ARCHITECTURAL PLASTICITY FOR CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks for continual reinforcement learning (CRL) often suffer from plasticity loss, i.e., a progressive decline in their ability to learn new tasks arising from increased churn and Neural Tangent Kernel (NTK) rank collapse. We propose InterpLayers, a drop-in architectural solution that combines a fixed, parameter-free reference pathway with a learnable projection pathway using input-dependent interpolation weights. Without requiring algorithmic adaptation, InterpLayers conserve gradient diversity and constrain output variability by integrating stable and adaptive computations. We provide theoretical guarantees for bounded churn and show that, under mild assumptions, InterpLayers prevent NTK rank collapse through a non-zero rank contribution from the interpolation weights. Across environments with distributional shifts including permutation, windowing, and expansion, InterpLayer variants (conv-only, fullinterp) consistently mitigate performance degradation compared to parameter-matched baselines. Furthermore, lightweight modifications such as dropout improve performance, especially under gradual shifts. These results position InterpLayers as a simple, complementary solution for maintaining plasticity in CRL.

1 INTRODUCTION

Continual reinforcement learning (CRL) requires agents to adapt to a non-stationary stream of tasks without external resets or explicit knowledge of task boundaries. Yet neural networks trained in this setting suffer from *plasticity loss*: their ability to adapt to new tasks diminishes over time. Plasticity loss has been attributed to several interacting factors, including rank collapse of the Neural Tangent Kernel (NTK) (Lyle et al., 2024), unbounded weight growth (Lyle et al., 2023), and representational drift or churn that destabilizes previously acquired knowledge (Tang et al., 2025).

Most existing solutions intervene at the algorithmic level. Reset-based strategies reinitialize parameters on a fixed schedule (Igl et al., 2020; Nikishin et al., 2022; 2023). Continuous plasticity methods modify the optimization process itself, e.g., shrink-perturb (Ash & Adams, 2020), ReDo (Sokar et al., 2023), or regenerative regularization (Kumar et al., 2023). Constraint-based approaches rely on normalization, clipping, or masking to restrict parameter dynamics (Ba et al., 2016; Abbas et al., 2023; Elsayed et al., 2024). While effective, these methods share limitations, including: (i) requiring task boundary information or chosen reset schedules; (ii) introducing hyperparameters such as reset frequencies, perturbation magnitudes, or regularization strengths; (iii) acting externally to the architecture, often outside the optimization framework.

Here, we offer a distinct alternative by addressing plasticity loss directly at the architectural level, without the need for interventions during training. Our method enhances standard network layers with additional pathways to build *Interpolation Layers* (InterpLayers). Each layer combines a fixed, parameter-free reference pathway that preserves stable representations throughout training and a learnable projection pathway that adapts through backpropagation, connected via input-dependent interpolation weights. By dynamically interpolating between these pathways, the network maintains representational stability while preserving the capacity for adaptive learning. Unlike ResNet-like skip connections, which only diversify gradient flow, or parameter-efficient tuning methods such as LoRA, which fine-tune computational efficiency, InterpLayers create a self-regulating mechanism that balances stability and plasticity without external intervention. Moreover, compared to algorithmic approaches like soft-shrink-perturb with layer normalization (Juliani & Ash, 2024), Inter-

pLayers require minimal computational overhead and no additional schedules or hyperparameters. Designed as orthogonal components to current solutions for plasticity loss, they can be integrated seamlessly into existing architectures or combined with intervention mechanisms.

We evaluate InterpLayers both theoretically and empirically. We perform a theoretical analysis to investigate how InterpLayers impact churn and NTK rank, demonstrating that these properties are enhanced by the interpolation mechanism between reference and projection pathways. For empirical evaluation, we evaluate the performance of InterpLayers over standard baselines for ProcGen tasks as described in (Juliani & Ash, 2024). We also investigate the performance of InterpLayers when combined with dropout (Srivastava et al., 2014) and discuss how to effectively combine InterpLayers orthogonally with other methods that tackle plasticity loss. We show that InterpLayers are effective in preventing plasticity loss and can be a direction for future architectural solutions for continual learning.

Our main contributions can be denoted as follows.

1. We introduce InterpLayers as drop-in replacements for conventional neural network layers. InterpLayers splits the layer input into a reference and a projection pathway that are further interpolated to obtain the layer's output.
2. We show that InterpLayers bound representational drift through controlled interpolation, limit churn growth via pathway stability, and maintain NTK rank under specific assumptions. These guarantees emerge from architectural constraints rather than external interventions.
3. Across ProcGen distribution shifts spanning pixel permutations, level expansion, and sequential task changes, InterpLayers preserve performance where standard multi-layer perceptron (MLP) layers collapse. We also empirically compare the performance of InterpLayers with other interventions to counter plasticity loss.

2 Related works

2.1 Algorithmic Approaches to Mitigate Plasticity Loss

Reset-based interventions. Periodic parameter reinitialization has often been applied to counter plasticity loss. (Igl et al., 2020) proposed resetting only the final layer to preserve learned features while restoring adaptability. (Nikishin et al., 2022) showed that resetting selected network parameters on a fixed schedule restores the network's capacity to learn. Later, (Nikishin et al., 2023) has shown that resetting the entire network leads to maintenance of plasticity at the cost of losing prior knowledge. To implement these methods, reset schedules and selecting which parameters to reinitialize is needed.

Continuous plasticity upkeep. Other methods continuously regulate plasticity during training. (Sokar et al., 2023) proposed ReDo, which periodically resets inactive neurons. A continual backpropagation method was presented by (Dohare et al., 2024), which adds a step to backpropagation where a small fraction of neurons are continuously reinitialized based on a utility metric. (Ash & Adams, 2020) applied a shrink-and-perturb methodology to the network after each update to scale down the weights and add noise in order to maintain plasticity. To prevent unbounded weight drift, (Kumar et al., 2023) used regenerative regularization applying L2 penalties to weights.

Normalization and constraint-based methods. Another approaches alleviate plasticity loss by constraining the network dynamics. (Lyle et al., 2023) showed that LayerNorm can slow down plasticity loss, as it helps to maintain NTK rank. (Abbas et al., 2023) investigated weight clipping to provide an upper bound to parameter growth. To stabilize optimization, (Miyato et al., 2018) has shown that spectral normalization can constrain Lipschitz constants. Even though plasticity loss is reduced, representational capacity is also affected by the constraints added by these methods.

2.2 Architectural Mechanisms for Stability in Neural Networks

Various innovations in neural network architectures have been proposed to balance stability and plasticity, even though they have not been directly applied to continual learning. Skip connections

and residual pathways have been vastly investigated to create gradient highways and regulate the information flows in computer vision (He et al., 2016; Srivastava et al., 2015). Gating mechanisms for controlling information flow have also been highly effective in natural language processing architectures (Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Networks that generate specific parameters conditioned on input features, such as HyperNetworks (Ha et al., 2016), have also been investigated to introduce architectural flexibility in meta-learning tasks. Here, these methods serve as a foundation for the theoretical modeling of InterpLayers, which introduce an asymmetry by keeping one pathway fixed and parameter-free, thereby achieving input specificity and representational stability.

