# Rethinking the Role of Dynamic Sparse Training for Scalable Deep Reinforcement Learning

**Guozheng Ma**[1], **Lu Li**[2], **Zilin Wang**[3], **Haoyu Wang**[1], **Shengchao Hu**[1], **Leszek Rutkowski**[4], **Dacheng Tao**[1]
[1]Nanyang Technical University, [2]Mila - Quebec AI Institute & Université de Montréal,
[3]University of Oxford, [4]The AGH University of Krakow

## Abstract

Scaling neural networks has driven breakthrough advances in machine learning, yet this paradigm fails in deep reinforcement learning (DRL), where larger models often degrade performance due to unique optimization pathologies such as plasticity loss. While recent works show that dynamically adapting network topology during training can mitigate these issues, existing studies have three critical limitations: (1) applying uniform dynamic training strategies across all modules despite encoder, critic, and actor following distinct learning paradigms, (2) focusing evaluation on basic architectures without clarifying the relative importance and interaction between dynamic training and architectural improvements, and (3) lacking systematic comparison between different dynamic approaches including sparse-to-sparse, dense-to-sparse, and sparse-to-dense. Through comprehensive investigation across modules and architectures, we reveal that dynamic sparse training strategies provide module-specific benefits that complement the primary scalability foundation established by architectural improvements. We finally distill these insights into Module-Specific Training (MST), a practical framework that further exploits the benefits of architectural improvements and demonstrates substantial scalability gains across diverse RL algorithms without algorithmic modifications.

## 1. Introduction

Scaling neural networks has emerged as a fundamental driver of progress in modern machine learning, with larger models consistently delivering superior performance. This fundamental scaling paradigm, however, fails dramatically in deep reinforcement learning (DRL), where increased model size frequently degrades rather than improves performance [49, 50]. This limited scalability of DRL models can be largely attributed to severe optimization pathologies that emerge during training [49, 52]. Primary pathological behaviors manifest as capacity collapse [41] and plasticity loss [53, 61], whereby networks progressively lose their ability to learn from newly collected experiences, resulting in catastrophic learning inefficiency or even stagnation [14, 45]. Moreover, these phenomena intensify as model size increases, essentially limiting the scaling potential that has been driving advances in other domains like natural language processing and computer vision [8, 9]. Consequently, developing RL-tailored DL mechanisms and employing targeted interventions to prevent networks from falling into these pathological states has become necessary for achieving truly efficient and scalable DRL algorithms [50, 58].

Network architectures and training strategies form the core mechanisms of deep learning (DL) systems.

Recent advances in addressing the unique pathologies in DRL have emerged along two complementary directions: architectural innovations that enhance DRL networks while maintaining conventional dense training, and training strategy improvements on top of standard network architectures [32]. Over the past few years, both directions have advanced significantly and contributed essential insights toward DRL scalability. • Architectural improvements including better normalization [5, 43], residual connections [50, 66], and activation function variants [37, 54] have each shown performance gains on their own. Recent approaches such as BRO [50] and Simba [34] combine these elements effectively, achieving strong scaling performance without modifying underlying RL algorithms. • For training strategies, dynamic alternatives to conventional static training have demonstrated significant benefits in DRL. Approaches including dynamic sparse training[1] [4, 26, 62], dense-to-sparse pruning [8], and sparse-to-dense growth [39] differ in specific implementation but share a common insight: ***introducing network sparsity and dynamicity during training helps DRL networks avoid optimization pathologies and achieve better scalability.***

Despite growing recognition of the potential benefits of dynamic sparse training in DRL, its widespread adoption remains limited due to three critical drawbacks in existing research. • **First**, existing work typically applies uniform dynamic training strategies across all network components, despite the fact that encoder, critic, and actor modules follow fundamentally different learning paradigms with distinct optimization characteristics [34, 45]. For example, pruning techniques that enhance scalability in value-based DRL fail when applied to actor-critic algorithms due to their distinct gradient dynamics [8]. This evidence suggests that ***dynamic training effectiveness should be evaluated separately for each module type***. • **Second**, evaluations of dynamic training have been conducted predominantly on basic architectures like vanilla MLPs, with limited investigation of their effects on recent advances in DRL network design [26, 39]. While dynamic training methods demonstrate benefits for basic networks suffering from severe pathologies, their interaction with architectural improvements remains unclear: ***which approach has greater impact on scalability, and whether dynamic training methods provide additional benefits beyond well-designed architectures***. • **Third**, there have been a severe lack of comparison between different dynamic training approaches, including sparse-to-sparse, dense-to-sparse, and sparse-to-dense paradigms. Although they all introduce network dynamicity, each creates distinct sparsity patterns throughout training that affect learning dynamics differently [3, 40]. Hence, ***a comprehensive comparison is crucial to provide practical dynamic sparse training implementation guidance for DRL algorithm design***.

To address these gaps, we conduct a systematic empirical investigation examining how dynamic training effectiveness varies across different DRL modules and architectural configurations. Specifically, we base our experiments and analysis on MR.Q [24], which provides explicit decoupling of module training methods: encoders learn through self-supervised dynamics prediction, critics learn through temporal difference learning, and actors learn via policy gradients. To investigate architecture-training interactions, we implement six network variants with different combinations of normalization, activation functions, and residual connections (detailed in Section 2.2). Building on the RigL [18] framework that prunes connections by weight magnitude while growing new ones based on gradient information, we implement three dynamic training regimes: ◆ Dynamic Sparse Training (DST), ◀ Sparse-to-Dense (S2D), and ▶ Dense-to-Sparse (D2S), while also establishing ● Dense Training and ■ Static Sparse Training (SST) with one-shot pruning as baselines for fair comparison. Our findings clarify the effectiveness and limitations of dynamic sparse training, providing clear implementation guidance. Key insights include:

[1] We use 'dynamic sparse training' as an umbrella term for methods that introduce sparsity and dynamically adapt network topology, including but not limited to the specific DST algorithm.

1. Interaction between DRL network architecture and training strategy:
   - Architectural improvements provide the primary foundation for DRL scalability, with dynamic training strategies offering complementary but secondary benefits.
   - Dynamic sparse training significantly helps basic networks suffering from pathologies, but provides diminishing returns when applied to architecturally advanced networks.
   - The combination of well-designed architecture and appropriate training strategies yields synergistic scalability benefits that neither approach can achieve alone.
2. Module-specific responses to dynamic sparse training, i.e, network sparsity and dynamicity:
   - Self-supervised encoder can maintain scalability with proper architecture, making dynamic sparse training unnecessary compared to simple dense configurations. (Section 3.1)
   - Critics suffer from severe plasticity loss even with advanced architecture, making dynamic sparse training consistently beneficial compared to static configurations. (Section 3.2)
   - Static sparse training with advanced architecture prevents actors from developing pathologies during scaling, avoiding dynamicity that disrupts policy stability. (Section 3.3)

Integrating these insights, we propose Module-Specific Training (MST) as a systematic framework that assigns tailored training strategies to each DRL module: dense training for the encoder with strong architectural foundations, DST for the critic to address TD learning pathologies, and SST for the actor to balance plasticity with policy stability. When paired with strong architectural foundations, MST enables successful scaling to massive 362M parameter models while maintaining stable learning dynamics and achieving superior sample efficiency. Crucially, MST accomplishes this without modifying underlying RL algorithms, and generalizes effectively across different algorithmic frameworks including SAC and DDPG, establishing a practical pathway toward truly scalable DRL.

## 2. Preliminary

In this section, we establish the foundational elements for our investigation: dynamic sparse training regimes, network architecture variants, and decoupled module training. The comprehensive background on DRL optimization pathologies and their mitigation approaches is provided in Appendix A.

### 2.1. Dynamic Sparse Training Regimes and Static Baselines

To comprehensively evaluate the impact of dynamic training, we implement three distinct dynamic training regimes, alongside static sparse training as a baseline to isolate sparsity effects. We outline key characteristics below, with implementation details and hyperparameters in Appendix B.1.

● **Dense Training**: The conventional approach where all network parameters are initialized and trained throughout the learning process, maintaining full connectivity between neurons.

■ **Static Sparse Training (SST)**: SST [40] applies one-shot random pruning at initialization to create a fixed sparse topology with predefined layer-wise sparsity. Following prior findings that ***Erdős-Rényi (ER)*** initialization outperforms uniform sparsity [26, 46, 62], we adopt ER throughout our experiments.

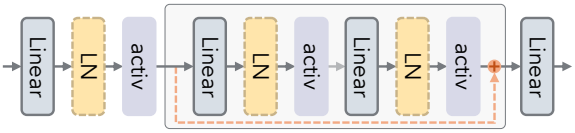
◆ **Dynamic Sparse Training (DST)**: DST maintains constant sparsity while dynamically evolving topology during training. Starting from a randomly pruned network, we implement ***RigL*** [18], which prunes connections based on weight magnitude and grows new ones guided by gradient.

◂ **Sparse-to-Dense Training (S2D)**: S2D begins sparse and gradually increases connectivity, releasing new parameters to maintain learnability [39]. We implement this by extending RigL's framework with a decreasing sparsity schedule from high initial values to zero.

▸ **Dense-to-Sparse Training (D2S)**: D2S progressively prunes connections starting from a dense network, potentially helping dormant neurons escape optimization traps [8]. Our D2S implementation follows the RigL framework with a gradual magnitude-based pruning schedule.

### 2.2. Network Architecture Advances Tailored for Deep RL

The DRL community traditionally treated neural networks primarily as function approximators, directing research efforts toward core RL challenges such as exploration [11] and value overestimation [22] rather than network architecture design. Hence, for a long period, most DRL research defaulted to basic MLPs [23], adding only a few convolutional layers when processing visual observations [67]. Only when network pathologies such as plasticity loss were recently identified as critical bottlenecks did research begin to focus on architectural improvements to address these unique issues.




| | Type 1 | Type 2 | Type 3 | Type 4 | Type 5 | Type 6 |
|---|---|---|---|---|---|---|
| Activation Function | ReLU | ELU | ReLU | ELU | ReLU | ELU |
| Layer Normalization | w/o | w/o | w/ | w/ | w/ | w/ |
| Residual Connection | w/o | w/o | w/o | w/o | w/ | w/ |

**Figure** 1: Network architecture variants for our investigation, incorporating three key architectural improvements: activation functions, layer normalization, and residual connections.

Three architectural elements have emerged as particularly significant: • **First**, after ReLU was identified as a key contributor to dormant neurons [61], numerous activation function variants have been proposed to mitigate this pathology, such as CReLU [1] and AID [54]. In this paper, we employ ELU [12] as a representative improvement in this direction. • **Second**, various normalization techniques have demonstrated effectiveness, including Spectral Normalization [6], Batch Normalization [5], Hyperspherical Normalization [35], and the widely adopted Layer Normalization [42, 43]. • **Third**, beyond merely using ResNet as a visual encoder [17], BRO [50] and Simba [34] have demonstrated that incorporating residual blocks is crucial for scalable policy and value networks. To investigate interactions between dynamic sparse training regimes and architectural design, we implement six network variants with progressive combinations of these elements, as shown in Figure 1. Detailed architectural designs and analyses can be found in Appendix B.2.

### 2.3. Decoupled Module Training via Specific Learning Paradigms

To systematically investigate how distinct RL components respond to dynamic training, we adopt MR.Q [24], which explicitly decouples training of the encoder, critic, and actor modules. Each module follows fundamentally different learning paradigms, resulting in unique optimization characteristics that demand tailored network properties: • **Encoder** learns through **self-supervised** dynamics prediction, where the loss is composed of three components: reward prediction, next-state dynamics prediction, and terminal state prediction [23, 24]. By unrolling these signals over a short horizon, the encoder develops rich representations grounded in environment dynamics rather than non-stationary value targets, transforming diverse observations into a unified latent space [57]. • **Critic** trains through **temporal difference (TD) learning** with bootstrapped targets based on TD3[22]. This paradigm is especially prone to plasticity loss [45, 61], wherein bootstrapping from non-stationary targets leads to unbounded

parameter growth and eventual capacity collapse [42]. • **Actor** employs **deterministic policy gradient** to optimize parameters toward maximizing expected returns [59]. This direct optimization approach differs from both self-supervised and TD learning, creating effective but potentially inflexible behaviors that depend on stable gradient pathways [49]. Algorithm details are provided in Appendix B.3.

These fundamental differences in learning paradigms have led to numerous observed implementation distinctions across modules. For example, actor networks can tolerate significantly higher sparsity levels than critics when reducing computational budgets [26], while scaling critic networks yields substantially greater benefits when pursuing RL scaling laws [34]. This suggests value functions represent more complex mappings requiring greater capacity. In visual RL settings with all three modules, critics have been identified as suffering the most severe plasticity loss, becoming the primary performance bottleneck [45]. Further evidence comes from contrastive RL [19], which demonstrates markedly superior scalability when critics are trained with InfoNCE objectives rather than TD losses [66]. Most relevant to our study, dense-to-sparse pruning methods that significantly enhance scalability in value-based RL (approximated as critic-only systems) fail to transfer successfully to actor-critic algorithms [8]. These documented differences strongly motivate our approach of investigating the distinct impacts of different training regimes across the encoder, critic, and actor modules separately.

## 3. Investigation: How DST Strategies Impact Deep RL Scalability?

To systematically understand how network dynamicity impacts deep RL scalability, we conduct a ***module-specific investigation*** following the natural dependency chain: encoders train independently, critics build on encoders, and actors depend on both. We progressively examine each module in isolation, comprehensively comparing five training strategies (● Dense, ■ SST, ◆ DST, ◀ S2D, ▶ D2S) and their ***interactions with architectural variants*** to reveal fundamental scaling principles.

**Setup.** We conduct our experiments using MR.Q [24] on state-based *Dog Run* and *Humanoid Run*, widely recognized as the most challenging tasks in the DMC suite [63] due to their complex dynamics and high-dimensional action spaces. For each module, we first examine the impact of different training strategies and architectural variants at default network sizes, then conduct scalability analysis by scaling networks by factors of 3× and 5× in both width and depth, creating models spanning three orders of magnitude in parameter count. We maintain a consistent sparsity level of 0.6 across all sparsity-based training regimes (60% of parameters pruned). Specifically, SST and DST maintain constant 0.6 sparsity throughout training, D2S gradually increases from 0 to 0.6, and S2D decreases from 0.6 to 0. Unless stated otherwise, all experimental results in this paper are averaged over 8 random seeds.

### 3.1. Encoder: Network Architecture Dominates Self-Supervised Learning

We begin our investigation with the encoder that learns through self-supervised dynamics prediction. To isolate the effects of encoder design, we vary only encoder architecture and training regime, while maintaining the original MR.Q architecture and size for critic (Type 4) and actor (Type 3) networks with standard dense training, which ensures reliable and consistent operation.

