d. controls, we ran a dedicated sweep over c, p, and lower/upper bounds; Split-CIFAR100 and i.i.d. CIFAR-100 report the selected best configuration.

Results and interpretation Table C.2 shows that Rand. Smooth-Leaky is a regime-sensitive rather than uniformly stronger activation. Its largest gains are often concentrated in GROW, including +35.14 on `5+1 CIFAR`, +11.91 on `Continual ImageNet`, +81.64 on `Random-Label CIFAR`, +61.38 on `Random-Label MNIST`, and +12.42 on `Split-CIFAR100`. On `Split-CIFAR100`, gains also persist in final accuracy (+14.56) and early-task adaptation (+4.10). RSL does not rescue every regime: on i.i.d. `CIFAR-100`, average accuracy declines for all methods. Overall, these results suggest that RSL is most beneficial when newly available capacity must become useful quickly, rather than providing a generic activation advantage.

C.1 RELU VS. RAND. SMOOTH-LEAKY NEWBORN INTEGRATION

As we just discussed, Rand. Smooth-Leaky often improves GROW in regimes where rapid adaptation is difficult. Next, we ask whether this performance gain is accompanied by the mechanistic consequences predicted by our main analysis: improved newborn integration after a growth event. To test this, we compare ReLU and Rand. Smooth-Leaky in the controlled CIFAR-100 GROW setting using the same newborn-to-incumbent signal diagnostics introduced in Sec. 4.2.

Benchmark / locus	Method	Δ Avg. ACC.	Δ Final ACC.	Δ Early TAA
`5+1 CIFAR`	DENSE	+29.61	+2.08	**+12.99**
	PRUNE	+3.60	+0.03	-1.20
	GROW	**+35.14**	**+3.32**	+4.38
`CIFAR-100` (Head)	DENSE	**-3.03**	**-0.92**	-3.03
	PRUNE	-11.30	-8.43	-11.30
	GROW	-4.99	-1.00	**-0.62**
`Continual ImageNet`	DENSE	-2.18	-0.13	**+1.66**
	PRUNE	+2.39	+0.08	+0.00
	GROW	**+11.91**	**+0.10**	+0.04
`Permuted MNIST`	DENSE	**+5.43**	-2.91	-0.99
	PRUNE	-0.74	**-0.59**	**-0.44**
	GROW	+4.79	-1.49	-2.61
`Random-Label CIFAR`	DENSE	+67.30	+1.75	+3.43
	PRUNE	+1.41	-0.04	+0.04
	GROW	**+81.64**	**+1.79**	**+6.76**
`Random-Label MNIST`	DENSE	+57.07	**+1.72**	+4.07
	PRUNE	-2.30	-0.01	-0.06
	GROW	**+61.38**	+1.67	**+6.27**
`Split-CIFAR100` (Head)	DENSE	+10.76	+11.63	-0.63
	PRUNE	+5.89	+5.75	+3.49
	GROW	**+12.42**	**+14.56**	**+4.10**

Table C.2: **Effect of Randomized Smooth-Leaky relative to ReLU.** Entries report $\Delta = \mathrm{RSL} - \mathrm{ReLU}$, averaged over compactness after learning-rate selection within each activation. Positive values favor RSL; bold denotes the largest delta within each benchmark and metric. `CIFAR-100` and `Split-CIFAR100` use the ResNet Head locus.