2.3 Theoretical Understanding of Plasticity Loss

Recent works have also explored key theoretical features to enhance understanding of plasticity loss in neural networks. (Lyle et al., 2024) showed that the effective NTK rank is strongly linked with the ability of the network to adapt in a continual learning setting. Specifically, they demonstrate that NTK rank collapse correlates with a decrease in performance. The unconstrained drift of internal network representation has also been described as a cause for catastrophic forgetting in CRL by (Kumar et al., 2023). The instability of network outputs, i.e., *churn*, is investigated by (Tang & Berseth, 2024; Tang et al., 2025) as an important factor in plasticity loss. Based on these findings, we theoretically investigate the effects of InterpLayers on these metrics.

3 Methods

3.1 Preliminaries

We consider an agent that learns in a CRL environment interacting with a sequence of tasks $\{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_K\}$ following a Markov Decision Process (MDP), where each $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{A}_i, P_i, r_i, \gamma)$ may have different state spaces $\mathcal{S}_i$, action spaces $\mathcal{A}_i$, transition dynamics P_i, and reward functions r_i. The tasks are separated by distribution shifts, which can range from small changes, e.g., reinitializing the environment with a new random seed, to substantial changes, e.g., permutations on the observation axis that completely modify the input distribution. At each timestep t, the agent observes state s_t, selects action a_t according to policy $\pi_\theta(a|s)$, receives reward r_t, and transitions to state s_{t+1}. The policy $\pi_\theta(a|s)$ is parameterized by a neural network with weight parameters θ and trained via backpropagation.

In a continual learning setting, the current task $\mathcal{M}$ is changed after a fixed number of environment steps. The agent is given no information about task boundaries or identities, so it does not know which task it has to solve at a given moment. The agent should adapt to a new task by modifying its set of parameters θ online, having a shared policy for multiple tasks. The policy does not store past experiences in another data structure to sample from during training. In this way, the policy should maintain a balance between stability (preserving knowledge) and plasticity (acquiring new knowledge) in a non-stationary environment.

3.2 The Interpolation Layer

As an architectural solution to tackle plasticity loss, we introduce InterpLayers (Figure 1), which are task-agnostic, require no additional hyperparameters, and can be seamlessly integrated into existing neural network architectures.

Core mechanism. Each InterpLayer splits computation into two complementary pathways: (i) a *reference pathway* given by a fixed, parameter-free mapping (identity, sparse selection, or padding when dimensions differ); and (ii) a *projection pathway* with standard learnable parameters. Learnable interpolation weights then combine both outputs, allowing the network to learn when to rely on preservation and when to adapt. Mathematically, given an input $\mathbf{x} \in \mathbb{R}^d$, the InterpLayer output is given as

$$\mathbf{h}(\mathbf{x}) = (1 - z(\mathbf{x})) \odot h_{\text{ref}}(\mathbf{x}) + z(\mathbf{x}) \odot h_{\text{proj}}(\mathbf{x}), \tag{1}$$

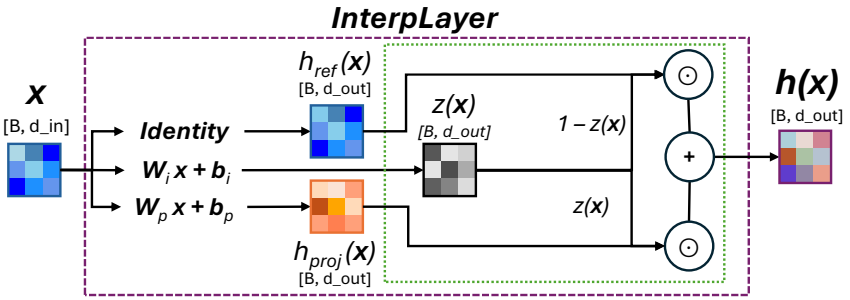


Figure 1: **The InterpLayer Architecture.** The input $\mathbf{x}$ is processed through a fixed reference pathway $h_{\text{ref}}(\mathbf{x})$ and a learnable projection pathway $h_{\text{proj}}(\mathbf{x})$. The learnable interpolation weights $z(\mathbf{x})$ dynamically interpolate the outputs from both pathways to produce the output $\mathbf{h}(\mathbf{x})$.

where $\odot$ denotes element-wise multiplication and h_{ref}, h_{proj}, and $z(\mathbf{x})$ are defined as

$$h_{\text{ref}}(\mathbf{x}) = \mathbf{P}\mathbf{x}, \quad (\mathbf{P} = \mathbf{I} \text{ when } d_{\text{in}} = d_{\text{out}}), \tag{2}$$

$$h_{\text{proj}}(\mathbf{x}) = \phi(\mathbf{W}_p\mathbf{x} + \mathbf{b}_p), \quad \mathbf{W}_p, \mathbf{b}_p \text{ (learnable)}, \tag{3}$$

$$z(\mathbf{x}) = \sigma(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i), \quad \mathbf{W}_i, \mathbf{b}_i \text{ (learnable)}, \tag{4}$$

d_{in} and d_{out} denote the input and output dimensionalities of the layer, ϕ is a non-linear activation function and σ is a sigmoid layer.

The reference pathway is designed to provide a parameter-free skip connection; if input and output dimensions/channels match, the reference pathway applies an identity mapping to the input; if the dimensions/channels differ, the input passes through a fixed projection $\mathbf{P}$. We implement $\mathbf{P}$ as either a sparse sampling matrix (selecting h dimensions from d_{in} when $d_{in} > d_{out}$) or a zero-padding operation (when $d_{in} < d_{out}$), both deterministically constructed without learned parameters. The weights of this pathway remain fixed throughout training to provide a stable reference and prevent unbounded representational drift. In contrast, the projection pathway enables adaptation through standard learning, similarly to an MLP layer. The interpolation weights $z(\mathbf{x}) \in (0,1)^h$ regulate the contribution of reference and projection, providing the network with a dynamic preservation–adaptation tradeoff. This mechanism is similar to input gates in GRUs (Cho et al., 2014), but has a key difference: h_{ref} is a fixed skip from the current input rather than a recurrent hidden state from the past. Weight magnitudes closer to 0 are related to $\mathbf{h}(\mathbf{x})$ being mostly represented by the reference, while weight magnitudes closer to 1 are related to $\mathbf{h}(\mathbf{x})$ being mostly represented by the projection.

Integration to convolutional layers. InterpLayers can replace standard MLP layers following Eqs. (1)-(4). For convolutional layers processing image data $\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times H \times W}$ as part of the state space, h_{ref}, h_{proj}, and $z(\mathbf{x})$ are defined as

$$h_{\text{ref}}(\mathbf{X}) = \mathbf{P}_r * \mathbf{X} \tag{5}$$

$$h_{\text{proj}}(\mathbf{X}) = \phi(\mathbf{W}_p * \mathbf{X} + \mathbf{b}_p), \tag{6}$$

$$z(\mathbf{X}) = \sigma(\mathbf{W}_i \cdot \beta(\mathbf{X}) + \mathbf{b}_i)\,, \tag{7}$$

where $*$ denotes a convolution operation and β is a global average pooling operation to produce channel-wise interpolation.

