Fractional Rectifier Activations for Off-Policy Reinforcement Learning: A Systematic Empirical Study

Anonymous authors
Paper under double-blind review

Abstract

Activation functions play a key role in deep reinforcement learning by shaping how neural networks approximate policies and value functions. The Rectified Linear Unit (ReLU) remains the dominant choice due to its simplicity and computational efficiency. However, ReLU and its common variants rely on piecewise linear transformations, which may limit representational flexibility when modelling complex nonlinear dynamics in continuous-control tasks. In this work, we investigate fractional-order nonlinearities that extend rectifier-style activations while preserving their computational simplicity. We investigate three fractional variants: Fractional ReLU (FReLU), Fractional Leaky ReLU (FLReLU), and Fractional Parametric ReLU (FPReLU), which incorporate a fractional exponent that enables smooth and continuously adjustable nonlinear transformations.

We conduct a systematic empirical study of these activations in off-policy reinforcement learning by integrating them into two widely used algorithms, TD3 and SAC. Experiments are performed on continuous-control benchmarks from MuJoCo and the DeepMind Control Suite. Across tasks, architectures, and algorithms, fractional activations frequently outperform conventional rectifier functions, with the best fractional configuration yielding an average improvement of about 21% in normalized Area Under the learning Curve (AUC) relative to the ReLU baseline. These results suggest that fractional rectifier activations provide a simple architectural modification that can improve function approximation and learning efficiency in deep reinforcement learning.

1 Introduction

Reinforcement learning (RL) has advanced significantly in recent years, expanding from relatively simple control problems to complex domains involving high-dimensional observations and dynamic environments Terven (2025); Tang et al. (2025). A key driver of this progress is the integration of deep neural networks, which serve as powerful function approximators for modelling policies and value functions Arulkumaran et al. (2017). Because modern RL methods rely heavily on neural networks for function approximation, the representational capacity and optimization dynamics of these networks play a critical role in determining learning stability, convergence speed, and sample efficiency Henderson et al. (2018). Consequently, the performance of deep RL systems depends not only on algorithmic advances but also on architectural design choices within the underlying neural networks, including components such as activation functions Goodfellow et al. (2016).

Activation functions introduce the nonlinearity required for neural networks to approximate complex mappings across diverse tasks Dubey et al. (2022). Among them, ReLU Nair & Hinton (2010) is the most widely adopted activation function in RL due to its simplicity, computational efficiency, and robustness against vanishing gradients Agarap (2018). Despite its popularity, ReLU exhibits several known limitations, including neuron inactivation, and restricted representational flexibility arising from its piecewise-linear structure Xu et al. (2015); Lu et al. (2020). To address these issues while maintaining computational efficiency, several variants have been proposed, such as LReLU Maas et al. (2013) and PReLU He et al. (2015), which modify the behavior of the activation in the negative region.

Beyond these variants, fractional-order formulations offer an alternative approach to increasing the flexibility of neural nonlinearities. Fractional calculus extends differentiation and integration to non-integer orders, enabling richer functional representations Ortigueira & Coito (2004). Fractional operators have been widely applied in fields such as control theory and signal processing, where they allow more flexible modelling of complex systems Jian et al. (2025). More recently, fractional-order activation functions have been explored in deep learning, where they have shown improvements in representation quality and classification performance Kumar et al. (2024); Job et al. (2022); Molek & Alijani (2025). These findings suggest that fractional activations may offer a promising alternative to conventional piecewise-linear functions.

This paper investigates fractional-order extensions of ReLU-based activation functions for reinforcement learning. In particular, we study three fractional variants: Fractional ReLU (FReLU), Fractional Leaky ReLU (FLReLU), and Fractional Parametric ReLU (FPReLU). These activations extend conventional rectifier functions through non-integer parameterization, enabling more flexible nonlinear transformations while preserving the simplicity of rectifier-style activations. Importantly, our approach does not modify the underlying RL algorithms. Instead, we replace only the activation functions within the agents' neural networks, allowing us to isolate the effect of activation design on learning performance.