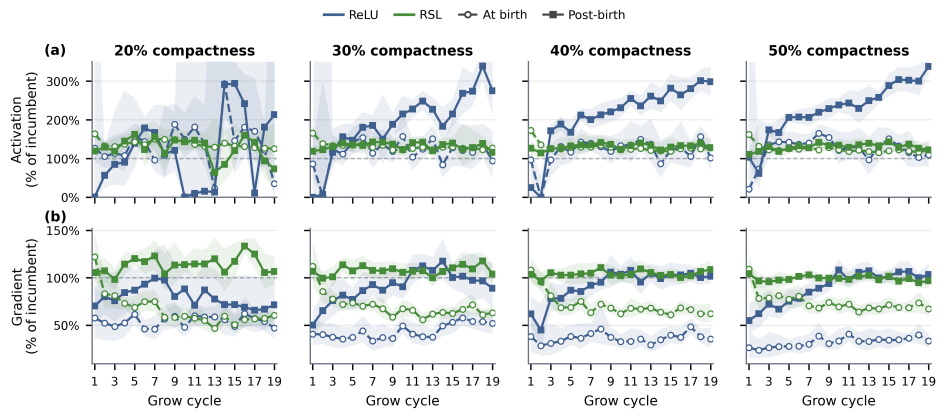

Figure C.1: **Rand. Smooth-Leaky improves newborn integration in Split-CIFAR100 Head GROW.** Newborn activation (a) and gradient signal (b) are reported relative to previously active units across compactness and growth cycles; 100% denotes equal per-unit signal. Under ReLU, newborn integration is less stable, with large early activation deviations and substantially weaker birth-time gradients. Rand. Smooth-Leaky keeps forward participation closer to the incumbent level and increases the gradient signal available to newborn units, especially at birth.

Interpretation Figure C.1 supports the activation-intervention account much more clearly in Split-CIFAR100 than in i.i.d. CIFAR-100. Rand. Smooth-Leaky keeps newborn units better coupled to the forward computation and, more importantly, increases their backward signal relative to incumbent units, especially at birth. Under ReLU, newborn activation is more variable and birth-time gradients remain substantially below the incumbent level. This matches the main-text result: in a plasticity-stressing regime, improving activation design helps GROW by making newly added units easier to integrate into ongoing learning.

D EARLY-TASK PLASTICITY UNDER REPEATED SHIFT

We additionally report *Early Task TAA*, the average accuracy over the initial portion of each task, measuring immediate post-shift plasticity before substantial within-task adaptation. Figure D.1 shows a clear dependence on adaptation time. Under rapid shifts, especially Permuted MNIST and Continual ImageNet, DENSE retains a strong early advantage, consistent with mature capacity being immediately usable while newborn units still integrate. On Random-Label MNIST and CIFAR, GROW+RAND. SMOOTH-LEAKY substantially improves over vanilla GROW and is the strongest growth variant, while other interventions provide smaller or less consistent gains (App. C). Under the longer optimization intervals of Split-CIFAR100, growth becomes more competitive, particularly with RAND. SMOOTH-LEAKY. Overall, growth depends critically on whether newborn units become usable before the learning signal changes again.

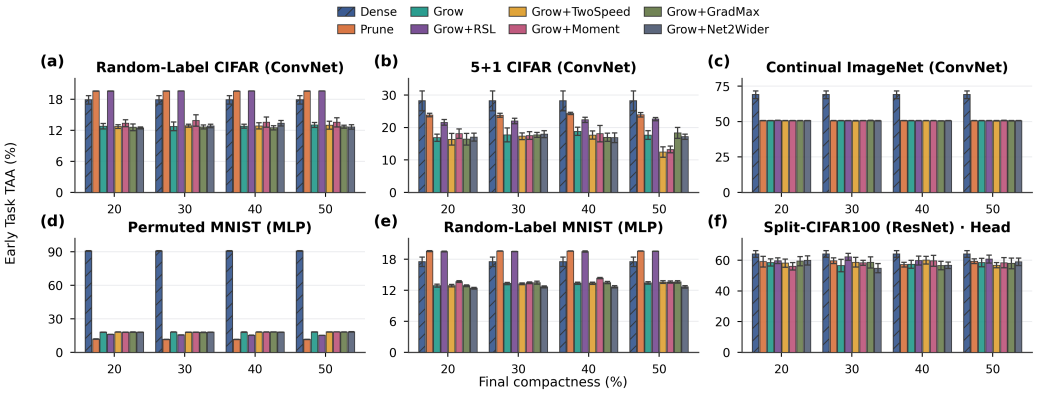

Figure D.1: **Early-task plasticity across sequential-accumulation and repeated-shift benchmarks.** We report *Early Task TAA*, defined as the average accuracy over only the initial portion of each task, emphasizing immediate post-shift adaptation rather than late within-task convergence. DENSE or PRUNE-based methods retain the strongest early-window performance when adaptation time is severely limited, whereas growth-based methods improve when insertion is made more integration-friendly. GROW+RAND. SMOOTH-LEAKY is the most consistently effective growth-family intervention, substantially improving over vanilla GROW. These results support the view that the utility of growth depends critically on how quickly newborn units can integrate after insertion.