3.3 Theoretical Properties of InterpLayers

We analyze the mathematical properties of InterpLayers, focusing on two key properties: bounded representational drift and preservation of gradient diversity.

3.3.1 Bounded Representational Drift

The dual-pathway structure of InterpLayers ensures that changes in the output remain bounded under parameter updates. For an update $\Delta\theta = (\Delta\theta_p, \Delta\theta_z)$, the first-order output change is given as

$$\Delta h(\mathbf{x}) = z(\mathbf{x}) \odot \Delta h_{\text{proj}}(\mathbf{x}) + \Delta z(\mathbf{x}) \odot [h_{\text{proj}}(\mathbf{x}) - h_{\text{ref}}(\mathbf{x})]. \tag{8}$$

This decomposition shows that updates are constrained. The projection pathway update is modulated by the interpolation weights $z(\mathbf{x}) \in (0,1)^h$, while the interpolation update is bounded by the pathway difference.

Theorem 1 (Bounded Output Variability)**.** *If h_{proj} is L_p-Lipschitz in its parameters θ_p and z is L_z-Lipschitz in θ_z, then*

$$\|\Delta h(\mathbf{x})\|_2 \leq \ \|z(\mathbf{x})\|_\infty L_p \|\Delta\theta_p\|_2 + L_z \|\Delta\theta_z\|_2 D(\mathbf{x}), \tag{9}$$

where $D(\mathbf{x}) = \|h_{proj}(\mathbf{x}) - h_{ref}(\mathbf{x})\|_2$.

The proof is deferred to Appendix A.1. This bound implies that churn is polynomially bounded in training steps, in contrast to standard MLP layers where churn may grow unboundedly with parameter norms. This theorem makes use of the fact that the reference pathway is parameter-free at the layer level, and so only projection and interpolation weights contribute to the drift.

3.3.2 Gradient diversity preservation.

InterpLayers preserve gradient diversity by altering the structure of the NTK. Given the InterpLayer formulation, the gradient with respect to network parameters decomposes as

$$\nabla_\theta h(x) = \begin{bmatrix} z(x) \odot \nabla_{\theta_p} h_{\text{proj}}(x) \\ \nabla_{\theta_z} z(x) \odot \big(h_{\text{proj}}(x) - h_{\text{ref}}(x)\big) \end{bmatrix}. \tag{10}$$

This yields an NTK of the form

$$N_{\text{IL}}(x_i, x_j) = (z(x_i) \odot z(x_j))^\top N_{\text{proj}}(x_i, x_j) + N_{\text{interp}}(x_i, x_j), \tag{11}$$

where N_{proj} and N_{interp} denote the NTK contributions from projection and interpolation parameters, respectively. Here i, j index input samples x_i, x_j rather than parameters. Intuitively, the interpolation mechanism adds a persistent gradient component even when the projection pathway degenerates, sustaining diversity in the NTK. For readers unfamiliar with NTK calculations, we provide a step-by-step derivation and empirical estimator details in Appendix A.2.1.

Theorem 2 (NTK Rank Preservation under Interpolation Variance)**.** *Suppose the interpolation weights $z(x)$ across samples have non-zero variance along at least one coordinate, i.e.,*

$$\text{Var}[z_{(k)}(x)] > 0 \quad \textit{for some dimension } k.$$

Then the effective NTK rank of an InterpLayer is lower-bounded by

$$\text{rank}(N_{IL}) \ \geq \ \text{rank}(N_{interp}).$$

In particular, the interpolation pathway guarantees a persistent gradient component, preventing full rank collapse even if the projection pathway degenerates.

The key requirement of Theorem 2 is simply that interpolation weights exhibit variance across samples. Intuitively, as long as $z(x)$ does not collapse to a constant vector, the interpolation pathway contributes an independent gradient component to the NTK. This guarantees a persistent lower bound on effective rank and prevents full rank collapse, even in cases where the projection pathway degenerates. Empirical verification of NTK rank during training is provided in Section 4.3.

4 Results

4.1 Experimental Setup

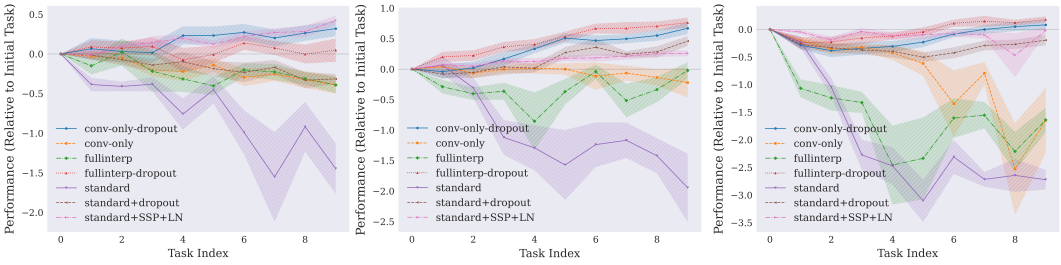
We employ the ProcGen environment (Cobbe et al., 2020) to evaluate the proposed framework on CRL settings. As benchmark tasks, we apply three distribution shifts on the *coinrun* environment previously introduced by (Juliani & Ash, 2024) (visualizations shown in Appendix G). These three variations are named *permute*, *window*, and *expand*. For the *permute* task, at each shift point, we randomly permute all pixels in the observation space. In the *window* task, the random seed used to generate the levels is changed at each shift point. In the *expand* task, training starts with 100 levels, and at each shift point the training set is continuously expanded by increments of 100, ending with 1000 levels after the final shift.

InterpLayer Variants. We evaluate two architectural variants: (i) **conv-only**, where InterpLayers replace only the convolutional encoder layers, and (ii) **fullinterp**, where both convolutional and linear layers are replaced with InterpLayers. We also investigate InterpLayers combined with dropout (Srivastava et al., 2014), which we name **conv-only-dropout** and **fullinterp-dropout** respectively, in which dropout is applied to the projection pathway. The conv-only variant emphasizes stability in low-level feature extraction, while fullinterp exposes the entire network to interpolation. Adding dropout aims to increase variance in the projection pathway, which increases the gap between reference and projection. We hypothesize that the characteristics of dropout enhance the effects of our proposed interpolation mechanism.

The policy used in the experiments consists of an encoder using 4 convolutional layers followed by a linear layer. The training is performed using PPO (Schulman et al., 2017). Given the additional number of parameters introduced by InterpLayers, we compare it with an architecture using a similar number of parameters as our *standard* baseline. Details regarding the training details and computational cost comparison are given in Appendix B and C, respectively. Additionally, our method is compared against the top-performing baseline proposed and benchmarked in (Juliani & Ash, 2024): soft shrink-perturb with layer norm (SSP+LN), which mixes the current weight with initialization noise after each optimizer step (See Appendix D for details. The results are average runs of 10 random seeds where training is performed for 50,000 epochs, with distribution shifts being applied every 5,000 epochs.