**Comparison at Base Scale.** We evaluate all combinations of six architecture types and five training regimes for the encoder module at its default model size. Figure 2 reveals two key findings: • Networks without layer normalization (LN) fail to train effectively, and ELU significantly outperforms ReLU for encoder learning. • In contrast, all dynamic training regimes only improve weaker architectures while

failing to overcome fundamental architectural limitations.

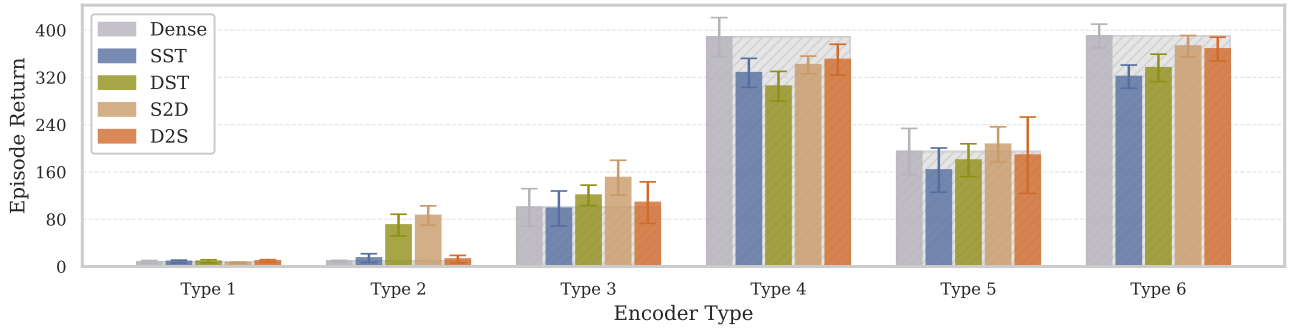



**Figure** 2: Impact of encoder architecture types and training regimes on agent performance at default size. The results demonstrate that architectural improvements (progression from Type 1 to Type 6) yield substantially greater performance gains than differences between training regimes.

To understand why architectures without layer normalization fail completely, we examine the representational capacity of the zsa network in the encoder module using the Stable-Rank (*SRank*) [33], which measures the effective dimensionality of learned features. As shown in Figure 3, Type 2 networks suffer catastrophic representational collapse with rapidly declining *SRank*, while simply adding layer normalization (Type 4) prevents this collapse entirely. This fundamental difference reveals that preventing representational collapse in self-supervised RL depends not only on algorithmic techniques such as stop-gradient operations [51] and EMA-updated target encoders [57], but also critically on appropriate DL mechanisms. Specifically, architectural improvements yield substantially greater benefits than introducing dynamic sparse training regimes.

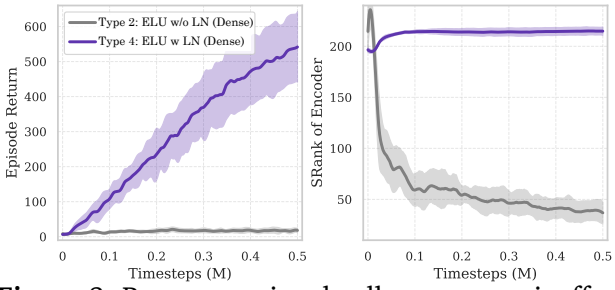



**Figure** 3: Representational collapse causes ineffective training without layer normalization.

**Scaling Behavior.** Based on base scale findings, we examine representative architectures (Types 2, 4, 6) across three orders of magnitude. As parameter counts increase, residual connections become essential for large encoder scalability. For Type 4 networks, dynamic sparse training regimes improve scalability, though these benefits stem primarily from network sparsity. Most notably, Type 6 networks

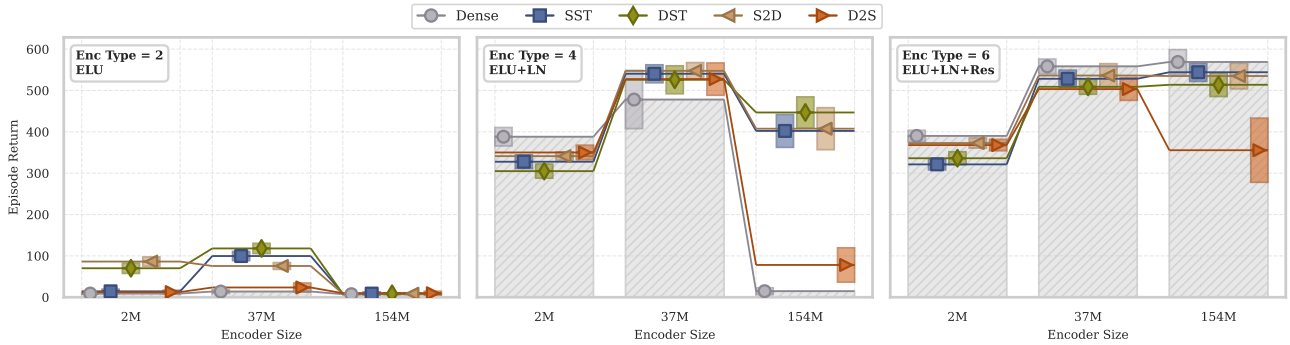



**Figure** 4: Scaling behavior of agent performance as encoder parameter count increases across three orders of magnitude. While introducing network sparsity and dynamicity can sometimes outperform dense training, architectural design remains the fundamental determinant of scalability potential.

maintain consistent scalability even at the largest scale with basic dense training alone. This progression demonstrates that architectural advances create inherently better optimization properties, enabling effective encoder scaling that dynamic sparse training strategies alone cannot achieve.

**Takeaway:** For encoders trained through self-supervised dynamics, architecture design inherently determines both representation quality and scalability, surpassing all dynamic training benefits.

### 3.2. Critic: Dynamic Sparse Training Enhances TD-based Value Learning

Critics trained through TD learning suffer unique pathologies due to bootstrapping from non-stationary targets [42], motivating our investigation of how dynamic training approaches mitigate these specific issues. Based on Section 3.1 findings, we employ a stronger encoder network (Type 6, 37M parameters) for critic module investigation while maintaining default actor configurations.

**Comparison at Base Scale.** Unlike encoders where architectural differences prove dramatic, Figure 5 shows that critic networks achieve more similar performance across architectures at default scale when supported by strong encoder representations. Training regime differences also remain minimal at this scale. This finding suggests that powerful encoder representations effectively capture approximate linear relationships between state-action pairs and their values, significantly reducing value learning complexity [23, 56, 69]. Among architectural elements, layer normalization stands out as uniquely beneficial across all training approaches, consistent with recent research highlighting its critical role in mitigating plasticity loss [34] and reducing value overestimation [49] in TD-based critic training.

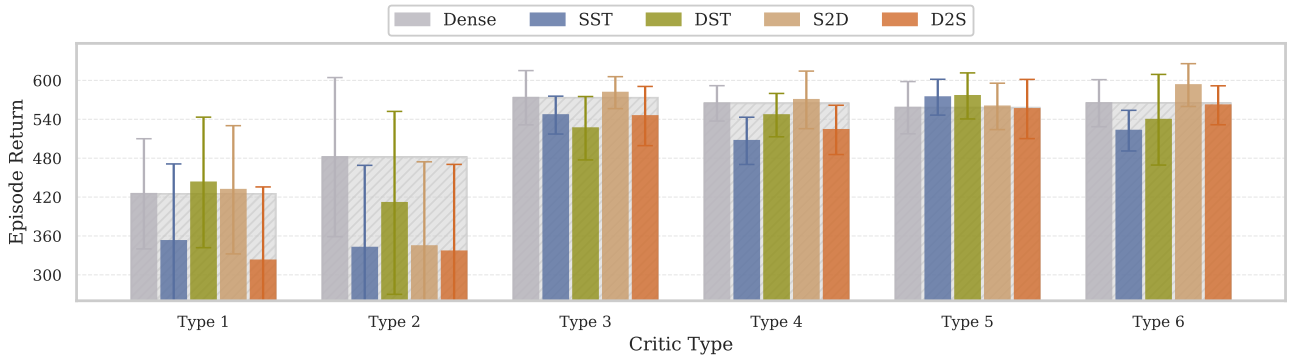



**Figure** 5: Comparison of critic architecture types and training regimes at default network size. With a strong encoder providing effective state and action representations, critic networks reliably achieve stable optimization. Beyond layer normalization, other architectural advances and training regime variations show limited impact on critic performance at this default scale.

**Scaling Behavior.** As critic networks scale up, Figure 6 reveals pronounced performance differences between network architectures and training strategies. Dense networks suffer severe degradation across most architectures, with catastrophic collapse in networks without residual connections (Types 1-4). Even advanced architectural combinations cannot overcome fundamental scaling barriers in dense training. In contrast, training strategies introducing network sparsity and dynamicity substantially enhance critic scalability across architectures. Among these approaches, ◆ DST demonstrates the most consistent improvements by maintaining constant sparsity while enabling dynamic topology adaptation, whereas ▶ D2S and ◀ S2D alter sparsity levels during training. These findings reveal a key insight about TD-based value learning: neither architectural innovations nor training regimes alone can overcome the critic

scaling barrier. Unlocking full scaling potential of the critic module demands the right combination of both architectural design and training strategy.

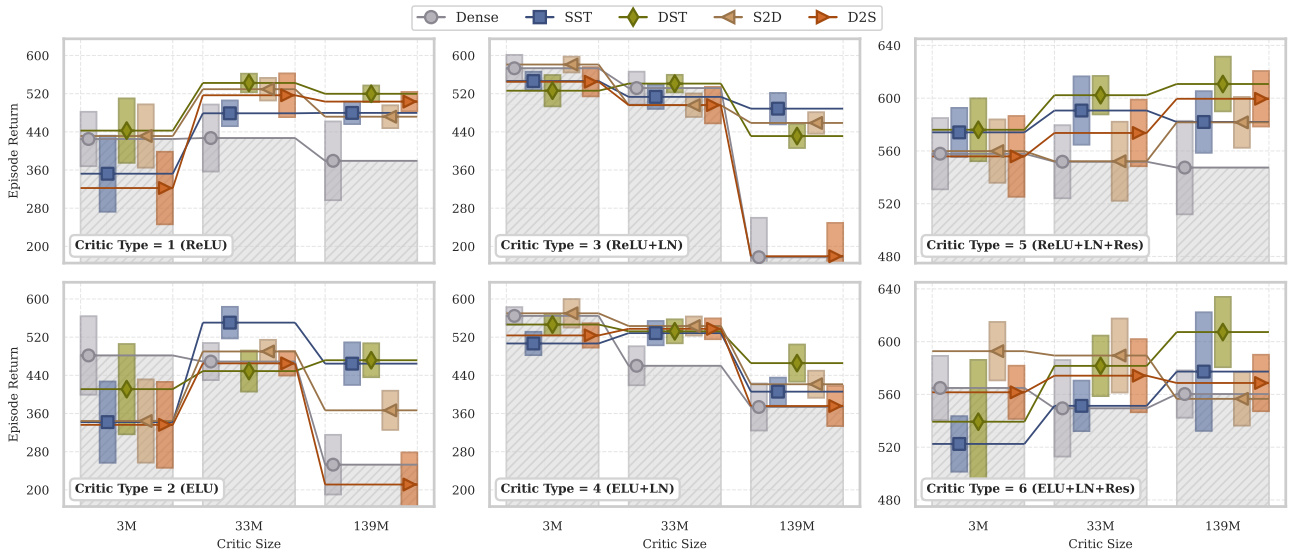



**Figure** 6: Dynamic training approaches, particularly DST, consistently outperform dense and static sparse baselines. Results demonstrate that unlocking the full potential of scaled critic networks requires the combination of advanced architectures with appropriate dynamic training regimes.

To further understand the mechanisms behind these scaling behaviors, we examine critic pathologies using two key metrics: *SRank* [33] for representational capacity and *Dormant Ratio* [61] for plasticity loss. Figure 7 demonstrates that dynamic training effectively mitigates plasticity loss in large-scale critics, aligning with previous findings on pruning's benefits for value-based RL scalability [8]. Meanwhile, network sparsity unlocks greater representational capacity, but only when combined with residual connections. Without these architectural safeguards, critics suffer catastrophic representational collapse even without severe plasticity loss. This dual mechanism explains why DST, combining network sparsity with dynamic topology adaptation, achieves superior scalability of the critic module.

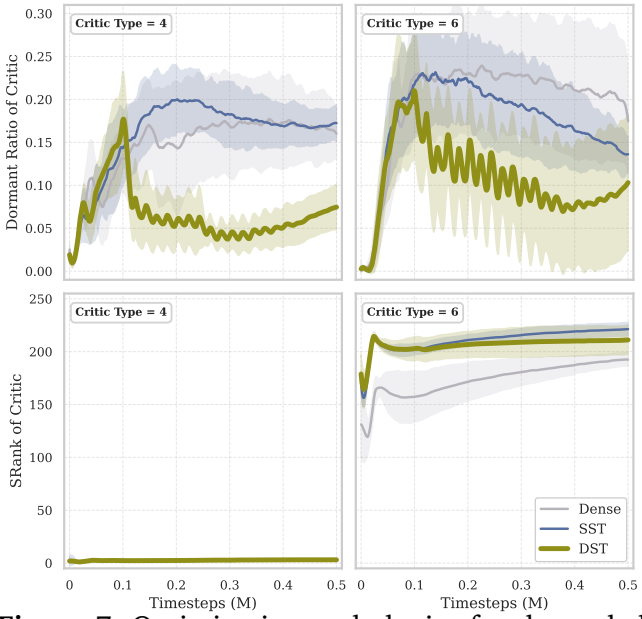



**Figure** 7: Optimization pathologies for the scaled critic networks with 139M parameters (scale=5).

**Takeaway:** DST can notably enhance the scalability of TD-learned critics by mitigating plasticity loss while maintaining representational capacity. However, full scalability of critics requires the combination of appropriate dynamic sparse training with powerful architectural foundations.

### 3.3. Actor: Static Sparsity Outperforms Dynamic Training in Policy Scaling

Unlike critics and encoders, actor networks directly optimize deterministic policy gradients, creating distinct stability requirements [64]. While dynamic training strategies proved beneficial for critic scaling, we investigate whether these same approaches effectively scale policy networks. For actor evaluation, we maintain a scaled Type 6 encoder (37M) and default Type 4 critic (3M) to isolate actor-specific effects while ensuring stable representation and value learning.