To evaluate this idea empirically, we replace the standard ReLU-based activations in two widely used actor–critic algorithms, Twin Delayed Deep Deterministic Policy Gradient (TD3) Fujimoto et al. (2018) and Soft Actor-Critic (SAC) Haarnoja et al. (2018), with their fractional counterparts as a drop-in architectural modification. While standard implementations of these algorithms typically rely on ReLU as the default activation function, our study examines whether more expressive nonlinearities can improve learning behaviour. Experiments are conducted across continuous-control benchmarks from both MuJoCo Todorov et al. (2012) and the DeepMind Control Suite Tassa et al. (2018). To ensure a controlled evaluation, we consider both single-hidden-layer and two-hidden-layer actor–critic architectures with fixed width. In the deeper setting, we further examine how the placement of activation functions within the actor and critic networks influences performance.

Our experimental results show that fractional activations generally outperform conventional rectifier activations across a wide range of tasks, algorithms, and architectural settings. In particular, fractional variants frequently achieve higher normalized AUC than ReLU, LReLU, and PReLU, indicating improved learning efficiency and policy performance. The strongest gains appear in several locomotion environments, where fractional activations often accelerate early learning and achieve higher final returns. Among the proposed methods, FPReLU provides the most consistent improvements across algorithms and architectures. These findings highlight fractional activations as a simple yet effective architectural modification for improving neural function approximation and reinforcement learning performance.

The implementation code will be made publicly available on GitHub after the anonymous review process.

The main contributions of this paper are summarized as follows:

- A systematic investigation of how activation function design influences learning behaviour in RL by replacing the default ReLU nonlinearity in actor–critic agents while keeping the underlying algorithms unchanged.

- A controlled empirical evaluation within the TD3 and SAC frameworks across continuous-control benchmarks from MuJoCo and the DeepMind Control Suite, considering both single-layer and two-layer neural architectures.

- Empirical evidence that fractional activations frequently outperform conventional rectifier activations across tasks and architectures, highlighting their potential to improve neural function approximation and RL performance.

2 Related Work

In this section, we first discuss the role of activation functions in deep learning. We then review architectural approaches used to improve neural network design in RL. Finally, we describe prior work on fractional activation functions and highlight the gap that motivates this study.

2.1 Activation Functions in Deep Learning

Activation functions play a critical role in deep learning by introducing the nonlinear transformations required for neural networks to approximate complex functions Dubey et al. (2022). Early activation functions such as the sigmoid and hyperbolic tangent were widely used, but their saturating behaviour often resulted in vanishing gradients and slow optimisation in deep networks Glorot & Bengio (2010). The introduction of the ReLU significantly improved training stability by providing non-saturating gradients in the positive region, and it quickly became the dominant activation function in modern deep learning Nair & Hinton (2010); Agarap (2018).

Despite its success, ReLU exhibits several limitations, including neuron inactivation, commonly referred to as the dying ReLU problem, where neurons permanently output zero for negative inputs Lu et al. (2020). To mitigate this issue, several rectifier variants have been proposed. For example, Leaky ReLU introduces a small fixed slope in the negative region Maas et al. (2013), while Parametric ReLU allows the negative slope to be learned during training He et al. (2015). Although these variants improve gradient propagation and reduce neuron inactivation, they retain the piecewise-linear structure of the original rectifier function.

Recent work has also explored automatically discovered activation functions and smoother nonlinear transformations. For example, the Swish activation function was obtained through neural architecture search and demonstrated improved performance across several deep learning tasks Ramachandran et al. (2017). Similarly, the Gaussian Error Linear Unit (GELU) provides smoother nonlinear behaviour and has been widely adopted in modern neural architectures Hendrycks & Gimpel (2017). These studies highlight that activation function design can significantly influence neural network optimisation and representational capacity.