4.2 InterpLayer Performance Under Distribution Shifts

We evaluate whether InterpLayers can maintain performance across sequential tasks separated by distribution shifts. Figure 2 shows the normalized performance, defined as the mean reward over the final 50 evaluation episodes of each task, normalized relative to the initial task and plotted with shaded regions denoting the standard error across 10 seeds for seven network variants: conv-only, fullinterp, and their dropout variants, as well as the baselines, standard, standard with dropout, and SSP+LN. For all dropout variants, we set the dropout rate to 0.1.



(a) Permute (b) Window (c) Expand

Figure 2: **Performance (relative to initial task) for InterpLayer variants and baselines under ProcGen distribution shifts.** Performance is defined as the mean reward over the final 50 evaluation episodes of each task, normalized relative to the initial task, with shaded regions denoting standard error across 10 seeds. (a) Permute: The standard baseline collapses throughout training, while non-dropout interpolation-based networks degrade more gradually. Conv-only-dropout and fullinterp-dropout negate any plasticity loss, with SSP+LN and conv-only-dropout reaching the highest performance. (b) Window: Conv-only-dropout sustains near-initial performance across tasks. Fullinterp-dropout improves steadily throughout training and outperforms all baselines, including SSP+LN. (c) Expand: The standard baseline collapses quickly, while conv-only and fullinterp decline with fluctuations. Fullinterp-dropout and conv-only-dropout reach above initial-task performance, while SSP+LN remains stable in comparison to the two dropout variants. Raw episodic returns are presented in Appendix H.

Permute (Figure 2a): The permute task involves the most severe shift, forcing full representational relearning. The standard baseline collapses quickly, dropping performance below -1.5 relative to the initial task. Adding dropout delays this decay, but standard-dropout still falls far below zero. Interpolation-based methods show similar degradation, albeit with some recovery in later tasks. Adding dropout to the projection in the interpolation variants yields strong resilience, with fullinterp-

dropout finishing close to initial-task performance. Conv-only-dropout and SSP+LN remain above zero, neither outperforming the other, with conv-only-dropout being computationally cheaper (see Appendix C).

Window (Figure 2b): Changing to newly generated levels at each shift produces a clear performance separation. The standard baseline drops steadily, remaining below -1.0 after task 3; adding dropout prevents this collapse, reaching above initial-task performance in the later task, similar to SSP+LN. Conv-only stays near zero, while its dropout variant also stabilizes and then exceeds initial performance. Fullinterp shows fluctuating performance but partial recovery in the later tasks. Its dropout variant shows the strongest overall performance, maintaining positive returns across all tasks and outperforming all other methods, with the conv-only dropout coming in as second best.

Expand (Figure 2c): Increasing the number of levels provides a gradual adaptation challenge. The standard baseline drops strongly to below -2.0 by mid-training. Standard-dropout lowers this drop but does not avoid the decrease in performance. Both interpolation variants decline with large variance; conv-only drops more smoothly, yet fullinterp shows some recovery in later tasks. SSP+LN remains stable near initial performance. Fullinterp-dropout and conv-only-dropout recover during training, and both end with a positive trend reaching performance above 0, outperforming all other methods.

Across all shift types, interpolation-based networks outperform the standard baseline. Dropout plays an important role: for the standard model, adding dropout improves resistance against plasticity loss but is unable to fully resolve it. For InterpLayer-based architectures, adding dropout mitigates plasticity loss. Conv-only-dropout reaches stable performance across all shifts, while fullinterp-dropout combines stability and adaptability, dominating in more gradual shifts like window and expand, while still remaining competitive in the permute condition. Compared to SSP+LN, dropout-enhanced InterpLayers preserve plasticity while requiring less computation (Appendix C) and not applying optimization-level interventions.

4.3 Empirical Validation of Theoretical Properties

We show the empirical validation for two metrics related to the theoretical properties of InterpLayers, churn and NTK rank, in Figure 3. Details on the methodology for calculating these metrics are provided in Appendix F and E.

Churn. The churn evolution for InterpLayer variants and baselines is presented in the first column of Figure 3 for the three tasks. We observe that fullinterp-dropout and conv-only-dropout achieve lower churn and reduces churn over time, while the churn for SSP+LN and standard-dropout remains stable. These values align with the theoretical analysis presented in Theorem 1. We also observe that patterns for the permute task are different from the ones for the window and expand tasks. For the permute, clear separation is seen in shift points, related to the need of learning a new set of features given the complexity of the distribution shift.

NTK Rank Preservation. We estimate NTK rank by computing per-sample gradients $g(x)$ of the PPO policy and value heads. The NTK matrix is constructed as $K_{ij} = \langle g(x_i), g(x_j) \rangle$, and we report the effective rank. Computation is performed every 50 epochs on a batch of 128 states, requiring a moderate overhead (about 15% training time increase). As shown in Figure 3, the fullinterp-dropout variant maintains significantly higher effective rank, especially for the window and expand tasks, and overall sustains gradient diversity, which is key for continual learning. For the permute task, SSP+LN also achieves a high rank throughout training.

4.4 Analyzing the Interpolation Mechanism

Figure 4 shows per-layer distributions of interpolation weights averaged for early training (tasks 1-5) and late training (tasks 6-10). In both variants, fullinterp-dropout and conv-only-dropout, early training is characterized by high variance and broad weight distributions. In late training the distributions shift towards the reference pathway, which indicates that low-level features are stabilized. The key difference between the variants emerges in the final layer (L4): in fullinterp-dropout, the interpolation weight distribution remains broad and biased towards the projection pathway, while in conv-only-dropout L4 is a standard linear layer, equivalent to fixing all interpolation weights z to 1.

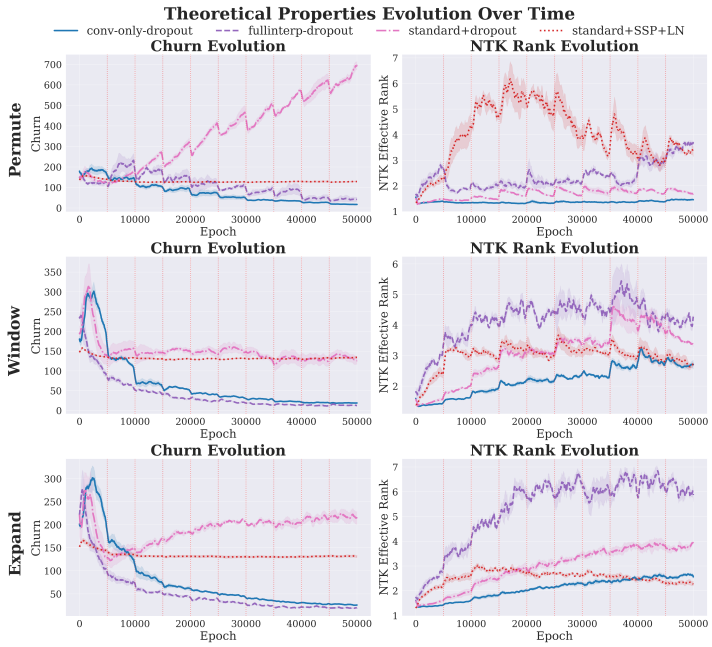


Figure 3: **Evolution of the theoretical metrics for InterpLayer variants and baselines under distribution shifts.** The metrics track representational stability and plasticity across training. Rows correspond to distribution shifts: (a) Permute, (b) Window, (c) Expand. Columns correspond to two different measures: Churn and effective NTK rank. Shaded regions denote variability across 3 seeds, and vertical lines indicate shift points. Across all conditions, the interpolation variants maintain lower churn compared to the standard baselines, while the NTK effective rank is best preserved under the fullinterp-dropout variant.