**Comparison at Base Scale.** As shown in Figure 8, LN consistently benefits actor networks across all training regimes. Interestingly, switching from ReLU to ELU activation functions significantly degrades actor performance, contrasting sharply with our findings for encoders and critics where ELU proved beneficial. This suggests that the feature sparsity induced by ReLU better supports policy gradient optimization, despite potential concerns about dormant neurons. In addition, training strategy differences remain minimal at the default scale across all architectural variants.

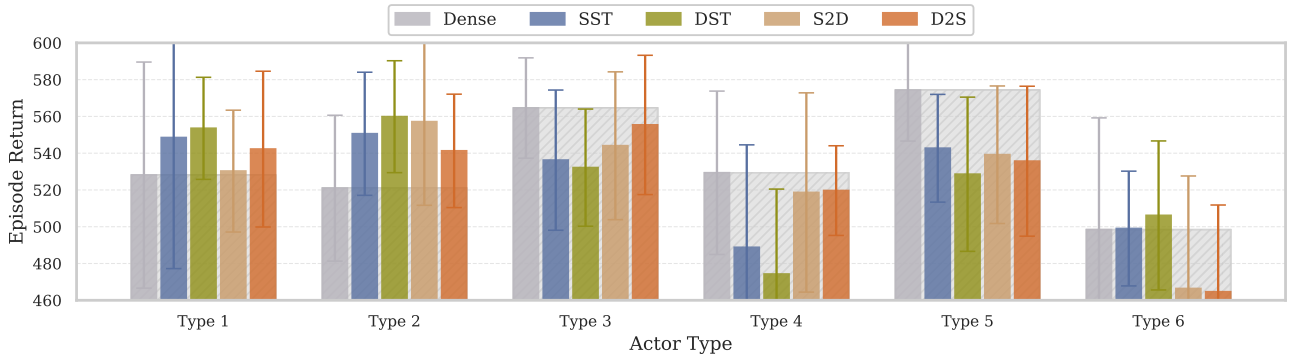



**Figure** 8: Impact of actor architecture types and training regimes at default network size. Layer normalization provides consistent benefits, while switching activation function from ReLU to ELU significantly reduces performance. Dynamic training approaches help networks without normalization but offer no advantages when applied to more advanced architectures.

**Scaling Behavior.** Figure 9 reveals distinct scaling patterns for actor networks. First, normalization and residual connections remain essential for achieving scalable actors. However, unlike critics where dynamic training worked best, SST consistently outperforms all dynamic approaches for scaled actor networks. This advantage becomes more pronounced at larger scales, where the three dynamic training approaches exhibit more severe training instability.

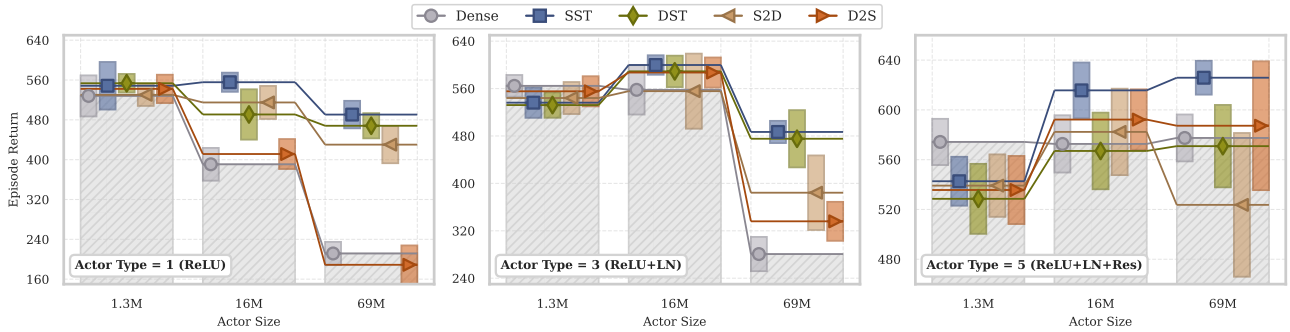



**Figure** 9: Actor network scaling performance across architectures and training regimes. Normalization and residual connections are critical for preventing collapse in large-scale actor networks. With this architectural foundation, SST consistently enables further effective scaling for policy networks.

To elucidate why SST outperforms dynamic approaches, Figure 10 examines plasticity loss at the largest scale. Type 3 networks without residual connections exhibit substantial dormant neuron proportions under dense training, while adding residual connections (Type 5) largely prevents this phenomenon, confirming that actors experience less severe plasticity issues than critics. Crucially, SST achieves the most consistent dormant neuron reduction across architectures while preserving training stability. Dynamic training approaches, while effective for critic plasticity, yield minimal benefits for actors and introduce detrimental training instability that disrupts policy optimization. This phenomenon explains SST's superiority and aligns with prior work [8] demonstrating that pruned networks effective in value-based RL fail to transfer successfully to actor-critic architectures.

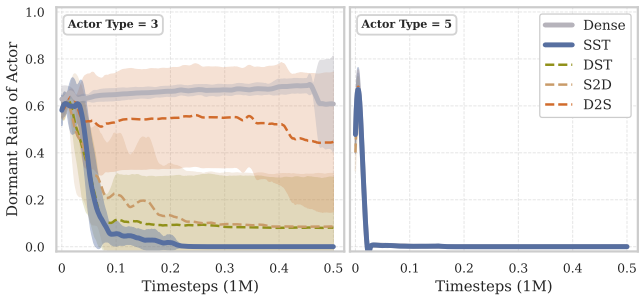



**Figure** 10: Network sparsity reduces dormant ratio and mitigates plasticity loss in scaled actors.

**Takeaway:** SST with well-designed architecture most effectively prevents actors from developing optimization pathologies during scaling by providing an ideal balance between reduced plasticity loss and stable gradient pathways essential for policy optimization.

### 3.4. Investigation Remark

This investigation reframes our understanding of how dynamic sparse training impacts scalable DRL. Consistent with prior work [8, 39], DST can significantly prevent basic MLP networks from developing severe optimization pathologies during scaling. However, these benefits are substantially overshadowed by architectural improvements alone. This demonstrates that ***network architecture establishes the fundamental optimization dynamics most crucial for stable scaling***, aligning with recent scalable DRL advances that primarily stem from architectural innovations such as BRO [50] and Simba series [34, 35].

Our second key contribution to current understanding is that while architecture is primary, architectural improvements alone can not suffice for maximizing DRL scalability. Instead, ***targeted training strategies with network sparsity and dynamicity can extract further scaling potential from strong architectural foundations***. Achieving this requires ***module-specific training strategies*** that align with each component's distinct learning paradigms and optimization characteristics.

## 4. Method: Module-Specific Training for Scalable Deep RL

Our investigation in Section 3 provides clear guidance for achieving scalable DRL through tailoring DL mechanisms without modifying underlying RL algorithms. This guidance operates on two essential levels: • **First,** establishing strong architectural foundations represents the fundamental prerequisite. As demonstrated throughout our investigation and consistent with prior studies [34, 35, 50, 66], these elements establish the optimization dynamics that are most crucial for stable scaling. • **Second, and equally important,** we propose **Module-Specific Training (MST)** as a systematic framework that assigns targeted training strategies based on the distinct learning paradigms of each module. Specifically, MST applies dense training for the encoder, DST for the critic to address the most severe TD learning pathologies, and SST for the actor to balance the plasticity and stability during policy optimization.

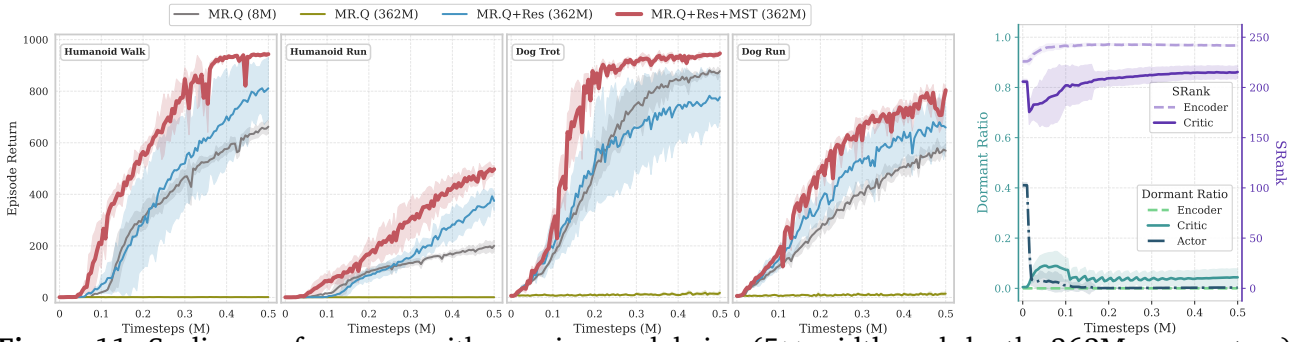



**Figure** 11: Scaling performance with massive model size (5× width and depth, 362M parameters). Baseline MR.Q fails completely at this scale, while adding residual connections effectively prevents catastrophic collapse. However, MST proves essential for unlocking substantial additional scalability gains on top of strong architectural foundations. The right panel shows MST maintains healthy optimization dynamics with high SRank and minimal dormant ratios even at this massive scale.

To validate the effectiveness and necessity of MST, we first scale MR.Q to massive size, as shown in Figure 11. Naively scaling up DRL models without tailored architecture and training strategies results in catastrophic failure, unlike the successful scaling observed in language and vision domains. These findings reveal that scalable DRL demands both strong architectural foundations and strategic MST approaches. We conduct ablations for MST design choices, as shown in Figure 12. ER initialization outperforms random initialization, and RigL [18] proves superior to Sparse Evolutionary Training (SET) [47] for the critic due to gradient-based growth. Consistent with [46], increasing sparsity benefits massive DRL models while excessive sparsity potentially causes instability.

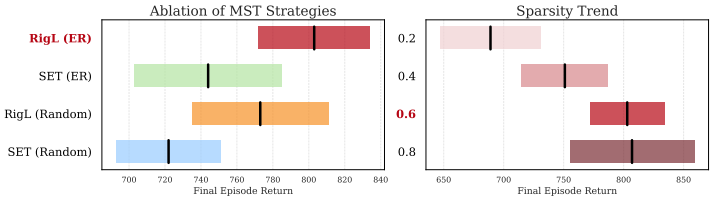



**Figure** 12: Ablation study on MST implementation details in MR.Q+Res+MST (362M) on Dog Run.

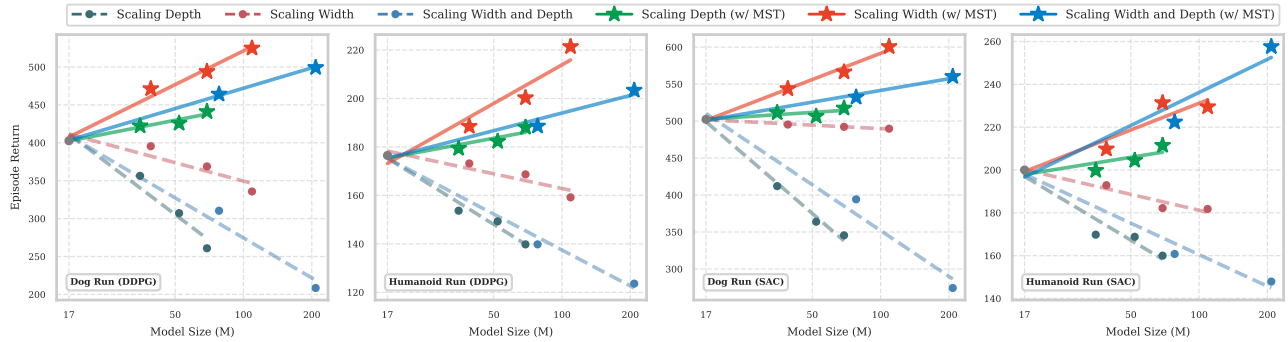



**Figure** 13: MST generalizes to DRL algorithms without decoupled encoders. On SAC and DDPG with Simba architectures, standard scaling approaches (dashed lines) lead to performance degradation, while MST (solid lines) enables successful scaling across different dimensions and tasks.

To demonstrate MST's broader applicability beyond algorithms with decoupled encoders, we evaluate its effectiveness on the widely-used SAC [27] and DDPG [38]. Using Simba [34] as our strong architectural foundation and following sparsity configurations from [46], we examine scaling trends across width, depth, and both dimensions simultaneously. As shown in Figure 13, naive scaling consistently degrades performance, while MST achieves stable scaling gains.

In addition, Figure 14 validates the importance of targeted training strategy assignment: introducing dynamicity benefits the critic network while proving detrimental to the actor network. We also validate MST's effectiveness in MR.SAC, which combines model-based representation learning from MR.Q with the SAC algorithm, with detailed results provided in Appendix E.

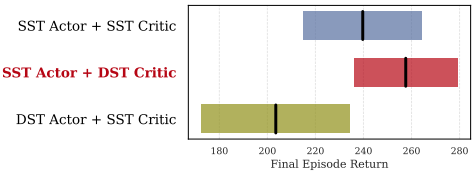



**Figure** 14: Validation of MST necessity on Humanoid Run (SAC, 207M).

## 5. Conclusion and Discussion

Dynamic sparse training (DST) was initially developed to reduce training costs with minimal performance degradation. However, DRL researchers discovered sparse networks surprisingly outperformed dense counterparts [26, 62]. Recent studies further revealed that such dynamic training strategies can alleviate the scaling barriers caused by DRL's unique optimization pathologies [8, 39]. Concurrently, scalable DRL has achieved significant progress, particularly through architectural advances. Against this backdrop, a critical question arises: ***what role does DST play now?*** Our investigation addresses this question and reveals two key insights. First, DST alone cannot match architectural improvements; instead, it serves as a complementary mechanism that unlocks additional potential. Second, effective dynamic training demands module-specific approaches: different RL modules require distinct training strategies tailored to their unique learning paradigms. We distill these insights into an empirical MST framework and validate its effectiveness in improving scalability across multiple base RL algorithms.

**Lessons.** Beyond technical implementation details, our investigation further reveals broader principles that fundamentally reshape how we understand and implement RL-tailored DL mechanisms.

• *Lesson 1:* ***Unlocking the full potential of DRL requires carefully designed DL mechanisms tailored to the unique challenges of RL optimization landscapes.*** Our experiments show that standard neural architectures developed for supervised learning fail to accommodate the non-stationarity and bootstrapping inherent to RL. Specialized elements like layer normalization and tailored training strategy deliver improvements beyond what algorithmic refinements alone can achieve. This confirms that DRL optimization pathologies require distinct solutions beyond conventional DL practices. Beyond our focus on architecture and training regimes, other DL components such as optimizers [7, 15, 25, 48] and regularization techniques [10] also represent promising solutions.