2.2 Neural Network Design in Reinforcement Learning

Deep RL relies on neural networks as function approximators for both policies and value functions. Consequently, architectural choices that influence optimisation dynamics and function approximation can directly affect agent performance. However, many RL algorithms adopt neural network components originally developed for supervised learning, with relatively limited investigation of their impact in RL settings. In practice, ReLU remains the default activation function in widely used algorithms such as Deep Q-Networks (DQN) Mnih et al. (2015), Deep Deterministic Policy Gradient (DDPG) Lillicrap et al. (2015), TD3 Fujimoto et al. (2018), and SAC Haarnoja et al. (2018).

Studies typically explore improving RL performance by modifying algorithmic components within deep RL agents. One prominent direction focuses on improving value estimation through ensembles of critic networks. For example, Randomized Ensembled Double Q-learning (REDQ) improves sample efficiency by maintaining a large ensemble of critic networks with randomized updates Chen et al. (2021). Similarly, Truncated Quantile Critics (TQC) reduce overestimation in distributional RL by truncating the upper tail of the learned value distribution Kuznetsov et al. (2020). These approaches demonstrate that modifying architectural components of the value function estimator can improve the stability and performance of deep RL algorithms.

Other work improves RL performance through architectural regularisation mechanisms. DroQ introduces dropout within critic networks to approximate ensemble-style regularisation with lower computational cost Hiraoka et al. (2022). Similarly, CrossQ incorporates batch normalisation within Q-networks to stabilise training and improve sample efficiency Bhatt et al. (2024). These studies further highlight that neural network architectural design choices can influence the optimisation behaviour and stability of deep RL systems

Despite these advances, comparatively little attention has been given to the role of activation functions themselves. As fundamental elements of neural network architectures, activation functions define the nonlinear transformations used in function approximation and therefore influence gradient propagation, optimisation dynamics, and the overall stability of training Goodfellow et al. (2016); Glorot & Bengio (2010). These observations suggest that the design of activation functions remains an underexplored yet potentially impactful direction for improving neural function approximation in reinforcement learning.

2.3 Fractional Activation Functions

One promising direction for designing more expressive nonlinearities arises from fractional calculus. Fractional calculus extends classical differentiation and integration to non-integer orders, enabling more flexible mathematical representations of dynamic systems Ortigueira & Coito (2004). Fractional-order models have been widely applied in areas such as control systems and signal processing, where they can capture complex system behaviours more effectively than integer-order formulations Jian et al. (2025).

More recently, fractional-order concepts have been incorporated into neural network activation functions. Prior studies have shown that introducing fractional-order parameterisations can produce more flexible nonlinear transformations and can improve performance in supervised learning settings across different architectures and activation families Job et al. (2022); Kumar et al. (2024); Molek & Alijani (2025). In particular, recent work has investigated fractional-order derivatives as tunable components of activation design, highlighting their ability to diversify activation behaviour while also analysing their effects on accuracy, computational cost, and memory usage Molek & Alijani (2025). These results suggest that fractional activations constitute a broader and increasingly active research direction rather than an isolated modification of a single rectifier variant.

However, existing studies on fractional activation functions have primarily focused on supervised learning tasks such as classification and regression, and their impact on RL remains largely unexplored Job et al. (2022); Kumar et al. (2024); Molek & Alijani (2025). In contrast to supervised learning, RL agents learn from interaction with an environment using bootstrapped value estimates, delayed reward signals, and continuously evolving data distributions, which introduce optimisation challenges not typically encountered in conventional deep learning settings Sutton & Barto (2018); Mnih et al. (2015). As a result, most advances in deep RL have focused on improving optimisation procedures, critic architectures, and regularisation mechanisms rather than modifying the activation functions themselves Haarnoja et al. (2018); Fujimoto et al. (2018); Chen et al. (2021); Kuznetsov et al. (2020); Hiraoka et al. (2022). Despite the central role of activation functions in shaping neural representations and gradient propagation, comparatively little work has examined their influence on RL training dynamics. In this work, we address this gap by systematically studying fractional variants of rectifier-based activation functions, including FReLU, FLReLU, and FPReLU, and comparing them with conventional activations such as ReLU, LReLU, and PReLU within modern actor–critic algorithms for continuous-control tasks.