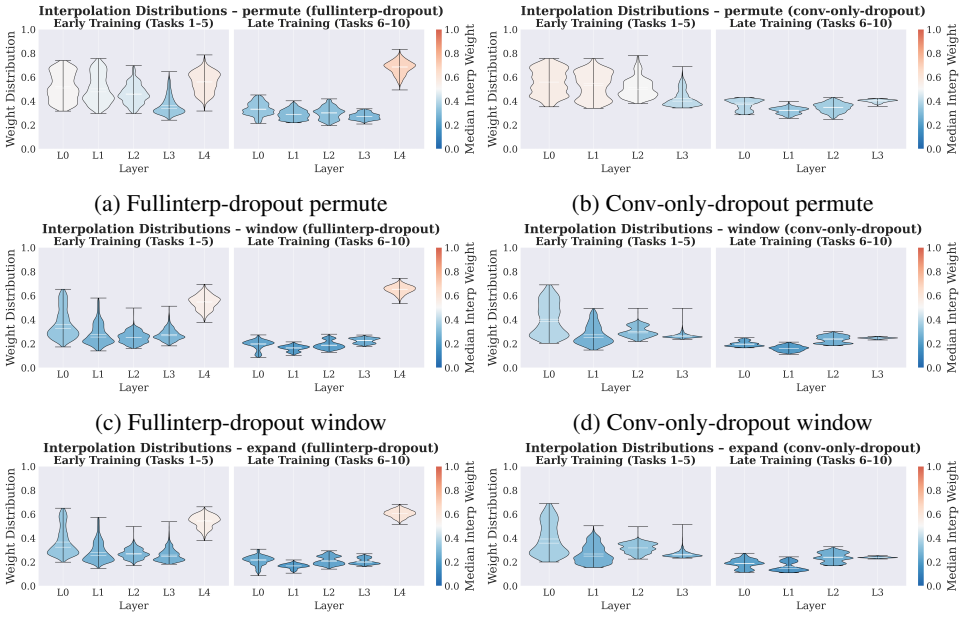
5 Discussion

In our experiments, we investigate the learning patterns for different InterpLayer variants (Figure 2). Applying InterpLayers only at the convolutional-based feature extractor (conv-only) is shown to be effective for the initial tasks, while applying InterpLayers for all layers (fullinterp) shows recovery in performance for the final tasks, even in situations where the performance drops for intermediate tasks. This suggests the potential of developing hybrid strategies in which interpolation is performed layer-wise, given plasticity requirements, for dynamic tasks.

The analysis in Figure 4 shows that InterpLayer variants develop a hierarchical structure implicitly. While fixed interpolation weights of $z = 0.5$ would act like ResNet-like skip connections, allowing z to learn in an input-conditioned manner enables lower layers to prioritize the reference while the final layer sustains projection contributions. This splitting of jobs is not hard-wired into the architecture but develops naturally from the input-specific interpolation. Such self-organization is similar to other findings in deep learning, where lower layers act as feature extractors while higher layers adapt to task-specific demands (Yosinski et al., 2014).

The evolution of metrics related to the theoretical properties presented in Section 3.3 is crucial to mitigate plasticity loss. Our empirical results for churn evolution (Figure 3) show that it decreases over time using InterpLayers variants. These results agree with results recently presented by (Tang et al., 2025), demonstrating that reducing churn is important to keep plasticity in neural networks. Another important property linked to effective network adaptation is NTK rank as discussed by (Lyle et al., 2024). For the NTK rank (Figure 3), InterpLayers also preserve rank during training. These findings suggest empirically the theoretical advantages of using InterpLayers in continual learning.

Furthermore, the analysis in Section 3.3 suggests that the plasticity of the network can be estimated through the joint behavior of the interpolation weights z and the representational gap D defined in Theorem 1. Together, z and D indicate how much a layer adapts. These variables are important



(a) Fullinterp-dropout permute

(b) Conv-only-dropout permute

(c) Fullinterp-dropout window

(d) Conv-only-dropout window

(e) Fullinterp-dropout expand

(f) Conv-only-dropout expand

Figure 4: **Distribution per-layer of interpolation weights in early training (tasks 1-5) and late training (tasks 6-10).** Six conditions are shown: fullinterp-dropout (first column) and conv-only-dropout (second column) under permute, window, and expand tasks.

to understand the learning of InterpLayer variants combined with dropout (Figure 2). Dropout is stochastically masking projection activations, preventing projection and reference from aligning, i.e., sustaining D. Even though z is not directly affected by dropout, because it receives gradients that scale with D, keeping a high representational gap D maintains the effective learning of z. These effects directly influence how InterpLayers can be orthogonally combined with other regularization methods. While combining with dropout improves performance, combining with other methods that decrease D, such as shrink and perturb, might hurt the performance of InterpLayers.

Architectures with gated mechanisms (Hochreiter & Schmidhuber, 1997; Cho et al., 2014) and residual networks (He et al., 2016; Srivastava et al., 2015) have been responsible for key advances in recurrent neural networks and convolutional neural networks, respectively. In the same direction, InterpLayers present an interpolation mechanism that sustains plasticity through different streams and gated interventions while also providing a complementary architectural axis to other methods, preventing plasticity loss. This resonates with neuroscience-inspired models where dendritic compartments and gating mechanisms solve the stability-plasticity trade-off in cortical circuits (Bengio et al., 2015; Urbanczik & Senn, 2014). Our findings place InterpLayers as a simple but general mechanism that enriches the toolbox of CRL toward architectures implicitly solving the plasticity loss issue.

6 Conclusion

In this paper, we introduce InterpLayers as an architectural solution to plasticity loss in CRL. Requiring no schedule, resets, or auxiliary objectives, InterpLayers provide continuous regulation of plasticity through a dual-pathway design. Our findings show that InterpLayers mitigate plasticity loss across three sequential tasks from ProcGen. Additionally, we show that combining InterpLayer with dropout improves its performance, achieving comparable performance to state-of-the-art methods for continual learning, suggesting that characteristics learned by dropout regularization help the interpolation dynamics of InterpLayers. Future works include investigating the performance of InterpLayers with different levels of sparsity in the policy network and the combination with different algorithmic approaches in CRL.

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.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3884–3894. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

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(8026): 768–774, 2024.

Mohamed Elsayed, Qingfeng Lan, Clare Lyle, and A Rupam Mahmood. Weight clipping for deep continual and reinforcement learning. *arXiv preprint arXiv:2407.01704*, 2024.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

Arthur Juliani and Jordan Ash. A study of plasticity loss in on-policy deep reinforcement learning. *Advances in Neural Information Processing Systems*, 37:113884–113910, 2024.

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

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.