• *Lesson 2:* ***Different learning paradigms exhibit distinct optimization characteristics that demand module-specific architectural and training considerations.*** Despite operating within a unified system, different DRL modules follow distinct learning objectives that create fundamentally different optimization dynamics. For instance, TD-learned critics suffer more severely from plasticity loss than other modules, while actors require stable gradient pathways for effective policy learning. Hence, effective RL-tailored mechanisms must recognize that different components within the same agent require distinct architectural and training approaches.

• *Lesson 3:* ***With basic optimization pathologies in standard DRL tasks now effectively addressed, it is time to advance toward more challenging real-world scenarios.*** Our results in Section 4 demonstrate that combining advanced architectures with MST enables massive model scaling while maintaining healthy optimization dynamics. However, real-world applications present fundamentally greater challenges such as larger action spaces [21], multi-task conflicts [30] and continual adaptation requirements [2, 16]. Therefore, future research should focus on developing more advanced RL-tailored DL mechanisms that preserve the benefits of scaling under these demanding optimization challenges.

## References


[1] 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*, pages 620–636. PMLR, 2023. 4, 20

[2] David Abel, André Barreto, Benjamin Van Roy, Doina Precup, Hado P van Hasselt, and Satinder Singh. A definition of continual reinforcement learning. *Advances in Neural Information Processing Systems*, 36:50377–50407, 2023. 12

[3] Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *arXiv preprint arXiv:2112.15579*, 2021. 2, 21

[4] Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, Amy Zhang, Alessandro Sordoni, and Doina Precup. Efficient reinforcement learning by discovering neural pathways. *Advances in Neural Information Processing Systems*, 37:18660–18694, 2024. 2, 21

[5] Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PczQtTsTIX. 2, 4

[6] Nils Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning with spectral normalization. *Advances in neural information processing systems*, 34:8242–8255, 2021. 4

[7] Roger Creus Castanyer, Johan Obando-Ceron, Lu Li, Pierre-Luc Bacon, Glen Berseth, Aaron Courville, and Pablo Samuel Castro. Stable gradients for stable learning at scale in deep reinforcement learning. *arXiv preprint arXiv:2506.15544*, 2025. 12

[8] Johan Samir Obando Ceron, Aaron Courville, and Pablo Samuel Castro. In value-based deep reinforcement learning, a pruned network is a good network. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=seo9V9QRZp. 1, 2, 4, 5, 8, 10, 12, 20, 21, 23

[9] Johan Samir Obando Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=X9VMhfFxwn. 1, 20, 21

[10] Wesley Chung, Lynn Cherif, Doina Precup, and David Meger. Parseval regularization for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 37:127937–127967, 2024. 12

[11] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. *Advances in Neural Information Processing Systems*, 32, 2019. 4

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015. 4

[13] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019. 22

[14] Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OpC-9aBBVJe. 1

[15] Benjamin Ellis, Matthew T Jackson, Andrei Lupu, Alexander D Goldie, Mattie Fellows, Shimon Whiteson, and Jakob Foerster. Adam on local time: Addressing nonstationarity in rl with relative adam timesteps. *Advances in Neural Information Processing Systems*, 37:134567–134590, 2024. 12

[16] Mohamed Elsayed and A. Rupam Mahmood. Addressing loss of plasticity and catastrophic forgetting in continual learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sKPzAXoylB. 12

[17] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018. 4, 20

[18] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pages 2943–2952. PMLR, 2020. 2, 3, 11, 21, 22

[19] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 35603–35620, 2022. 5

[20] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024. 20

[21] Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. In *International Conference on Machine Learning*, pages 3040–3051. PMLR, 2020. 12

[22] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 4

[23] Scott Fujimoto, Wei-Di Chang, Edward J. Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For SALE: State-action representation learning for deep reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xZvGrzRq17. 4, 7, 28

[24] Scott Fujimoto, Pierluca D'Oro, Amy Zhang, Yuandong Tian, and Michael Rabbat. Towards general-purpose model-free reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=R1hIXdST22. 2, 4, 5, 21, 26

[25] Alexander D Goldie, Chris Lu, Matthew T Jackson, Shimon Whiteson, and Jakob Foerster. Can learned optimization make reinforcement learning less difficult? *Advances in Neural Information Processing Systems*, 37:5454–5497, 2024. 12, 20

[26] Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pages 7766–7792. PMLR, 2022. 2, 3, 5, 12, 21

[27] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 11

[28] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 28

[29] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023. 28

[30] Viraj Joshi, Zifan Xu, Bo Liu, Peter Stone, and Amy Zhang. Benchmarking massively parallelized multi-task reinforcement learning for robotics tasks. In *Reinforcement Learning Conference*, 2025. URL https://openreview.net/forum?id=z0MM0y20I2. 12

[31] Arthur Juliani and Jordan T Ash. A study of plasticity loss in on-policy deep reinforcement learning. *arXiv preprint arXiv:2405.19153*, 2024. 20

[32] Timo Klein, Lukas Miklautz, Kevin Sidak, Claudia Plant, and Sebastian Tschiatschek. Plasticity loss in deep reinforcement learning: A survey. *arXiv preprint arXiv:2411.04832*, 2024. 2, 20

[33] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. 6, 8, 20

[34] Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. *arXiv preprint arXiv:2410.09754*, 2024. 2, 4, 5, 7, 10, 11, 20, 24

[35] Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *arXiv preprint arXiv:2502.15280*, 2025. 4, 10

[36] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *ArXiv e-prints*, pages arXiv–1607, 2016. 20

[37] Alex Lewandowski, Dale Schuurmans, and Marlos C. Machado. Plastic learning with deep fourier features. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=NIkfix2eDQ. 2

[38] TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 11

[39] Jiashun Liu, Johan Obando-Ceron, Aaron Courville, and Ling Pan. Neuroplastic expansion in deep reinforcement learning. *arXiv preprint arXiv:2410.07994*, 2024. 2, 4, 10, 12, 21, 22

[40] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=VBZJ_3tz-t. 2, 3, 22

[41] Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZkC8wKoLbQ7. 1, 20

[42] 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*, pages 23190–23211. PMLR, 2023. 4, 5, 7, 20

[43] Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *arXiv preprint arXiv:2407.01800*, 2024. 2, 4, 35

[44] 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. 20

[45] Guozheng Ma, Lu Li, Sen Zhang, Zixuan Liu, Zhen Wang, Yixin Chen, Li Shen, Xueqian Wang, and Dacheng Tao. Revisiting plasticity in visual reinforcement learning: Data, modules and training stages. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=0aR1s9YxoL. 1, 2, 4, 5, 20

[46] Guozheng Ma, Lu Li, Zilin Wang, Li Shen, Pierre-Luc Bacon, and Dacheng Tao. Network sparsity unlocks the scaling potential of deep reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=mIomqOskaa. 3, 11

[47] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018. 11, 21, 22

[48] Aneesh Muppidi, Zhiyu Zhang, and Heng Yang. Fast trac: A parameter-free optimizer for lifelong reinforcement learning. *Advances in Neural Information Processing Systems*, 37:51169–51195, 2024. 12

[49] Michal Nauman, Michał Bortkiewicz, Piotr Miłoś, Tomasz Trzcinski, Mateusz Ostaszewski, and Marek Cygan. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=5vZzmCeTYu. 1, 5, 7, 20, 35

[50] Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. In *Advances in Neural Information Processing Systems*, 2024. 1, 2, 4, 10, 20, 25

[51] Tianwei Ni, Benjamin Eysenbach, Erfan SeyedSalehi, Michel Ma, Clement Gehring, Aditya Mahajan, and Pierre-Luc Bacon. Bridging state and history representations: Understanding self-predictive RL. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ms0VgzSGF2. 6

[52] Evgenii Nikishin. *Parameter, experience, and compute efficient deep reinforcement learning*. PhD thesis, Université de Montréal, 2024. 1, 20

[53] 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*, pages 16828–16847. PMLR, 2022. 1, 20

[54] Sangyeon Park, Isaac Han, Seungwon Oh, and Kyung-Joong Kim. Activation by interval-wise dropout: A simple way to prevent neural networks from plasticity loss. *arXiv preprint arXiv:2502.01342*, 2025. 2, 4

[55] Joan Puigcerver, Carlos Riquelme Ruiz, Basil Mustafa, Cedric Renggli, André Susano Pinto, Sylvain Gelly, Daniel Keysers, and Neil Houlsby. Scalable transfer learning with expert models. In *International Conference on Learning Representations*, 2020. 20

[56] Aidan Scannell, Kalle Kujanpää, Yi Zhao, Mohammadreza Nakhaei, Arno Solin, and Joni Pajarinen. iqrl–implicitly quantized representations for sample-efficient reinforcement learning. *arXiv preprint arXiv:2406.02696*, 2024. 7

[57] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uCQfPZwRaUu. 4, 6

[58] Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pages 30365–30380. PMLR, 2023. 1, 20

[59] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014. 5

[60] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. *arXiv preprint arXiv:2106.04217*, 2021. 21

[61] Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pages 32145–32168. PMLR, 2023. 1, 4, 8, 20, 24

[62] Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. RLx2: Training a sparse deep reinforcement learning model from scratch. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DJEEqoAq7to. 2, 3, 12, 21, 22

[63] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. 5

[64] Ahmed Touati, Amy Zhang, Joelle Pineau, and Pascal Vincent. Stable policy optimization via off-policy divergence regularization. In *Conference on Uncertainty in Artificial Intelligence*, pages 1328–1337. PMLR, 2020. 9

[65] Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. *arXiv preprint arXiv:2105.01648*, 2021. 21

[66] Kevin Wang, Ishaan Javali, MichaŁ Bortkiewicz, Benjamin Eysenbach, et al. 1000 layer networks for self-supervised rl: Scaling depth can enable new goal-reaching capabilities. *arXiv preprint arXiv:2503.14858*, 2025. 2, 5, 10

[67] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=_SJ-_yyes8. 4

[68] Mingqi Yuan, Qi Wang, Guozheng Ma, Bo Li, Xin Jin, Yunbo Wang, Xiaokang Yang, Wenjun Zeng, and Dacheng Tao. Plasticine: Accelerating research in plasticity-motivated deep reinforcement learning. *arXiv preprint arXiv:2504.17490*, 2025. 20

[69] Ruijie Zheng, Xiyao Wang, Yanchao Sun, Shuang Ma, Jieyu Zhao, Huazhe Xu, Hal Daumé III, and Furong Huang. Taco: Temporal latent action-driven contrastive loss for visual reinforcement learning. *Advances in Neural Information Processing Systems*, 36:48203–48225, 2023. 7

The appendix is divided into several sections, each giving extra information and details.

# A. Related Work and Background

This section examines the key research areas that inform our work. We first explore the distinctive optimization pathologies that emerge specifically in deep reinforcement learning (DRL) environments. We then analyze the fundamental scaling challenges that have limited DRL model expansion and survey recent innovations aimed at overcoming these barriers. Finally, we review existing applications of dynamic sparse training in DRL, whether aimed at model compression, training acceleration, or performance enhancement.

## A.1. Optimization Pathologies in DRL

Although deep neural networks have driven remarkable advances in current deep reinforcement learning (DRL) applications, growing evidence indicates that DRL networks are prone to severe optimization pathologies during training [25, 49, 52]. These pathologies emerge from the unique challenges of integrating DL mechanisms with the RL paradigm, presenting distinctive issues not encountered in traditional RL settings. Such difficulties stem from fundamental characteristics of RL that distinguish it from supervised learning: non-stationary data distributions and optimization objectives, as well as the inherent nature of learning through online interactions.

These DRL-specific pathologies have recently been identified and characterized through various phenomena: primacy bias [53, 68], the dormant neuron phenomenon [61], implicit under-parameterization [33], capacity loss [41], and the broader issue of plasticity loss [1, 31, 32]. Although these studies approach the problem from different angles, they converge on a common finding: DRL networks routinely develop severe optimization pathologies during training that fundamentally impair their ability to learn from new experiences. The consequences manifest either as severe sample inefficiency or, in the worst cases, complete learning stagnation [45]. These pathologies manifest through several observable symptoms: a high proportion of inactive neurons [61], reduced effective rank of representational features [33, 41], unbounded growth in parameter norms [42], and increased gradient interference across training samples [44]. Each of these symptoms contributes to the network's diminishing ability to effectively learn and optimize both the policy and value functions.

## A.2. Unique Scaling Barrier in DRL

Using deep neural networks is a key factor in successfully applying reinforcement learning to complex tasks. However, while some recent advances in supervised learning have been driven by scaling up the number of network parameters, a phenomenon commonly referred to as *scaling laws*, it remains challenging to increase the number of parameters in deep reinforcement learning without experiencing performance degradation. Several recent works in DRL have addressed this by scaling up network sizes through various strategies. [58] transitioned from the original CNN architecture to the ResNet-based Impala-CNN architecture [17] and scaled the network width by a factor of 4. Both BRO [50] and SimBa [34] employed deeper networks that incorporate layer normalization [36] and residual connections. [9] incorporated a soft Mixture-of-Experts module [55] into value-based networks, resulting in more parameter-scalable models and improved performance. [20] show that value functions trained using categorical cross-entropy substantially enhance performance and scalability in multiple domains. [8] utilized magnitude pruning on value-based networks, progressively decreasing the number of parameters in dense architectures during training to achieve highly sparse models, leading to improved performance

when scaling network width.

### A.3. Dynamic Sparse Training in DRL

Initial explorations of sparse networks in DRL were primarily motivated by the potential for model compression, aiming to accelerate training and facilitate efficient model deployment [62]. Early explorations of network sparsification in DRL primarily focused on behavior cloning and offline RL settings [3, 65]. In the more challenging context of online RL, [60] explored the application of Sparse Evolutionary Training (SET) [47] and successfully achieved 50% sparsity. However, attempts to increase sparsity beyond this level resulted in significant training instability. Subsequently, [62] enhanced the efficacy of dynamic sparse training through a novel delayed multi-step temporal difference target mechanism and a dynamic-capacity replay buffer, ultimately achieving sparsity levels of up to 95%. [26] conducted a comprehensive investigation and demonstrated that pruning consistently outperforms standard dynamic sparse training methods, such as SET [47] and RigL [18]. DAPD [4] dynamically adjusts network pathways in response to online RL distribution shifts, maintaining effectiveness at high sparsity levels.

Beyond the initial goal of achieving parameter-efficient architectures through sparsity, recent studies have recognized that sparse and adaptive networks can enhance DRL model scalability while mitigating training pathologies such as plasticity loss. For instance, [8] shows that applying gradual magnitude pruning to large models significantly enhances the performance of value-based agents. Similarly, [9] demonstrates that incorporating Soft MoEs into value-based RL networks enables better parameter scaling. Furthermore, Neuroplastic Expansion [39] addresses plasticity challenges by progressively evolving networks from sparse to dense architectures, effectively leveraging increased model capacity. Although these approaches differ in implementation, they all fall under the broader category of dynamic sparse training, where network topology evolves during training.