3 Methodology

This section describes the methodology used to study fractional activation functions in RL. We first review conventional rectifier activations and then introduce the proposed fractional rectifier variants derived from fractional calculus. The experimental setup, including the treatment of the fractional order parameter, network architectures, activation placement strategies, and integration into TD3 and SAC, is then presented.

3.1 ReLU and Its Variants

Rectifier-based nonlinearities are widely used in modern neural networks due to their simplicity, computational efficiency, and favorable gradient propagation properties Glorot et al. (2011); Goodfellow et al. (2016). Among these, ReLU has become the most widely adopted nonlinearity in deep learning architectures Nair & Hinton (2010). The ReLU activation is defined as

$$\mathrm{ReLU}(x) = \max(0, x), \tag{1}$$

which produces sparse activations and maintains non-saturating gradients for positive inputs.

Two extensions of ReLU modify the behavior of the activation in the negative region while preserving the rectifier structure. Leaky ReLU introduces a small fixed slope for negative inputs,

$$\mathrm{LReLU}(x) = \begin{cases} x, & x \geq 0, \\ kx, & x < 0, \end{cases} \tag{2}$$

where k is a small constant controlling the slope for negative inputs. By allowing a non-zero gradient when $x < 0$, LReLU reduces the risk of neuron inactivation and improves gradient propagation during training.

Another extension is the PReLU, which introduces a learnable parameter controlling the negative slope,

$$\mathrm{PReLU}(x) = \begin{cases} x, & x \geq 0, \\ ax, & x < 0, \end{cases} \tag{3}$$

where the parameter a is optimized during training. This learnable slope enables the network to adapt the activation behavior to different tasks and data distributions. ReLU serves as the primary baseline activation in this study, while LReLU and PReLU are included as commonly used rectifier variants that modify the treatment of negative inputs. These activations provide representative baselines for evaluating alternative nonlinearities within the rectifier family.

Despite their differences, rectifier-based activations share a common piecewise-linear structure, which limits the flexibility of the nonlinear transformations they can represent. In RL settings, where neural networks approximate complex value functions and policies under noisy and non-stationary training signals, such limited flexibility may constrain representation and optimization behavior.

Rectifiers remain the dominant nonlinearities in modern RL architectures, including algorithms such as DQN Mnih et al. (2015), DDPG Lillicrap et al. (2015), TD3 Fujimoto et al. (2018), and SAC Haarnoja et al. (2018). By extending this family using fractional transformations, we seek to introduce adjustable curvature while preserving the underlying rectifier structure.

More complex activations, such as Swish Ramachandran et al. (2017) or GELU Hendrycks & Gimpel (2017), introduce additional nonlinear characteristics beyond rectification. While such functions may also be beneficial in deep learning, comparing fundamentally different activation families would introduce additional confounding factors. Focusing on fractional rectifier variants, therefore, allows us to isolate the effect of curvature within the same activation framework.

3.2 Fractional Rectifier Activation Functions

Fractional activation functions are based on fractional calculus, which extends the concept of differentiation to non-integer orders Oldham & Spanier (1974). Unlike classical derivatives that are restricted to integer orders, fractional derivatives form a continuous family of operators that interpolate between different derivative orders Podlubny (1998). This additional degree of freedom allows nonlinear transformations to be smoothly adjusted via a fractional-order parameter. Fractional calculus has been widely used to model complex nonlinear systems in fields such as control theory, signal processing, and dynamical systems Kilbas et al. (2006). In this work, the same principle is used to construct fractional rectifier activation functions, where the fractional order controls the curvature of the activation and thereby influences the neural network's representational capacity.