E THE USE OF LARGE LANGUAGE MODELS (LLMS)

We used an LLM as a tool for early brainstorming, code debugging, and writing/editing of the earlier drafts of this paper. During ideation, we used it to enumerate experiment variants and sanity-check design choices; for code, we requested bug-finding hints and refactoring suggestions that we implemented only after manual review and testing; for text, we used it to improve clarity, organization, and grammar. The LLM did not generate novel research ideas, experiments, or results on our behalf; all methodological innovations, analyses, and conclusions are our own. We verified any technical claims, equations, and citations suggested during assisted drafting and we did not include uncited, model-generated text verbatim. No proprietary or personally identifiable data was provided to the LLM. The authors retain full responsibility for the content of this paper, and we affirm the novelty and originality of the work.

Comp.	Method	Random-Label CIFAR		5+1 CIFAR		Continual ImageNet		Permuted MNIST		Random-Label MNIST		Split-CIFAR100	
		Avg.	Early	Avg.	Early	Avg.	Early	Avg.	Early	Avg.	Early	Avg.	Early
20%	**Dense**	57.5 ± 15.2	17.9 ± 0.8	18.7 ± 6.7	**28.2 ± 3.1**	72.6 ± 0.5	**68.9 ± 2.8**	81.5 ± 1.2	**90.8 ± 0.3**	48.7 ± 12.8	17.5 ± 0.9	28.5 ± 1.0	**64.1 ± 2.2**
	Prune	**97.9 ± 0.4**	19.6 ± 0.0	**73.8 ± 1.1**	23.9 ± 0.6	**91.8 ± 0.9**	50.7 ± 0.1	**85.9 ± 0.1**	12.0 ± 0.2	**94.4 ± 0.5**	**19.6 ± 0.1**	24.4 ± 0.7	59.1 ± 3.5
	Grow	13.5 ± 0.9	12.8 ± 0.5	23.7 ± 3.8	16.9 ± 1.1	70.7 ± 1.6	50.7 ± 0.1	71.5 ± 0.2	18.1 ± 0.1	16.3 ± 1.4	12.9 ± 0.3	24.5 ± 1.0	58.6 ± 2.3
	Grow+Rand. Smooth-Leaky	94.4 ± 0.2	**19.6 ± 0.0**	63.6 ± 0.9	21.6 ± 0.9	81.5 ± 0.1	50.9 ± 0.1	75.1 ± 0.1	16.2 ± 0.1	72.1 ± 2.3	19.5 ± 0.1	**36.2 ± 0.8**	59.7 ± 1.9
	Grow+TwoSpeed	13.8 ± 0.7	12.8 ± 0.4	20.8 ± 4.4	16.4 ± 1.8	67.3 ± 0.6	50.7 ± 0.1	71.5 ± 0.3	18.3 ± 0.2	16.1 ± 1.3	12.9 ± 0.2	24.9 ± 0.6	58.2 ± 2.6
	Grow+Moment	15.2 ± 1.8	13.4 ± 0.7	25.0 ± 2.9	18.1 ± 1.5	68.9 ± 3.2	50.8 ± 0.1	72.0 ± 0.1	18.1 ± 0.1	17.7 ± 0.7	13.7 ± 0.2	24.4 ± 1.0	56.1 ± 2.4
	Grow+GradMax	13.7 ± 1.5	12.6 ± 0.6	24.4 ± 4.4	16.4 ± 1.8	71.6 ± 2.8	50.7 ± 0.0	71.5 ± 0.3	18.2 ± 0.2	16.2 ± 0.9	12.9 ± 0.2	24.9 ± 1.0	59.5 ± 3.0