## B. Implementation Details for Training Methods and Network Architectures

In this section, we provide comprehensive implementation details essential for reproducing our experimental results and understanding the technical foundations of our research. We first present the implementation specifics of both dynamic and static training regimes, including sparsity mechanisms, pruning schedules, and topology adaptation procedures (Section B.1). Next, we detail the architectural specifications for our network variants, analyzing the design considerations behind each component and their interactions (Section B.2). Finally, we describe the implementation and hyperparameter configurations of our base algorithm MR.Q [24], covering the core mechanism that enables module-specific training paradigms in reinforcement learning (Section B.3). These details collectively establish the technical framework upon which our investigation into dynamic training regimes for RL is built.

### B.1. Dynamic and Static Training Implementation Details

#### *B.1.1. Static Sparse Training (SST) with One-Shot Random Pruning*

Static sparse training with one-shot random pruning creates binary masks for each layer at initialization. These masks define the network's sparse topology and remain unchanged throughout training. For a network with $L$ layers, each layer $l$ has a binary mask $\mathbf{M}^l \in \{0,1\}^{n^l \times n^{l-1}}$, where $n^l$ represents the number of units in layer $l$. The effective weights during both training and inference are calculated as

$\mathbf{W}^l_{\text{eff}} = \mathbf{M}^l \odot \mathbf{W}^l$, where $\odot$ denotes element-wise multiplication.

Random pruning implements a straightforward sampling process that requires only predetermined layer-wise sparsity ratios. Two common approaches for determining these ratios from the overall network sparsity are:

1. ***Uniform***: Each layer $l$ uses the same sparsity ratio $s^l$ equal to the overall network sparsity $S$.

2. ***Erdős-Rényi (ER)***: This method generates sparse masks where the sparsity in each layer $s^l$ is proportional to $1 - \frac{n^{l-1}+n^l}{n^{l-1}n^l}$ for fully-connected layers [47] and to $1 - \frac{n^{l-1}+n^l+w^l+h^l}{n^{l-1}n^l w^l h^l}$ for convolutional layers with kernel dimensions $w^l \times h^l$ [18].

Multiple studies in both supervised learning [40] and reinforcement learning [39, 62] have demonstrated that ER-based initialization consistently outperforms uniform sparsity, particularly at high sparsity levels. Therefore, we adopt ER-based layer-wise sparsity ratios throughout our experiments, with a fixed overall sparsity level of 0.6, which our preliminary experiments identified as an effective balance between network size and performance.

### *B.1.2. Implementation of RigL [18] for Dynamic Sparse Training*

Similar to SST, DST begins with a randomly pruned network using ER. However, while SST maintains a fixed topology throughout training, DST dynamically evolves topology while preserving overall sparsity.

**Algorithm 1** Dynamic Sparse Training based on RigL [18]

$N_l$: Number of parameters in layer $l$
$\theta_l$: Parameters in layer $l$
$M_{\theta_l}$: Sparse mask of layer $l$
$s_l$: Sparsity of layer $l$
$L$: Loss function
$\zeta_t$: Update fraction in training step $t$
**for** each layer $l$ **do**
    $k = \zeta_t(1 - s_l)N_l$
    $\mathbb{I}_{\text{drop}} = \text{ArgTopK}(-|\theta_l \odot M_{\theta_l}|, k)$
    $\mathbb{I}_{\text{grow}} = \text{ArgTopK}_{i \notin \theta_l \odot M_{\theta_l} \setminus \mathbb{I}_{\text{drop}}}(|\nabla_{\theta_l} L, k|)$
    Update $M_{\theta_l}$ according to $\mathbb{I}_{\text{drop}}$ and $\mathbb{I}_{\text{grow}}$
    $\theta_l \leftarrow \theta_l \odot M_{\theta_l}$
**end for**

The core mechanism of RigL [18] involves periodically updating the network topology during training. Every $N$ timesteps, a fraction of connections are modified through a coordinated drop-and-grow process: the lowest magnitude weights are pruned, and an equal number of previously inactive connections with the highest gradient magnitudes are activated [13]. This allows the network to reallocate parameters toward more important connections discovered during the learning process.The fraction of weights

updated at each interval (drop fraction $\zeta_t$) follows a cosine decay schedule throughout training:

$$\zeta_t = \zeta_{\text{final}} + \frac{1}{2}(\zeta_{\text{initial}} - \zeta_{\text{final}})(1 + \cos(\pi \cdot t/T)) \tag{1}$$

where $t$ is the current step and $T$ is the total number of training steps. This schedule enables greater exploration early in training and more exploitation later, helping the network converge to a stable topology. Algorithm 1 presents this process in detail. To ensure consistent comparison across different network architectures and training regimes, we maintain the same overall sparsity level (0.6) throughout our experiments. This level provides a balanced trade-off between parameter efficiency and network expressivity, ensuring that sparsity itself doesn't become a limiting factor in performance.

#### *B.1.3. Extending RigL Framework to Dense-to-Sparse and Sparse-to-Dense*

While Dynamic Sparse Training (DST) maintains constant sparsity, we extend the RigL framework to implement two additional dynamic training regimes with changing sparsity levels: Dense-to-Sparse (D2S) and Sparse-to-Dense (S2D) training. Both approaches follow the same core drop-and-grow mechanism as RigL but incorporate a time-dependent sparsity schedule [8]. The sparsity at training step $t$ is calculated according to:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_{start}}{t_{end} - t_{start}}\right)^{\lambda} \tag{2}$$

where $s_i$ is the initial sparsity, $s_f$ is the final sparsity, $t_{start}$ and $t_{end}$ define the scheduling period, and $\lambda$ controls the non-linearity of the transition.

For Dense-to-Sparse (D2S) training, we set $s_i = 0$ and $s_f = 0.6$, starting with a fully dense network and gradually increasing sparsity. This approach allows the network to establish important connections early in training before progressively removing less useful parameters. The pruning process follows magnitude-based criteria similar to RigL but adjusts the number of pruned connections at each update step to match the target sparsity schedule. Conversely, Sparse-to-Dense (S2D) training sets $s_i = 0.6$ and $s_f = 0$, beginning with a sparse network and gradually increasing connectivity throughout training. This approach follows the neuroplastic expansion hypothesis that releasing additional capacity during learning can help networks escape optimization pathologies while maintaining training efficiency. The growth process continues to use gradient information to identify promising connections, but more connections are added than removed at each update to follow the decreasing sparsity schedule. Both approaches maintain the overall RigL topology update mechanism while adapting the number of parameters added or removed at each update step. For our experiments, we set $\lambda = 2$.

### B.2. Network Architecture Specifications and Analysis

This section discusses key architectural components that significantly impact deep reinforcement learning performance, focusing on activation functions and residual connections that help mitigate optimization pathologies.

### *B.2.1. Activation Function: ReLU vs. ELU*

Activation functions play a critical role in deep neural networks by introducing non-linearity and regulating information flow. In deep reinforcement learning, where optimization pathologies are particularly severe, the choice of activation function can significantly influence network plasticity and learning dynamics.

The Rectified Linear Unit (ReLU) has been the dominant activation function due to its computational efficiency and effectiveness in addressing vanishing gradient problems:

$$\text{ReLU}(x) = \max(0, x) \tag{3}$$

However, ReLU suffers from a fundamental limitation: it produces zero gradients for all negative inputs, leading to the "dormant neuron" problem where neurons become permanently inactive during training [61]. This issue is particularly problematic in DRL, where non-stationary targets can drive many neurons into inactive states.

The Exponential Linear Unit (ELU) addresses this limitation by providing non-zero outputs and gradients for negative inputs:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \tag{4}$$

where $\alpha$ is typically set to 1. The negative saturation in ELU ensures that inactive neurons can be "revived" during training as gradients can still flow through them, making the network more robust to shifting data distributions in reinforcement learning settings.

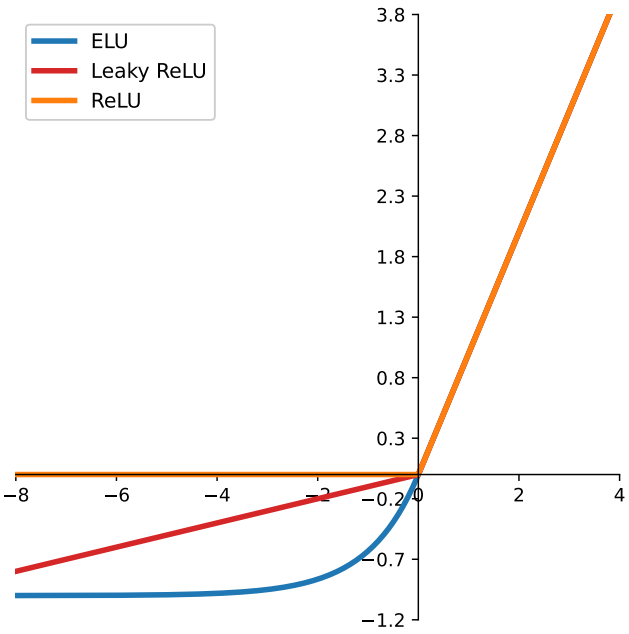



**Figure** 15: Comparison of activation functions: ReLU (red) produces zero output and gradients for negative inputs, while ELU (blue) provides smooth, non-zero negative outputs bounded at $-\alpha$ (typically $-1$). This difference is critical for preventing dormant neurons in reinforcement learning.

Our experiments show that ELU consistently benefits encoder and critic by maintaining higher representational capacity, while surprisingly, ReLU remains more effective for actor networks by providing beneficial sparsity for policy optimization (see Section 3.3).

### *B.2.2. Pre-Layer Residual Connection vs. Post-Layer Residual Connection*

Residual connections are crucial for mitigating gradient flow issues in deep networks, but their implementation details can significantly impact learning dynamics in reinforcement learning. We examine two common approaches to residual connections that differ in their placement within network blocks.

In Pre-Layer Residual Connection (Figure 16 a), the identity shortcut bypasses normalization and activation functions, similar to the approach used in Simba [34]. The forward pass can be expressed as:

$$y = F(x) + x \tag{5}$$

where $F(x)$ represents the entire transformation sequence (normalization, activation, and linear layers). This approach preserves the raw input information through the residual pathway, allowing direct access to unmodified representations.

In contrast, Post-Layer Residual Connection (Figure 16 b) applies normalization and activation before adding the identity shortcut, similar to the implementation in BRO [50]. The computation flow can be described as:

$$y = G(F'(x)) + F'(x) \tag{6}$$

where $F'(x)$ represents the initial transformation (typically normalization and activation), and $G$ represents subsequent linear layers. This approach normalizes inputs before the residual pathway, potentially stabilizing the representation scale throughout the network.

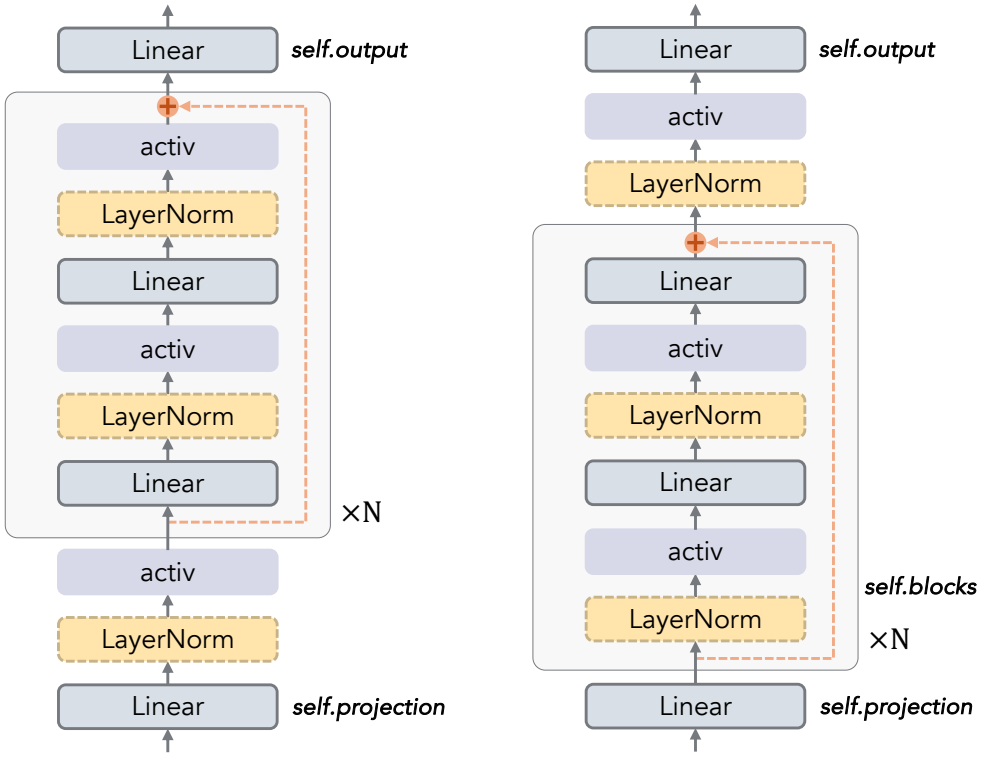



**Figure** 16: Architectural comparison of (a) Pre-Layer Residual Connection, where identity shortcuts bypass normalization and activation, and (b) Post-Layer Residual Connection, where normalization and activation are applied before the residual pathway. The key difference lies in whether raw input information is preserved in the residual path.