Several equivalent formulations of fractional derivatives exist, including the Riemann–Liouville, Caputo, and Grünwald–Letnikov definitions Podlubny (1998); Samko (1993). Although these formulations differ in their

mathematical construction, they yield consistent expressions for fractional derivatives of power functions, which are particularly useful when designing nonlinear transformations Miller & Ross (1993).

For a power function $f(x) = x^k$, the fractional derivative of order α can be written as

$$\frac{d^\alpha}{dx^\alpha} x^k = \frac{\Gamma(k+1)}{\Gamma(k+1-\alpha)} x^{k-\alpha} \tag{4}$$

where $\Gamma(\cdot)$ denotes the Gamma function that generalises the factorial to non-integer values. The exponent $k - \alpha$ shows that fractional differentiation modifies the curvature of the function continuously through the parameter α, providing a mathematical basis for constructing activation functions with adjustable nonlinear behaviour.

3.2.1 Definition of Fractional Activations

Using the fractional power formulation in Eq. equation 4, we define three fractional extensions of rectifier activation functions. Each activation incorporates a fractional power transformation that allows the curvature of the nonlinearity to be continuously controlled through the fractional order parameter α. To maintain consistent scaling across different fractional orders, each activation is normalised by the factor $1/\Gamma(2-\alpha)$, ensuring that the proposed activations reduce exactly to their classical rectifier forms when $\alpha = 0$.

Fractional ReLU (FReLU). Fractional ReLU modifies the positive region of the standard ReLU by introducing a fractional power transformation:

$$\mathrm{FReLU}_\alpha(x) = \begin{cases} \dfrac{1}{\Gamma(2-\alpha)} x^{1-\alpha}, & x > 0, \\ 0, & x \leq 0. \end{cases} \tag{5}$$

The exponent $(1 - \alpha)$ controls the curvature of the activation function. When α is close to zero, the function behaves similarly to the standard ReLU and grows approximately linearly for positive inputs. As α increases, the exponent decreases and the positive branch becomes progressively more curved, compressing large activations. FReLU therefore preserves the zero-valued negative branch of ReLU while introducing curvature control through α without adding learnable parameters.

Fractional Leaky ReLU (FLReLU). Fractional Leaky ReLU extends the LReLU formulation by applying fractional curvature in both positive and negative regions:

$$\mathrm{FLReLU}_{\alpha,k}(x) = \begin{cases} \dfrac{1}{\Gamma(2-\alpha)} x^{1-\alpha}, & x > 0, \\ -\dfrac{k}{\Gamma(2-\alpha)} |x|^{1-\alpha}, & x \leq 0, \end{cases} \tag{6}$$

where $k > 0$ controls the slope of the negative branch. Unlike ReLU, which suppresses negative inputs entirely, FLReLU preserves gradient flow in the negative region while allowing curvature modulation through the fractional exponent. The leakage parameter k determines the magnitude of the negative slope, while α governs the curvature of both branches.

Fractional Parametric ReLU (FPReLU). Fractional Parametric ReLU further generalises this formulation by introducing a learnable parameter for the negative slope:

$$\mathrm{FPReLU}_{\alpha,a}(x) = \begin{cases} \dfrac{1}{\Gamma(2-\alpha)} x^{1-\alpha}, & x > 0, \\ -\dfrac{a}{\Gamma(2-\alpha)} |x|^{1-\alpha}, & x \leq 0, \end{cases} \tag{7}$$