	Grow+Net2Wider	13.1 ± 0.5	12.5 ± 0.2	23.0 ± 3.9	17.1 ± 1.2	70.7 ± 3.2	50.8 ± 0.1	71.3 ± 0.2	18.1 ± 0.1	14.8 ± 0.8	12.4 ± 0.2	22.3 ± 0.9	59.9 ± 3.0
30%	**Dense**	57.5 ± 15.2	17.9 ± 0.8	18.7 ± 6.7	**28.2 ± 3.1**	72.6 ± 0.5	**68.9 ± 2.8**	81.5 ± 1.2	**90.8 ± 0.3**	48.7 ± 12.8	17.5 ± 0.9	28.5 ± 1.0	**64.1 ± 2.2**
	Prune	**97.8 ± 0.4**	19.6 ± 0.0	**74.6 ± 1.0**	23.8 ± 0.6	**93.3 ± 0.6**	50.7 ± 0.1	**86.4 ± 0.1**	11.7 ± 0.1	**94.7 ± 0.5**	**19.6 ± 0.0**	24.6 ± 0.6	59.7 ± 1.8
	Grow	14.9 ± 2.3	12.8 ± 0.9	29.4 ± 4.3	17.7 ± 2.2	71.9 ± 2.1	50.7 ± 0.1	73.3 ± 0.2	18.2 ± 0.1	19.4 ± 1.5	13.3 ± 0.2	24.9 ± 0.8	56.6 ± 4.0
	Grow+Rand. Smooth-Leaky	96.1 ± 0.1	**19.6 ± 0.0**	65.3 ± 1.0	22.1 ± 0.8	82.8 ± 0.1	50.8 ± 0.1	78.1 ± 0.0	15.6 ± 0.0	83.4 ± 1.2	19.5 ± 0.0	**38.1 ± 0.9**	62.3 ± 2.3
	Grow+TwoSpeed	15.4 ± 2.1	12.9 ± 0.3	28.3 ± 1.7	17.3 ± 1.0	69.3 ± 0.1	50.7 ± 0.2	73.3 ± 0.2	18.2 ± 0.1	20.1 ± 1.6	13.3 ± 0.2	26.1 ± 1.1	58.8 ± 3.1
	Grow+Moment	19.5 ± 5.1	13.9 ± 1.1	35.1 ± 2.8	17.5 ± 1.2	72.2 ± 1.7	50.7 ± 0.1	74.0 ± 0.1	18.2 ± 0.1	20.3 ± 1.4	13.5 ± 0.2	25.4 ± 0.7	58.4 ± 1.6
	Grow+GradMax	13.9 ± 1.8	12.7 ± 0.4	31.8 ± 2.3	17.7 ± 0.8	69.9 ± 0.2	50.8 ± 0.1	73.4 ± 0.3	18.1 ± 0.2	20.6 ± 1.6	13.5 ± 0.3	25.3 ± 1.0	58.7 ± 3.6
	Grow+Net2Wider	15.1 ± 1.5	12.9 ± 0.3	30.6 ± 3.0	18.0 ± 1.1	67.9 ± 0.3	50.7 ± 0.1	73.1 ± 0.2	18.1 ± 0.2	16.5 ± 1.1	12.7 ± 0.2	23.2 ± 0.8	54.9 ± 3.0
40%	**Dense**	57.5 ± 15.2	17.9 ± 0.8	18.7 ± 6.7	**28.2 ± 3.1**	72.6 ± 0.5	**68.9 ± 2.8**	81.5 ± 1.2	**90.8 ± 0.3**	48.7 ± 12.8	17.5 ± 0.9	28.5 ± 1.0	**64.1 ± 2.2**
	Prune	**98.0 ± 0.3**	19.6 ± 0.0	**75.4 ± 0.9**	24.4 ± 0.3	**94.0 ± 0.4**	50.7 ± 0.1	**86.7 ± 0.2**	11.7 ± 0.1	**94.9 ± 0.5**	**19.6 ± 0.0**	24.3 ± 0.4	57.2 ± 1.5
	Grow	14.1 ± 0.6	12.8 ± 0.4	33.8 ± 4.3	18.8 ± 1.3	70.4 ± 0.2	50.7 ± 0.1	74.7 ± 0.2	18.2 ± 0.1	21.1 ± 2.0	13.4 ± 0.2	25.3 ± 1.0	57.3 ± 2.6
	Grow+Rand. Smooth-Leaky	96.8 ± 0.1	**19.6 ± 0.0**	65.0 ± 0.8	22.4 ± 0.8	83.5 ± 0.1	50.7 ± 0.1	80.0 ± 0.0	15.4 ± 0.0	84.1 ± 0.9	19.5 ± 0.1	**36.9 ± 1.2**	59.8 ± 2.9
	Grow+TwoSpeed	14.6 ± 1.4	12.9 ± 0.6	30.2 ± 2.8	17.7 ± 1.3	70.6 ± 0.2	50.7 ± 0.1	75.0 ± 0.2	18.2 ± 0.1	21.0 ± 1.8	13.4 ± 0.2	25.9 ± 0.8	60.1 ± 2.5