Our experimental results (Figure 17) demonstrate that Pre-Layer Residual Connection consistently outperforms Post-Layer implementation across tasks, particularly when scaling network depth. The performance gap widens significantly at larger scales (3x), where unimpeded gradient flow becomes

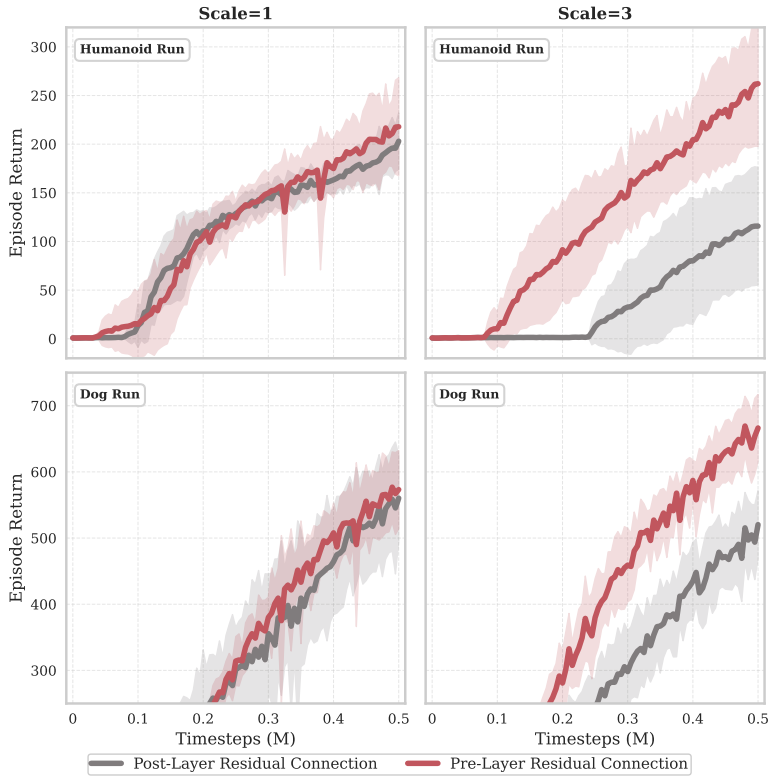



**Figure** 17: Performance comparison between pre-layer and post-layer residual connections on Humanoid Run and Dog Run tasks at default scale and larger scale. pre-layer connections consistently deliver better performance, with advantages becoming more pronounced as model size increases.

increasingly important. Pre-Layer connections maintain stronger learning signals through deep networks by avoiding potential representation bottlenecks that can occur when normalization layers precede the residual pathway.

### B.3. Implementation and Hyperparameters of Base Algorithm MR.Q

In order to achieve a general-purpose model-free reinforcement learning algorithm, MR.Q (Model-based Representations for Q-learning) [24] was proposed with decoupled model-based representation learning. It learns a state encoder $f_\omega(s)$ and a state-action encoder $g_\omega(z_s, a)$ to construct unified latent state embedding $z_s$ and state-action embedding $z_{sa}$, respectively. Following these encoders, decoupled reinforcement learning components including value $Q_\theta(z_{sa})$ and policy $\pi_\phi(z_s)$ are learned in a TD7 manner. The encoders are trained with self-predictive dynamics learning refering from model-based RL. Specifically, at timestep $t-1$, given output state-action embedding $\tilde{z_{sa}}^{t-1} = g_\omega(f_\omega(s^{t-1}), a^{t-1})$, MR.Q try to predict the state embedding $\tilde{z_s}^t$, reward $\tilde{r}^t$, terminal $\tilde{d}^t$ at the next timestep $t$ with a predictor parameterised by $\mathbf{m}$:

$$\tilde{z_s}^t, \tilde{r}^t, \tilde{d}^t = g_\omega(\tilde{z}^{t-1}, a^{t-1})^T \mathbf{m}. \tag{7}$$

And the encoder loss is calculated by a weighted-sum losses of each item along horizon $H_{enc}$:

$$L_{enc}(f_\omega, g_\omega, \mathbf{m}) = \sum_{t=1}^{H_{env}} \lambda_{rwd} L_{rwd}(\tilde{r}^t) + \lambda_{dnm} L_{dnm}(\tilde{z}_s^t) + \lambda_{tmn} L_{tmn}(\tilde{d}^t). \tag{8}$$

Here reward loss adopts the cross entropy between the predicted reward and a two-hot encoding of the true reward:

$$L_{rwd}(\tilde{r}) = \text{CE}(\tilde{r}, \text{Two-Hot}(r)). \tag{9}$$

Dynamics loss and terminal loss are both mean squared error (MSE) over prediction and ground truth:

$$\begin{aligned} L_{dnm}(\tilde{z}_s) &= (\tilde{z}_s - z_s)^2, \\ L_{tmn}(\tilde{d}) &= (\tilde{d} - d)^2. \end{aligned} \tag{10}$$

Although with learned dynamics, as a model-free RL algorithm, MR.Q doesn't perform any planning or rollout simulated trajectories, which avoids introducing extra substantial computation as in model-based methods. Instead, it only utilise reward and dynamics losses as auxiliary tasks to learn the intermediate representation space. The model-based representation learning manners can not only provide richer learning signals, but also help decouple state embedding from original state spaces which might be quite depending on characteristics of different environments, making it possible for MR.Q being an general algorithm working well with different tasks given one set of hyperparameters.

For value function, MR.Q adopts loss in TD3 with a few modification like multi-step returns over a horizon $H_Q$ and replacing MSE with huber loss:

$$L_{value}(\tilde{Q}_i) = \text{Huber}(\tilde{Q}_i, \frac{1}{\overline{r}}(\sum_{t=0}^{H_Q-1} \gamma^t r_t + \gamma^{H_Q} \tilde{Q}'_j)), \quad \tilde{Q}'_j = \overline{r}' \min_{j=1,2} Q_{\theta'_j}(z_{s_{H_Q} a_{H_Q}}), \tag{11}$$

where $\overline{r}$ denotes the reward scale computed over recent expreiences in the replay buffer. It is used to normalise the temporal difference target in the value update, thus stablise the updating process. In addition, it also plays a role to unify value learning in environments with very different reward magnitudes. And $\overline{r}'$ denotes the target reward scale, which helps the target value being stable even the reward scale $\overline{r}$ changes dramatically. $\overline{r}'$ are periodically updated with target network.

Policy is optimised by deterministic policy gradient with additional regularisation penalty on pre-activations $z_\pi$:

$$L_{policy}(a_\pi) = -\frac{1}{2} \sum_{i=\{1,2\}} \tilde{Q}_i(z_{sa_\pi}) + \lambda_{reg} z_\pi^2. \tag{12}$$

The regularization is added to avoid local minima when the reward is sparse.

A comprehensive evaluation of MR.Q on four common RL benchmarks with different observation and action spaces with a fix set of hyperparameters show the compatitive performance of MR.Q over other

state-of-the-art RL algorithms including TD7 [23], DrearmerV3 [28], and TD-MPC2 [29], etc. Notably, Compared with its model-based counterpart, MR.Q has much fewer parameters and much faster training and evaluation speed. Table 1 presents the algorithm hyperparameters from the original MR.Q implementation that remain consistent across all our experiments.

Table 1: **Algorithm Hyperparameters for MR.Q.**

| | Hyperparameter | Value |
|---|---|---|
| | Dynamics loss weight $\lambda_{\text{Dynamics}}$ | 1 |
| | Reward loss weight $\lambda_{\text{Reward}}$ | 0.1 |
| | Terminal loss weight $\lambda_{\text{Terminal}}$ | 0.1 |
| | Pre-activation loss weight $\lambda_{\text{pre-activ}}$ | $1e-5$ |
| | Encoder horizon $H_{\text{Enc}}$ | 5 |
| | Multi-step returns horizon $H_Q$ | 3 |
| TD3 | Target policy noise $\sigma$ | $\mathcal{N}(0, 0.2^2)$ |
| | Target policy noise clipping $c$ | $(-0.3, 0.3)$ |
| LAP | Probability smoothing $\alpha$ | 0.4 |
| | Minimum priority | 1 |
| Exploration | Initial random exploration time steps | 10k |
| | Exploration noise | $\mathcal{N}(0, 0.2^2)$ |
| Common | Discount factor $\gamma$ | 0.99 |
| | Replay buffer capacity | 1M |
| | Mini-batch size | 256 |
| | Target update frequency $T_{\text{target}}$ | 250 |
| | Replay ratio | 1 |

## C. Setup

To implement the six network architecture variants described in Section B.2, we modified the MR.Q network architecture to flexibly incorporate different combinations of normalization, activation functions, and residual connections. We developed a modular implementation that allows us to systematically investigate the impact of each architectural element while maintaining the core functionality of the original algorithm.

First, we implement the `BaseMLPBlock` class, which serves as the fundamental building block in our architecture. This component encapsulates a configurable network block with options for layer normalization and activation functions, enabling systematic investigation of their impact on network performance.

```
class BaseMLPBlock(nn.Module):
    """
    Base block for MLP networks with optional layer normalization
    and activation functions
    """
    def __init__(self, hidden_size: int,
```

```
                 use_layer_norm: bool = True,
                 activation: str = 'elu'):
        super().__init__()

        self.use_layer_norm = use_layer_norm
        self.activ = getattr(F, activation)

        # Two linear layers with optional layer norm
        self.linear1 = nn.Linear(hidden_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, hidden_size)

        self.apply(weight_init)

    def forward(self, x: torch.Tensor):
        # Apply optional LN and activation before first layer
        if self.use_layer_norm:
            y = ln_activ(x, self.activ)
        else:
            y = self.activ(x)

        # First layer
        if self.use_layer_norm:
            y = ln_activ(self.linear1(y), self.activ)
        else:
            y = self.activ(self.linear1(y))

        # Second layer (only linear transformation, without
        # normalization or activation)
        y = self.linear2(y)

        return y
```

The `BlockMLP` module composes multiple `BaseMLPBlock` instances to create complete network architectures. It implements pre-layer residual connections and handles the projections between dimensions. This design allows us to scale network depth while maintaining consistent behavior across different architectural configurations.

```
class BlockMLP(nn.Module):
    """
    MLP with multiple residual blocks using
    pre-layer residual connections
    """
    def __init__(self,
                 input_dim: int,
                 output_dim: int,
                 hidden_dim: int,
                 use_layer_norm: bool = True,
                 activation: str = 'elu',
                 num_blocks: int = 1,
                 use_residual: bool = False):
        super().__init__()

        self.use_residual = use_residual
        self.activ = getattr(F, activation)
        self.use_layer_norm = use_layer_norm

```

```
        # Projection layer - maps input to hidden dimension
        self.projection = nn.Linear(input_dim, hidden_dim)

        # Multiple residual blocks
        self.blocks = nn.ModuleList()
        for _ in range(num_blocks):
            self.blocks.append(BaseMLPBlock(hidden_dim,
                                            use_layer_norm,
                                            activation))

        # Output layer
        self.output = nn.Linear(hidden_dim, output_dim)

        self.apply(weight_init)

    def forward(self, x: torch.Tensor):
        # Initial projection
        x = self.projection(x)

        # Process through blocks with
        # Pre-Layer Residual Connections
        for block in self.blocks:
            # Store input to the block for residual connection
            block_input = x

            # Get output of the block
            # (includes internal LayerNorm + activ operations)
            block_output = block(x)

            # Apply residual connection
            # right after linear transformations
            if self.use_residual:
                x = block_output + block_input
            else:
                x = block_output

        # Apply LN and activation after all blocks are processed
        if self.use_layer_norm:
            x = ln_activ(x, self.activ)
        else:
            x = self.activ(x)

        # Final output layer
        return self.output(x)
```

Finally, we define the `ARCH_CONFIGS` dictionary that specifies the six architecture types corresponding to Figure 1. Each type represents a specific combination of activation function (ReLU vs. ELU), layer normalization (with vs. without), and residual connections (with vs. without), enabling systematic comparison of their effects on performance.

```
# Define network architecture configurations
ARCH_CONFIGS = {
    1: {
        "use_layer_norm": False,
        "activation": "relu",
        "use_residual": False
```

```
    },  # Vanilla MLP (ReLU)

    2: {
        "use_layer_norm": False,
        "activation": "elu",
        "use_residual": False
    },   # Vanilla MLP (ELU)

    3: {
        "use_layer_norm": True,
        "activation": "relu",
        "use_residual": False
    },   # MLP with Layer Norm (ReLU)

    4: {
        "use_layer_norm": True,
        "activation": "elu",
        "use_residual": False
    },    # MLP with Layer Norm (ELU)

    5: {
        "use_layer_norm": True,
        "activation": "relu",
        "use_residual": True
    },    # MLP with Layer Norm and Residual Connections (ReLU)

    6: {
        "use_layer_norm": True,
        "activation": "elu",
        "use_residual": True
    },     # MLP with Layer Norm and Residual Connections (ELU)
}
```

Table 2 details the network-specific hyperparameters. The default configuration shown corresponds to Encoder Type 4 (scale=1), Critic Type 4 (scale=1), and Actor Type 3 (scale=1) in our implementation. These parameters are systematically varied throughout our experiments to evaluate different architecture types and training regimes. For scalability experiments, we multiply the hidden dimensions and block numbers (highlighted in gray) by the corresponding scale factor (3× or 5×).

Table 2: **Network Hyperparameters for MR.Q.** The default configuration corresponds to Encoder Type 4 (scale=1), Critic Type 4 (scale=1), and Actor Type 3 (scale=1) in our experiments. Parameters in shaded rows (hidden dimensions and block numbers) are scaled by the corresponding factor (3× or 5×) in scalability experiments.

| | Hyperparameter | Value |
|---|---|---|
| Encoder Network | Optimizer | AdamW |
| | Learning rate | $1e-4$ |
| | Weight decay | $1e-4$ |
| | $\mathbf{z}_s$ dim | $512$ |
| | $\mathbf{z}_s$ block num | $1$ |
| | $\mathbf{z}_{sa}$ dim | $512$ |
| | $\mathbf{z}_{sa}$ block num | $1$ |
| | $\mathbf{z}_a$ dim (only used within architecture) | $256$ |
| | Hidden dim | $512$ |
| | Activation function | ELU |
| | Layer Normalization | True |
| | Residual Connection | False |
| | Weight initialization | Xavier uniform |
| | Bias initialization | $0$ |
| | Reward bins | $65$ |
| | Reward range | $[-10, 10]$ (effective: $[-22\text{k}, 22\text{k}]$) |
| Value Network | Optimizer | AdamW |
| | Learning rate | $3e-4$ |
| | Hidden dim | $512$ |
| | Block num | $1$ |
| | Activation function | ELU |
| | Layer Normalization | True |
| | Residual Connection | False |
| | Weight initialization | Xavier uniform |
| | Bias initialization | $0$ |
| | Gradient clip norm | $20$ |
| Policy Network | Optimizer | AdamW |
| | Learning rate | $3e-4$ |
| | Hidden dim | $512$ |
| | Block num | $1$ |
| | Activation function | ReLU |
| | Layer Normalization | True |
| | Residual Connection | False |
| | Weight initialization | Xavier uniform |
| | Bias initialization | $0$ |
| | Gumbel-Softmax $\tau$ | $10$ |

## D. Detailed Investigation Results

This section provides comprehensive visualization of experimental results across different network components, architectures, and training regimes. We present detailed performance data from Dog Run and Humanoid Run tasks that supports the main findings discussed in the paper. All experiments were conducted with 8 random seeds to ensure statistical reliability, with error bars representing one standard deviation.

### D.1. Encoder

**Default Scale.** As shown in Figure 2, the average performance comparison between Dog Run and Humanoid Run tasks reveals the striking dominance of architectural design over training regimes for encoder performance. Overall, non-saturating ELU activation function and layer normalization prove essential for developing powerful encoders, while residual connections offer minimal benefits at this scale. In contrast, dynamic training regimes slightly improve weaker architectures but fail to overcome fundamental architectural limitations.