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the causes of plasticity loss in neural networks. *arXiv preprint arXiv:2402.18762*, 2024.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. *Advances in Neural Information Processing Systems*, 36:37142–37159, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Hongyao Tang and Glen Berseth. Improving deep reinforcement learning by reducing the chain effect of value and policy churn. *Advances in Neural Information Processing Systems*, 37:15320–15355, 2024.

Hongyao Tang, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Glen Berseth. Mitigating plasticity loss in continual reinforcement learning by reducing churn. *arXiv preprint arXiv:2506.00592*, 2025.

Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

A Theoretical Properties: Proofs and Extensions

A.1 Proof of Theorem 1

Starting from the first-order output change (Eq. 8):

$$\Delta h(\mathbf{x}) = z(\mathbf{x}) \odot \Delta h_{\text{proj}}(\mathbf{x}) + \Delta z(\mathbf{x}) \odot [h_{\text{proj}}(\mathbf{x}) - h_{\text{ref}}(\mathbf{x})]. \tag{12}$$

By the triangle inequality and the property $\|a \odot b\|_2 \leq \|a\|_\infty \|b\|_2$:

$$\|\Delta h(\mathbf{x})\|_2 \leq \|z(\mathbf{x})\|_\infty \|\Delta h_{\text{proj}}(\mathbf{x})\|_2 + \|\Delta z(\mathbf{x})\|_\infty \cdot D(\mathbf{x}), \tag{13}$$

where $D(\mathbf{x}) = \|h_{\text{proj}}(\mathbf{x}) - h_{\text{ref}}(\mathbf{x})\|_2$.

By Lipschitz continuity assumptions:

$$\|\Delta h_{\text{proj}}(\mathbf{x})\|_2 \leq L_p \|\Delta \theta_p\|_2, \tag{14}$$

$$\|\Delta z(\mathbf{x})\|_\infty \leq L_z \|\Delta \theta_z\|_2. \tag{15}$$

Therefore:

$$\|\Delta h(\mathbf{x})\|_2 \leq \|z(\mathbf{x})\|_\infty L_p \|\Delta \theta_p\|_2 + L_z \|\Delta \theta_z\|_2 D(\mathbf{x}). \tag{16}$$

Since $z(\mathbf{x}) \in (0,1)^h$ due to the sigmoid, $\|z(\mathbf{x})\|_\infty < 1$, completing the proof. □

A.2 Corollary: Bounded Churn

Consider a sequence of updates $\{\theta_t\}_{t=0}^T$ under learning rate η. By Theorem 1, each step incurs an output change bounded by

$$\|\Delta h_t(\mathbf{x})\|_2 \leq \eta \left(\|z(\mathbf{x})\|_\infty L_p \|\nabla_{\theta_p} \mathcal{L}_t\|_2 + L_z \|\nabla_{\theta_z} \mathcal{L}_t\|_2 D(\mathbf{x}) \right). \tag{17}$$

Accumulating over t and applying Cauchy–Schwarz yields

$$\|h_{\theta_T}(\mathbf{x}) - h_{\theta_0}(\mathbf{x})\|_2 \leq BT, \tag{18}$$

for a constant B depending on η, L_p, L_z, and the gradient magnitudes. Squaring and taking expectation over $\mathcal{D}_{\text{ref}}$ gives

$$\mathcal{C}_T \leq B^2 T^2, \tag{19}$$

establishing bounded churn.

A.2.1 Proof of Theorem 2

We restate the NTK for InterpLayers (Eq. 11):

$$N_{\text{IL}}(x_i, x_j) = (z(x_i) \odot z(x_j))^\top N_{\text{proj}}(x_i, x_j) + N_{\text{interp}}(x_i, x_j).$$

Step 1: PSD decomposition. Both N_{proj} and N_{interp} are positive semidefinite (PSD) Gram matrices of gradients. Therefore, their weighted sum is also PSD. The interpolation kernel can be written explicitly as

$$N_{\text{interp}}(x_i, x_j) = \left\langle \nabla_{\theta_z} z(x_i) \odot (h_{\text{proj}}(x_i) - h_{\text{ref}}(x_i)),\ \nabla_{\theta_z} z(x_j) \odot (h_{\text{proj}}(x_j) - h_{\text{ref}}(x_j)) \right\rangle,$$

which is PSD by construction.

Step 2: Rank contribution of interpolation. If $z(x)$ collapses to a constant vector c across all samples, then the interpolation gradients vanish (since $\nabla_{\theta_z} z(x)$ is zero almost everywhere after saturation). In that case N_{interp} degenerates to zero. Conversely, if $\text{Var}[z_{(k)}(x)] > 0$ for at least one coordinate k, then the interpolation gradients differ across samples, producing at least one non-zero eigenvalue in N_{interp}.

Step 3: Rank inequality. Because $N_{\text{IL}} = \underbrace{(z_i \odot z_j)^\top N_{\text{proj}}}_{\text{possibly degenerate}} + N_{\text{interp}}$ and both terms are PSD, we have

$$\text{rank}(N_{\text{IL}}) \;\geq\; \text{rank}(N_{\text{interp}}).$$

This follows from the fact that adding a PSD matrix cannot reduce the rank contribution of another PSD component.

Step 4: Conclusion. Thus, provided $z(x)$ is not constant across all samples, the interpolation term guarantees a non-zero rank contribution. In particular, $\text{rank}(N_{\text{IL}})$ cannot collapse below $\text{rank}(N_{\text{interp}})$, ensuring gradient diversity even if N_{proj} degenerates.

□

A.3 Weight Norm Regularization (Extended)

Although not central to the main text, we note that interpolation gates implicitly regularize effective weight norms. Define the effective contribution at time t as

$$\|\mathbf{W}_{\text{eff}}(t)\|_F^2 \leq \mathbb{E}_x[\|z_t(x)\|_\infty^2] \cdot \|\mathbf{W}_{p,t}\|_F^2 + \|\mathbf{W}_{z,t}\|_F^2, \tag{20}$$

Since $\|z_t(x)\|_\infty \leq 1$, the contribution of $\mathbf{W}_{p,t}$ is strictly bounded relative to its norm. This prevents unbounded growth of effective weights even when $\|\mathbf{W}_{p,t}\|_F \to \infty$. Empirical evidence for this effect is provided in Section 4.3.

B Training Details

Training protocol. The RL policy is trained using Proximal Policy Optimization (PPO) (Schulman et al., 2017). Following the framework described in (Juliani & Ash, 2024), we report the performance at the epoch level and mark task boundaries at each distribution shift. In our setup, one epoch denotes the following steps: (i) we collect buffer_size = 1024 transitions across 11 parallel environments, then (ii) perform 3 PPO passes with minibatch size set to 64. The PPO hyperparameters are set as follows: $\gamma = 0.99, \lambda = 0.95$, clip = 0.2, entropy = 0.02, learning rate = 5e-4. We train the policy for 50,000 epochs, with distribution shifts at every 5000 epochs, i.e., 5000, 10000, ..., 45000.