	Grow+Moment	17.4 ± 5.9	13.6 ± 1.0	38.3 ± 6.3	18.2 ± 2.5	70.2 ± 2.5	50.7 ± 0.1	75.0 ± 0.1	18.3 ± 0.1	22.4 ± 1.2	14.4 ± 0.1	26.1 ± 0.8	59.7 ± 3.5
	Grow+GradMax	13.6 ± 0.7	12.5 ± 0.4	30.0 ± 2.4	17.0 ± 1.3	70.5 ± 0.3	50.6 ± 0.1	74.6 ± 0.2	18.3 ± 0.1	22.2 ± 2.0	13.5 ± 0.2	25.2 ± 0.7	56.6 ± 2.7
	Grow+Net2Wider	18.0 ± 2.7	13.4 ± 0.5	31.8 ± 4.4	16.9 ± 1.5	68.5 ± 0.3	50.7 ± 0.1	74.5 ± 0.2	18.1 ± 0.1	17.5 ± 1.4	12.7 ± 0.2	24.1 ± 0.8	56.8 ± 2.1
50%	**Dense**	57.5 ± 15.2	17.9 ± 0.8	18.7 ± 6.7	**28.2 ± 3.1**	72.6 ± 0.5	**68.9 ± 2.8**	81.5 ± 1.2	**90.8 ± 0.3**	48.7 ± 12.8	17.5 ± 0.9	28.5 ± 1.0	**64.1 ± 2.2**
	Prune	**98.1 ± 0.3**	19.6 ± 0.0	**75.5 ± 0.8**	23.9 ± 0.7	**94.5 ± 0.3**	50.7 ± 0.1	**86.9 ± 0.3**	11.6 ± 0.1	**95.1 ± 0.5**	**19.6 ± 0.0**	24.7 ± 0.3	59.4 ± 1.5
	Grow	15.4 ± 2.4	13.0 ± 0.5	34.4 ± 3.7	17.7 ± 1.3	71.3 ± 0.3	50.7 ± 0.1	75.7 ± 0.1	18.3 ± 0.1	23.2 ± 2.3	13.4 ± 0.2	26.0 ± 0.8	58.6 ± 2.7
	Grow+Rand. Smooth-Leaky	97.1 ± 0.1	**19.6 ± 0.0**	67.9 ± 0.5	22.6 ± 0.5	84.2 ± 0.1	50.7 ± 0.1	81.3 ± 0.0	15.2 ± 0.0	85.9 ± 0.7	19.6 ± 0.0	**37.8 ± 1.2**	60.8 ± 2.6
	Grow+TwoSpeed	15.8 ± 3.4	13.0 ± 0.8	29.3 ± 1.3	12.5 ± 1.6	71.2 ± 0.3	50.6 ± 0.1	75.6 ± 0.2	18.4 ± 0.1	23.6 ± 1.8	13.6 ± 0.2	25.6 ± 0.8	56.8 ± 1.7
	Grow+Moment	17.3 ± 3.5	13.6 ± 0.9	31.6 ± 0.7	13.3 ± 1.0	70.8 ± 0.2	50.8 ± 0.1	76.1 ± 0.2	18.4 ± 0.1	23.7 ± 2.0	13.6 ± 0.2	26.0 ± 1.2	58.4 ± 3.3
	Grow+GradMax	13.8 ± 0.6	12.7 ± 0.3	33.7 ± 5.2	18.4 ± 1.7	71.5 ± 0.3	50.6 ± 0.1	75.5 ± 0.2	18.3 ± 0.2	24.3 ± 2.4	13.6 ± 0.2	25.6 ± 0.9	58.0 ± 3.5
	Grow+Net2Wider	14.8 ± 2.4	12.7 ± 0.4	30.9 ± 3.9	17.2 ± 0.8	69.6 ± 0.3	50.7 ± 0.1	75.4 ± 0.1	18.4 ± 0.1	18.5 ± 1.7	12.6 ± 0.2	24.8 ± 0.5	59.0 ± 2.5

Table D.1: **Continual-learning summary (cycle)**. For each dataset we report Avg. Acc. and Early Task TAA. Within each compactness block, the best value for each dataset–metric pair is shown in bold. Dense is shown as the full-capacity reference within each compactness block, while non-dense methods are compared at matched compactness. Dense uses a dataset-specific fixed learning rate; all other methods select the best learning rate by Avg. Acc. among $\{0.01, 0.001, 0.0001\}$.