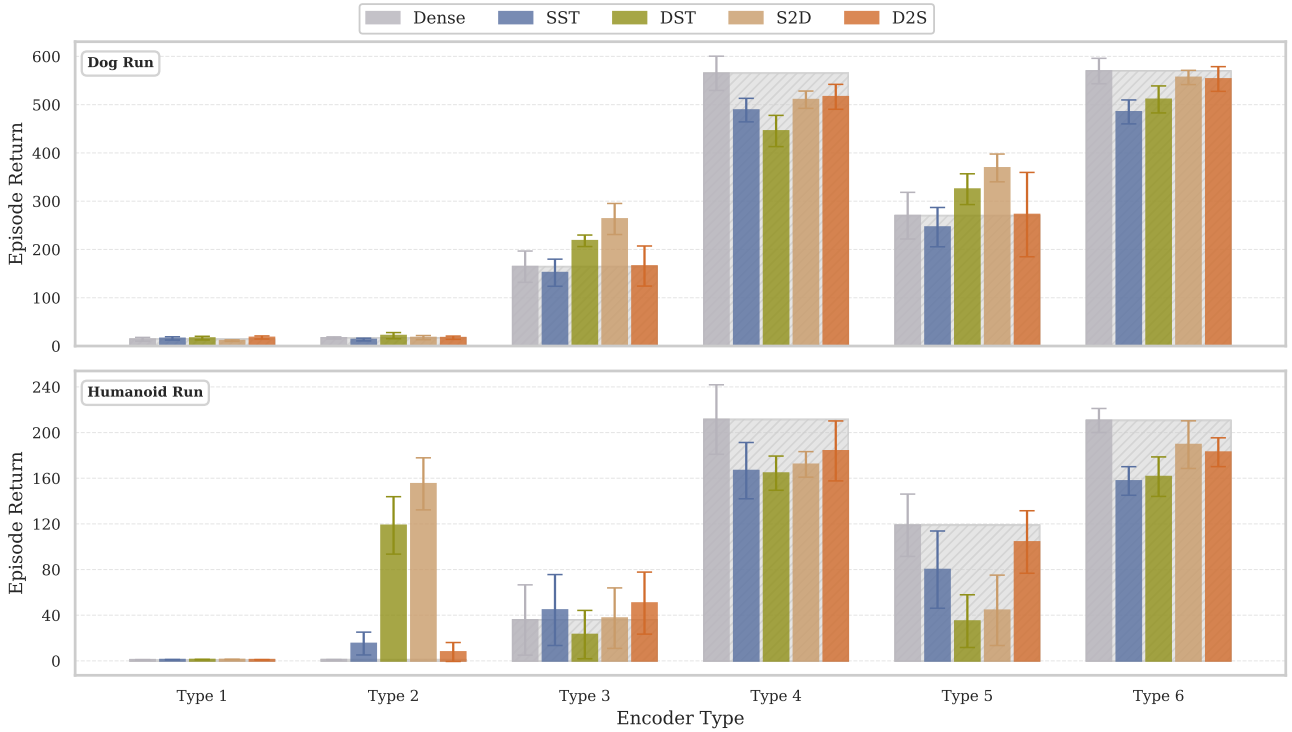



**Figure** 18: Detailed performance comparison of encoder architecture types (1-6) and training regimes (Dense, SST, DST, S2D, D2S) on Dog Run and Humanoid Run at default network size. Note that architectures with layer normalization (Types 3-6) substantially outperform those without (Types 1-2), regardless of training regime employed.

Figure 18 shows the detailed performance comparison of different encoder architecture types and training regimes at the default model size (approximately 2M parameters) on Dog Run and Humanoid Run tasks. These results demonstrate that architectural improvements (progression from Type 1 to Type 6) provide substantially greater performance gains than differences between training regimes. Networks without layer normalization (Types 1-2) show consistently poor performance across all training regimes, while

networks with both layer normalization and residual connections (Types 5-6) achieve strong performance regardless of training approach.

**Scale Trend.** Figure 19 illustrates how scaling affects performance across representative encoder architectures (Types 2, 4, and 6) and training regimes. As parameter counts increase by factors of 3 and 5, the importance of architectural design becomes even more pronounced. Type 2 networks fail catastrophically at larger scales regardless of training regime, Type 4 networks benefit from sparsity-based approaches when scaled, and Type 6 networks maintain strong performance across all scales with dense training. This progression clearly demonstrates that well-designed architectures create inherently better optimization landscapes that enable effective scaling.

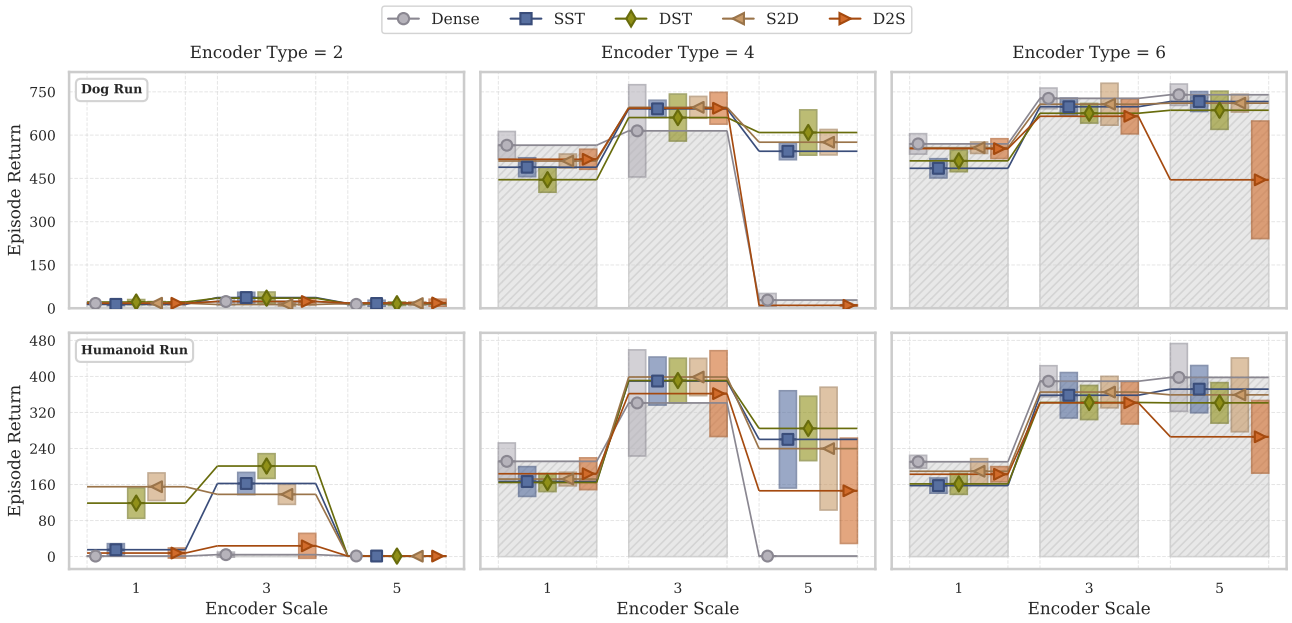



**Figure** 19: Scaling behavior of encoder performance as parameter count increases across three orders of magnitude (Scale 1: 2M, Scale 3: 37M, Scale 5: 154M) for representative architecture types. Results are shown for Dog Run (top) and Humanoid Run (bottom) tasks. While network sparsity and dynamic training regimes can improve scalability for some architectures, the fundamental network design remains the primary determinant of scaling potential.

### D.2. Critic

Following our encoder analysis, we next investigate critic networks that learn through temporal difference (TD) bootstrapping. With a powerful encoder providing stable state representations, we examine how different critic architectures and training regimes impact performance and scalability.

**Default Scale.** Figure 20 presents the performance comparison of different critic architecture types and training regimes at the default model size (approximately 3M parameters). With effective state representations from our encoder, performance differences between architectures and training regimes

become less pronounced at this default scale. The most significant improvement comes from layer normalization (comparing Types 1-2 vs. Types 3-6), which provides consistent benefits across all training approaches. This aligns with recent research highlighting layer normalization's critical role in mitigating plasticity loss and reducing value overestimation in TD-based critic training [43, 49].

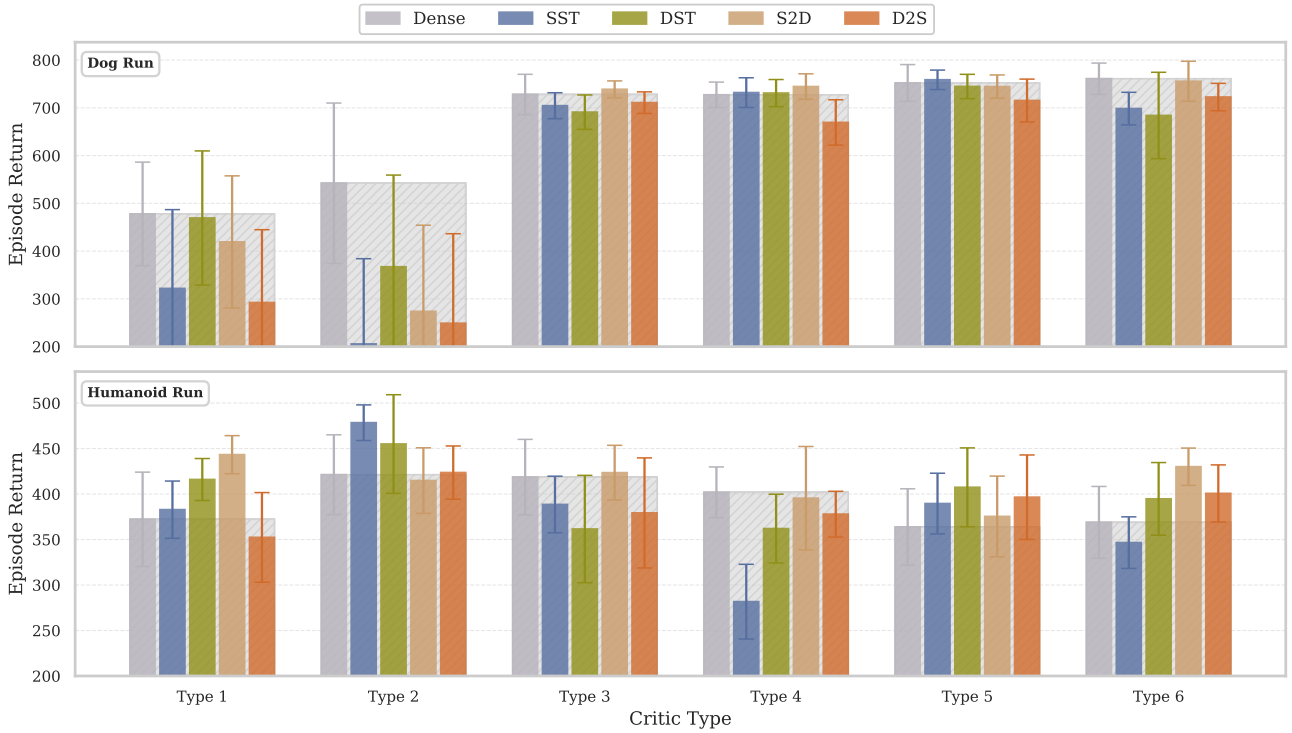



**Figure** 20: Performance comparison of critic architecture types (1-6) and training regimes at default network size on Dog Run (top) and Humanoid Run (bottom) tasks. With a powerful encoder providing effective state representations, critic networks achieve more consistent performance across architectures compared to encoders. Layer normalization emerges as the most important architectural element for critic networks at default scale.

**Scale Trend.** Figure 21 and Figure 22 reveal pronounced differences between architectures and training regimes as critic networks are scaled up. At larger scales (33M and 139M parameters), dense networks suffer severe performance degradation across most architectures, with catastrophic collapse particularly evident in networks without residual connections (Types 1-4). Training regimes that introduce network sparsity and dynamicity substantially enhance critic scalability, with Dynamic Sparse Training (DST) demonstrating the most consistent improvements by maintaining constant sparsity while enabling topology adaptation throughout training.

These results reveal a unique characteristic of TD-based learning: neither architectural innovations nor training regimes alone can overcome the critic scaling barrier. Achieving full scalability requires both robust architectural foundations and specialized training regimes that induce beneficial network properties.

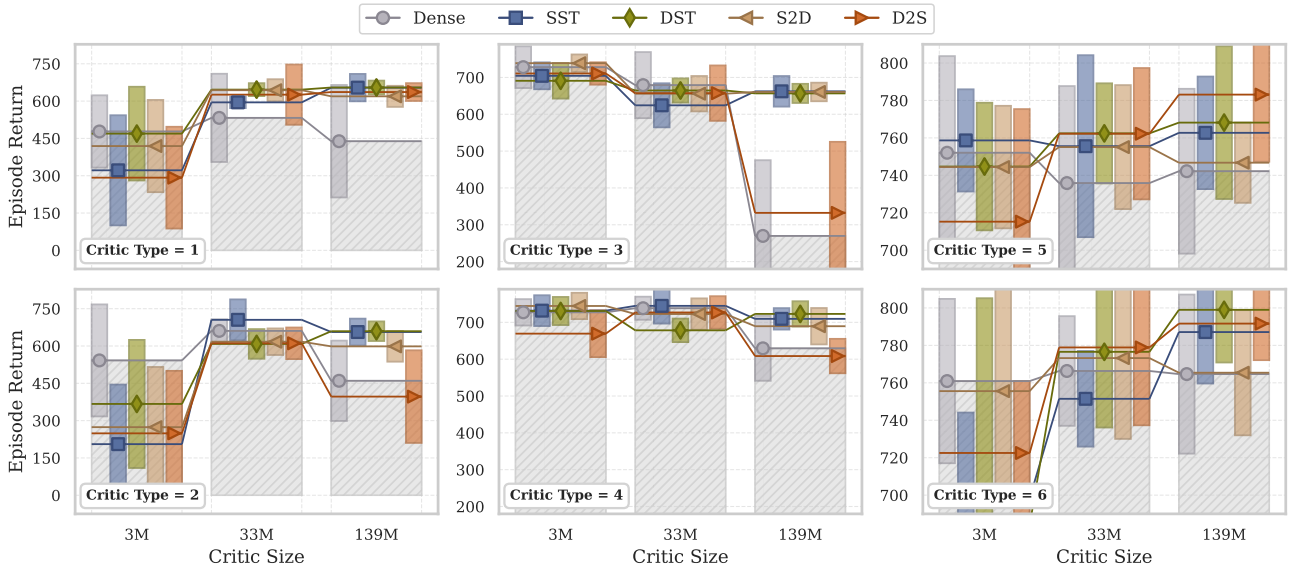



**Figure** 21: Scalability of critic networks on the Dog Run task across different architecture types and training regimes. As parameter count increases from 3M to 33M and 139M, dense networks (gray) suffer severe performance degradation in architectures without residual connections. Dynamic training approaches, particularly DST (green), consistently provide the best scaling performance. Networks with both advanced architecture (Types 5-6) and dynamic training achieve the highest and most stable results at large scales.