C Computational Cost Comparison

Table 1 presents the parameter counts and forward-pass FLOPs for the main architectures evaluated in this paper. We count one multiple-accumulate as a single FLOP. The conv128 encoder requires nearly 35% more computational load than the InterpConv64 variant used in our InterpLayers, despite the latter showing higher performance in later experiments.

Encoder Variant	Params (M)	FLOPs (M)
Conv128 (standard)	1.98	63.5
Conv128 (standard+SSP+LN)	1.98	67.5
InterpConv64 (fullinterp)	1.52	50.8
InterpConv64 (conv-only)	0.99	49.7

Table 1: Parameter counts and forward FLOPs per inference step.

D Soft Shrink-Perturb with LayerNorm (SSP+LN)

We implement soft shrink-perturb following (Juliani & Ash, 2024), where after each optimizer step we apply the shrink and perturb update to the parameters x:

$$x_{\text{new}} = \alpha\, x_{\text{current}} + \beta\, x_{\text{init}}, \quad x_{\text{init}} \sim \mathcal{D}_{\text{init}}. \tag{21}$$

with $\alpha = 0.999999$ and $\beta = 0.000001$

In SSP+LN, this continuous update is combined with LayerNorm (Ba et al., 2016) applied throughout training.

E Details on the NTK Computation

We measure the empirical NTK of the policy and value PPO heads throughout training. The goal is to understand if InterpLayers maintain gradient diversity compared to baselines.

Reference batch. At initialization, we collect a reference batch of observations from the training environments. To ensure diversity, samples are drawn from multiple environments using three strategies: (i) fresh resets, (ii) short random walks, and (iii) mid-episode states. We target 200 samples, capped at 50 per environment. If fewer samples are available, we fall back to a minimum of 16.

NTK matrix construction. For each reference observation x, we compute the gradient of the PPO objective with respect to all trainable parameters of the policy (and optionally value) networks:

$$g(x) = \nabla_\theta \mathcal{L}_{\text{PPO}}(x).$$

The empirical NTK matrix is then

$$K_{ij} = \langle g(x_i), g(x_j) \rangle.$$

Gradients are computed in mini-batches, and the resulting kernel is assembled as a Gram matrix of dimension up to 200×200.

Effective rank and spectra. We report the *effective rank* of the NTK, defined as the participation ratio:

$$r_{\text{eff}} = \frac{(\sum_k \lambda_k)^2}{\sum_k \lambda_k^2},$$

where λ_k are eigenvalues of K. This value measures the number of significant gradient directions. We also record the minimum eigenvalue and condition number to diagnose degeneracy.

Logging frequency and cost. NTK metrics are computed every 100 epochs, aligned with test evaluations. Each computation uses the fixed reference batch from initialization and incurs approximately 10–15% additional runtime overhead relative to standard PPO training.

Implementation. The NTK logger is implemented in PyTorch and integrated into the PPO training loop. It automatically detects whether gating is enabled and saves all metrics to disk in JSON/CSV format for post-hoc analysis.

F Details on the Churn Computation

We measure churn from the encoder outputs using a fixed reference batch that is stored at initialization. At epoch t, churn is defined as the mean squared deviation of the current encoder representations from the initial ones:

$$C_t = \mathbb{E}_{x \sim \mathcal{D}} \left[\|h_t(x) - h_0(x)\|_2^2 \right], \tag{22}$$

where $h_t(x)$ denotes the encoder representation of input x at epoch t, and $h_0(x)$ the corresponding representation at initialization. We also log the first- and second-order finite differences of C_t over epochs.

G Visualization of the Distribution Shifts of ProcGen Tasks

Sample visualizations for the three ProcGen coinrun tasks evaluated in this paper is shown in Figure 5. For **permute**, a fixed random pixel permutation is applied per shift. Given the change in the entire state space, this task requires robust feature relearning. For **window**, the environment is resampled with a different random seed to create other environments. Finally, the expand tasks increase the number of training environments from 100 to 1000 across 9 shifts. This characteristic evaluates the generalization capabilities of the trained policy.

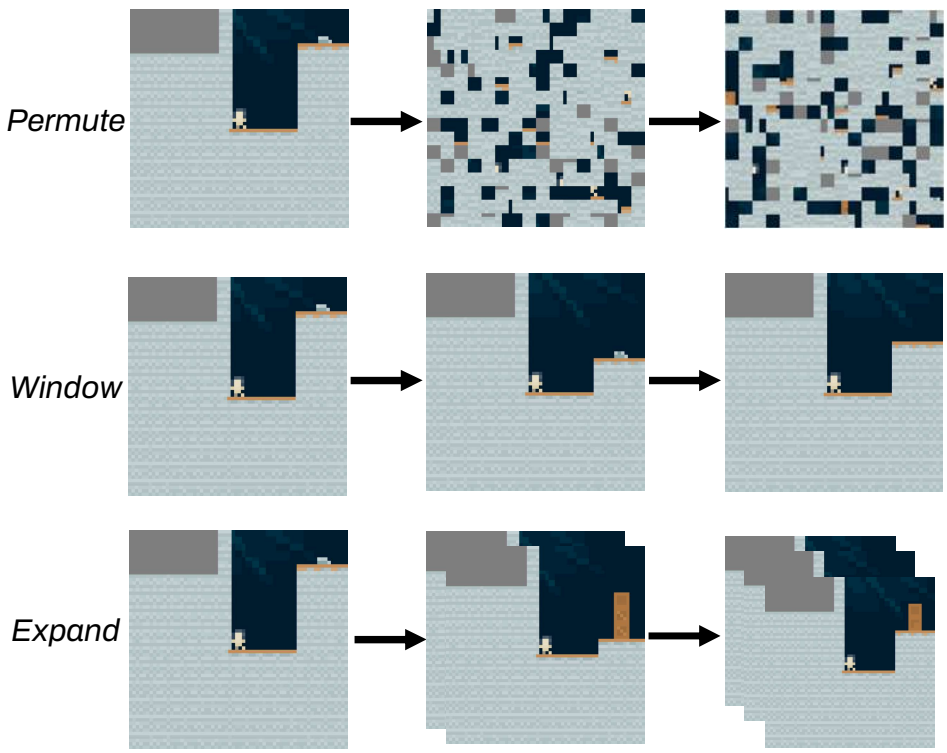


Figure 5: **Visualization of the distribution shifts used in ProcGen coinrun.** Each panel shows the transformation applied at the shift boundaries (every 5,000 epochs): **Permute** applies a fixed random pixel permutation per shift; **Window** resamples the environment seed to generate new levels; **Expand** increase the number of training environments from 100 to 1000 across 9 shifts. Visual representations of the environments ae shown.

H TOTAL REWARD PERFORMANCE ON PROCGEN TASKS

The raw episodic returns obtained by the different methods in the three ProcGen coinrun tasks are shown in Figures 6, 7, and 8. In these figures, it is observed that the higher number of raw episodic returns is obtained by the SSP+LN methods and is followed closely by the two best InterpLayer methods, conv-only dropout and fullinterp dropout. For the three tasks, a standard neural network achieves the worst final raw episodic reward.