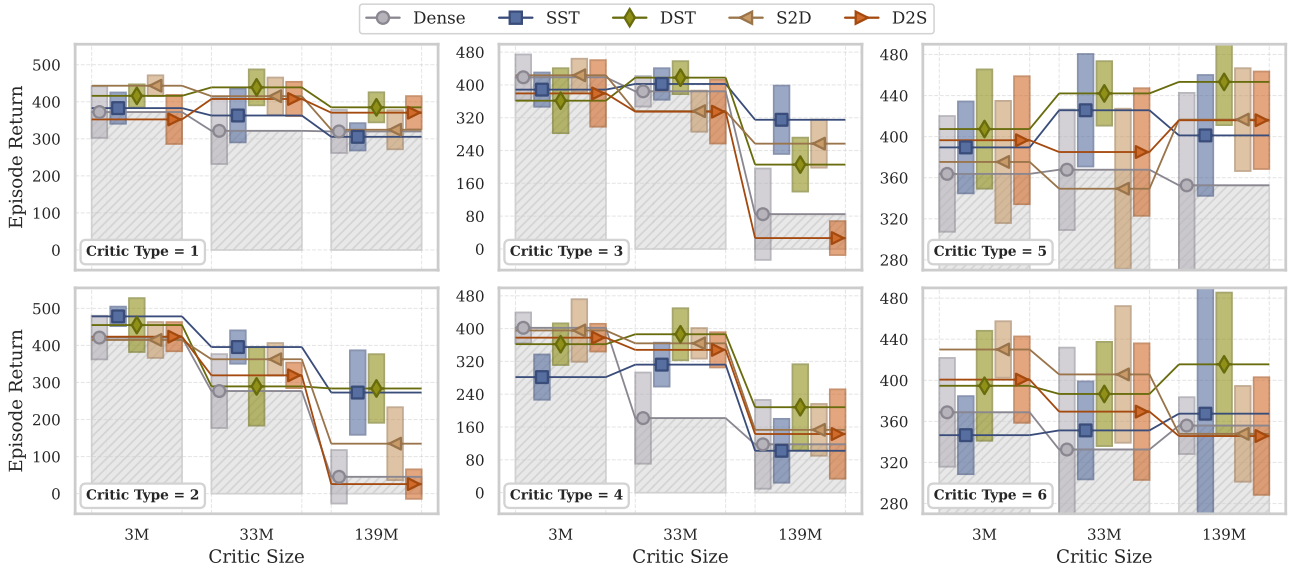



**Figure** 22: Scalability of critic networks on the Humanoid Run task across architecture types and training regimes. The results parallel those seen on Dog Run, with dense networks showing catastrophic collapse at larger scales, particularly for Types 1-4. Dynamic Sparse Training (DST) consistently enables better scaling across architectures, and the combination of advanced architecture features (Types 5-6) with dynamic training regimes produces the most stable scaling behavior.

### D.3. Actor

Finally, we examine actor networks which directly optimize deterministic policy gradients, creating distinct stability requirements and optimization challenges compared to encoders and critics. Here we maintain a scaled Type 6 encoder (37M parameters) and default Type 4 critic (3M parameters) to isolate actor effects.

**Default Scale.** Figure 23 illustrates the performance of different actor architectures and training regimes at default network size (approximately 1.3M parameters). Layer normalization consistently benefits all actor architectures regardless of training regime, similar to our observations with critics. However, unlike encoders where ELU activation improved performance, switching from ReLU to ELU significantly degrades actor performance. This suggests that feature sparsity from ReLU better supports policy gradient optimization despite potential dormant neuron concerns. Dynamic training approaches help simpler architectures without normalization but provide no advantages for more advanced configurations, indicating that actors maintain stable optimization pathways with proper architectural elements.

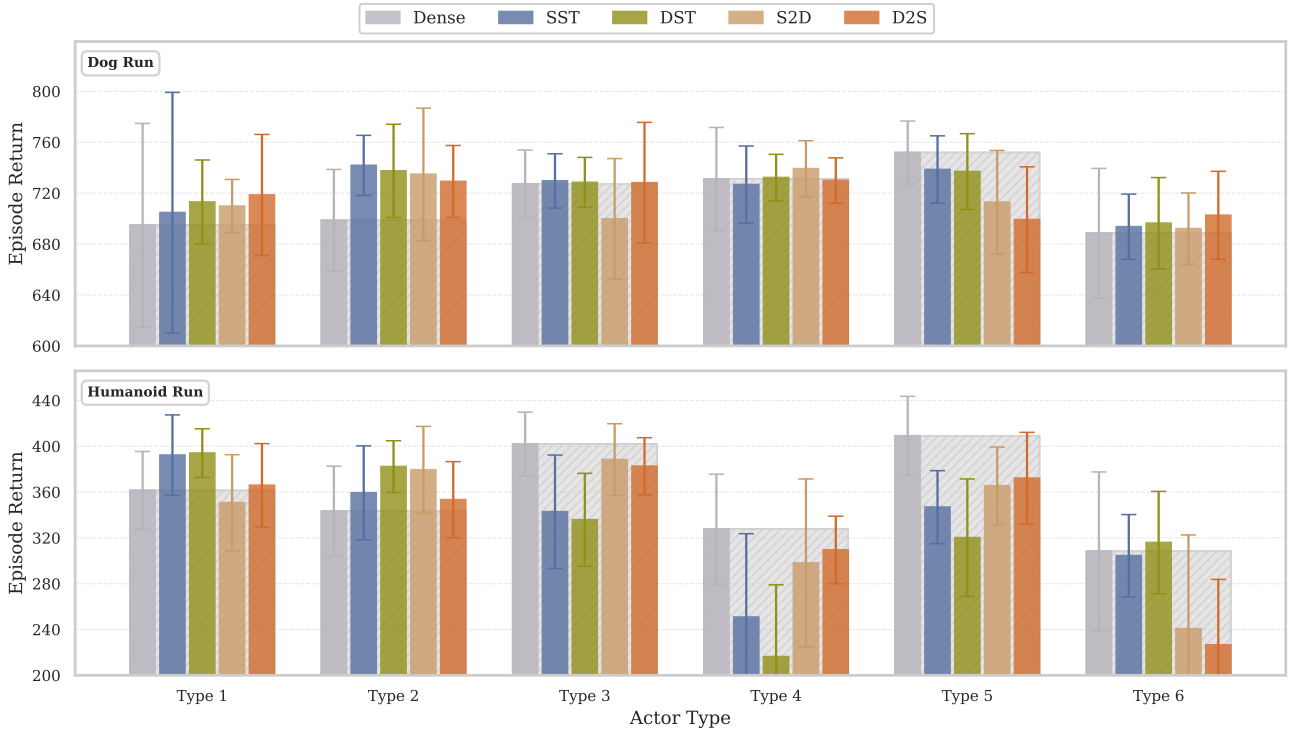



**Figure** 23: Performance comparison of actor architecture types (1-6) and training regimes at default network size on Dog Run (top) and Humanoid Run (bottom) tasks. Layer normalization provides consistent benefits across all architectures, while ELU activation surprisingly degrades performance compared to ReLU. Dynamic training regimes offer little advantage for actors with well-designed architectures, unlike the pattern observed with critics.

**Scale Trend.** Figure 24 and Figure 25 reveal clear differences in scaling patterns across actor training regimes. As network size increases to 16M and 69M parameters, normalization and residual connections become essential to prevent catastrophic performance decline. Most strikingly, and contrary to our findings with critics, Static Sparse Training (SST) consistently outperforms all dynamic approaches for scaled actor networks. This advantage becomes more pronounced at larger scales, where dynamic approaches like DST show much higher performance variance than SST.

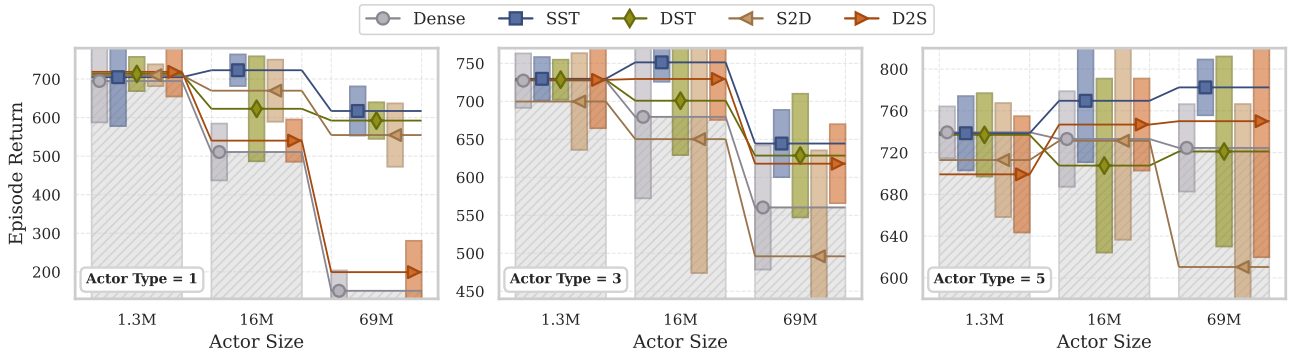



**Figure** 24: Scaling behavior of actor networks on the Dog Run task as parameter count increases from 1.3M to 16M and 69M. For networks without residual connections (Types 1 and 3), performance collapses dramatically at the largest scale regardless of training regime. Static Sparse Training (SST, blue) provides the most consistent performance gains across scales, contrasting sharply with critics where dynamic approaches worked best.

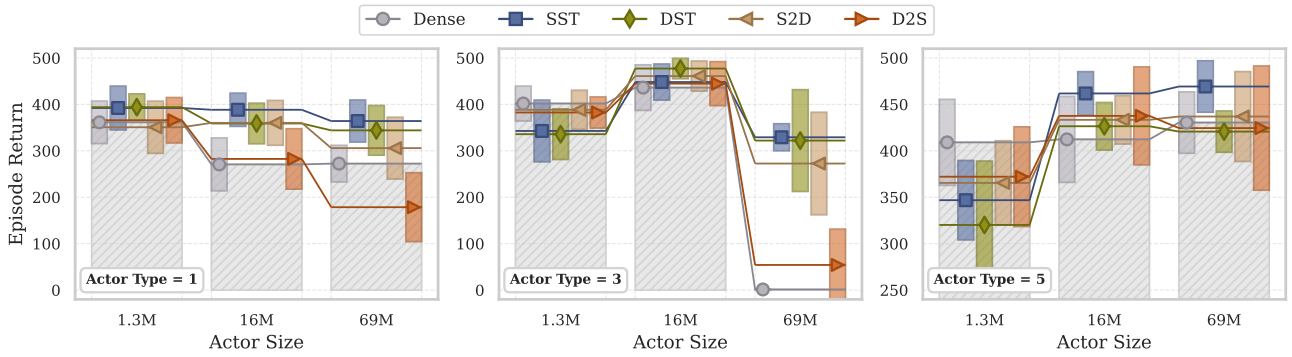



**Figure** 25: Scaling behavior of actor networks on the Humanoid Run task across different architectures and training regimes. The results mirror the Dog Run findings, with SST providing superior scaling performance. Networks with residual connections (Type 5, right column) maintain performance even at the largest scale when trained with static sparsity. The contrast with critic scaling behavior highlights the fundamentally different optimization requirements of actor and critic networks.

These results highlight a fundamental trade-off in actor networks that differs from critics: while mitigating plasticity loss remains important, preserving stable optimization pathways is equally crucial for policy learning. Static Sparse Training provides this ideal balance by reducing parameter redundancy without disrupting established gradient flows, explaining its superior performance in policy network scaling. This pattern directly contrasts with critics, where dynamic training performed best, emphasizing the need for

module-specific training approaches in DRL.

## E. Extended Experiments of Module-Specific Training (MST)

To demonstrate MST's broader applicability beyond MR.Q, we evaluate its effectiveness on MR.SAC, which combines the self-supervised encoder training from MR.Q with SAC's actor-critic learning. We test three scaling approaches: scaling depth only, scaling width only, and scaling both dimensions.

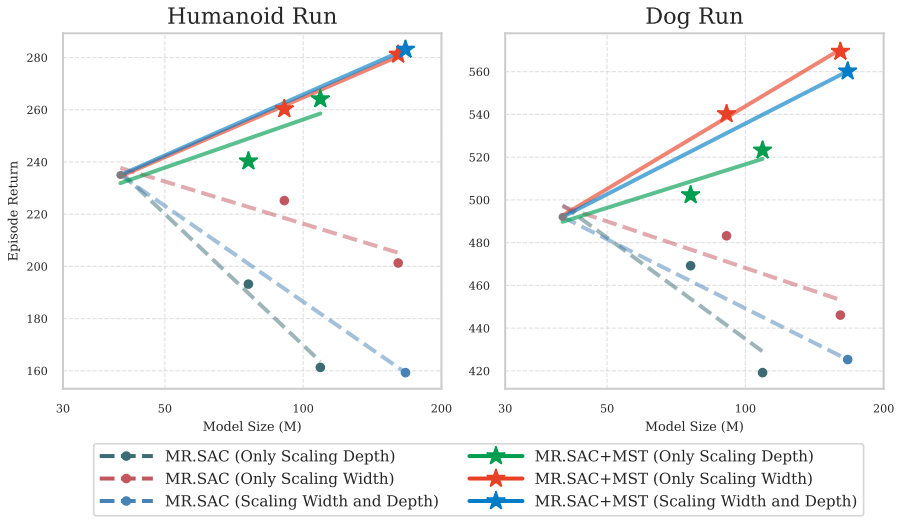



**Figure** 26: MST enables successful scaling in MR.SAC across different scaling dimensions. Standard scaling approaches (dashed lines) lead to performance degradation as model size increases, while MST (solid lines) successfully harnesses the benefits of scaling. Results averaged over 5 seeds on Humanoid Run and Dog Run tasks.

Figure 26 shows that standard scaling consistently degrades performance across all approaches and both tasks. In contrast, MST enables successful scaling across all dimensions. These results confirm that MST's benefits generalize beyond the specific MR.Q framework to other RL algorithms that incorporate model-based representation learning, establishing its practical utility for scalable DRL.