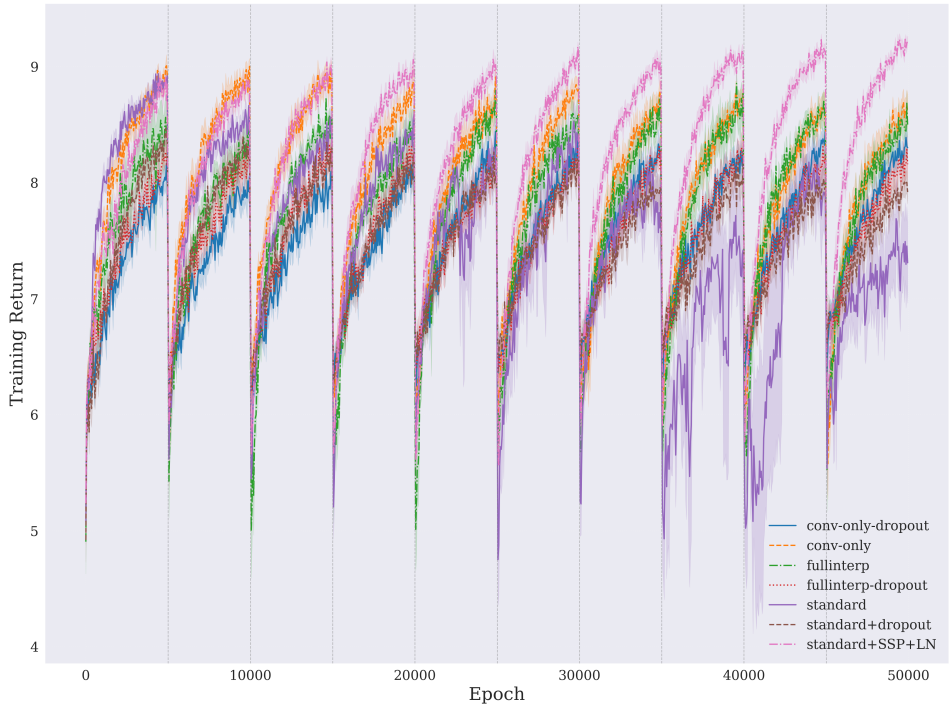


Figure 6: **Raw episodic returns in the permute setting.** Curves show the per-epoch mean reward for the seven network variants reported in Figure 2a; vertical lines indicate the shift boundaries; results are averaged across 10 seeds.

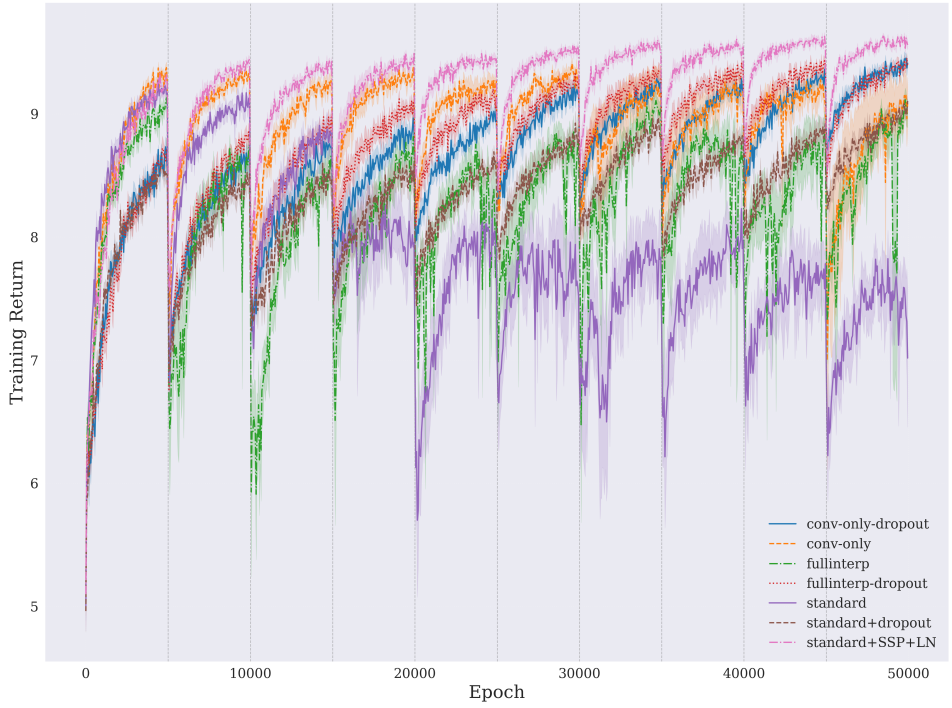


Figure 7: **Raw episodic returns in the window setting.** Curves show the per-epoch mean reward for the seven network variants reported in Figure 2b; vertical lines indicate the shift boundaries; results are averaged across 10 seeds.

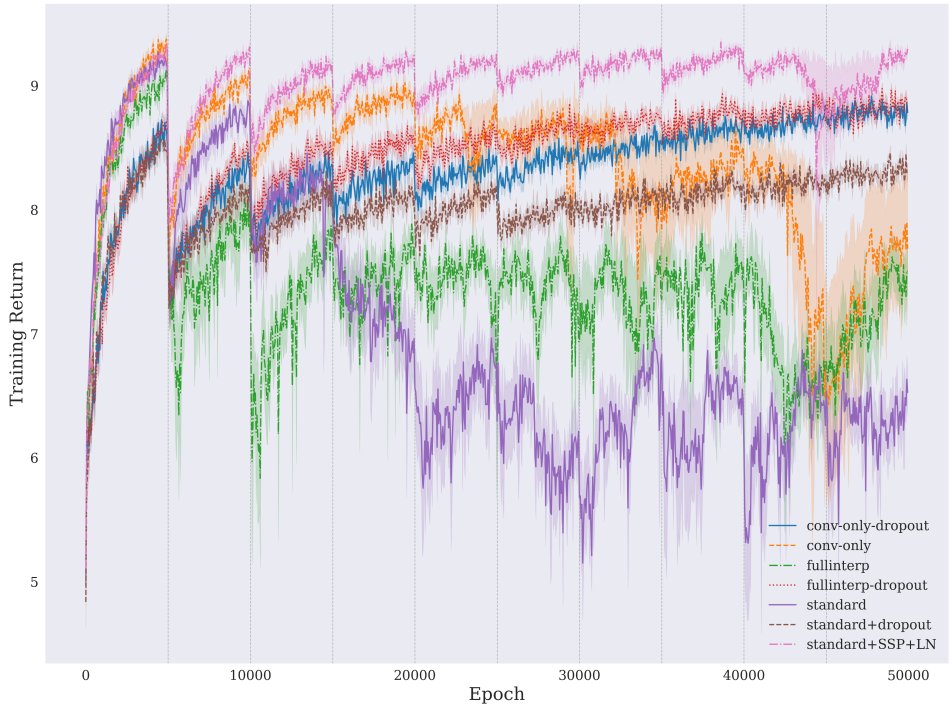


Figure 8: **Raw episodic returns in the expand setting.** Curves show the per-epoch mean reward for the seven network variants reported in Figure 2c; vertical lines indicate the shift boundaries; results are averaged across 10 seeds.