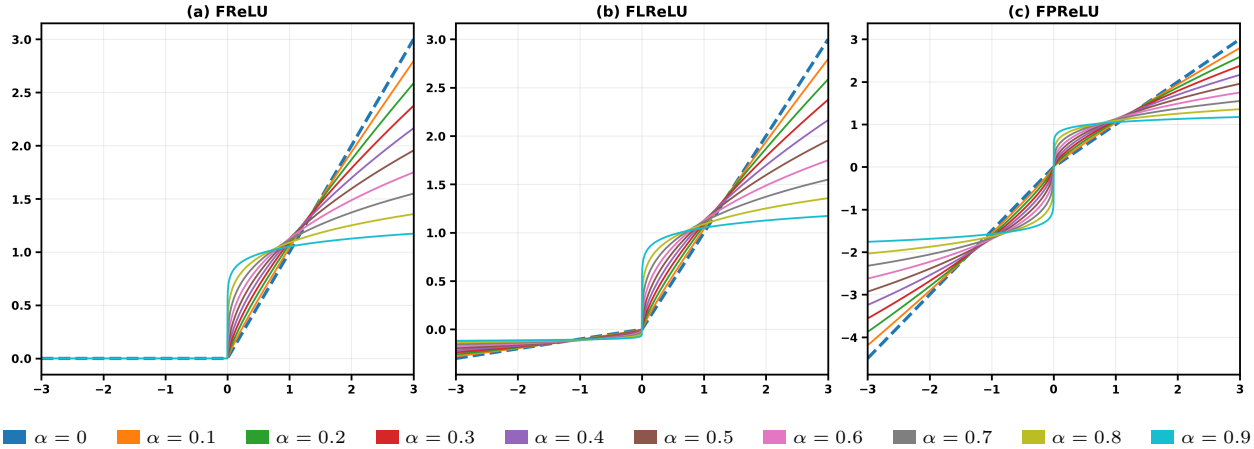


Figure 1: Fractional rectifier activation functions for different values of the fractional order α: (a) FReLU, (b) FLReLU, and (c) FPReLU. The case $\alpha = 0$ corresponds to the classical rectifier baseline.

Table 1: Structural comparison between conventional rectifier activations and the proposed fractional rectifier variants. The fractional order α introduces curvature control while preserving compatibility with classical rectifier functions.

Activation	**Negative branch**	**Curvature control (α)**	**Learnable parameter**	**Smooth**[†]	**Limit behaviour**
ReLU	0	✗	✗	✗	Baseline
LReLU_k	$-k\,\lvert x\rvert$	✗	Fixed k	✗	Baseline
PReLU_a	$-a\,\lvert x\rvert$	✗	Learnable a	✗	Baseline
FReLU_α	0	✓	✗	✓	$\alpha \to 0 \Rightarrow$ ReLU
$\text{FLReLU}_{\alpha,k}$	$-k\,\lvert x\rvert^{1-\alpha}$	✓	Fixed k	✓	$\alpha \to 0 \Rightarrow \text{LReLU}_k$
$\text{FPReLU}_{\alpha,a}$	$-a\,\lvert x\rvert^{1-\alpha}$	✓	Learnable a	✓	$\alpha \to 0 \Rightarrow \text{PReLU}_a$

[†] Smooth everywhere except at $x = 0$; smoothness increases as $\alpha \to 1$.

where the parameter a is optimised during training. In this formulation, α controls the curvature of the activation, while a enables adaptive adjustment of the negative slope according to the learning dynamics of the task. By combining fractional curvature control with a learnable leakage parameter, FPReLU provides the most flexible activation behaviour among the proposed functions. When $\alpha \to 0$, the formulation reduces to the classical PReLU function while retaining continuous curvature modulation for $\alpha > 0$.

For consistency with commonly used rectifier variants, the parameters of the baseline functions are fixed to standard values. In particular, the negative slope of LReLU is set to $k = 0.1$, while the parameter of PReLU is set to $a = 1.5$, following the configuration reported in the original fractional rectifier activation study Job et al. (2022).

Together, these formulations introduce a unified family of fractional rectifier activations that extend classical rectifier functions with continuous curvature control while preserving their computational simplicity.

Figure 1 illustrates the behaviour of the proposed activations for different values of the fractional order α. The colored curves correspond to fractional orders ranging from $\alpha = 0$ to $\alpha = 0.9$. When $\alpha = 0$, the proposed functions reduce to their classical rectifier counterparts: ReLU in Figure 1(a), LReLU in Figure 1(b), and PReLU in Figure 1(c). As α increases, the exponent $(1 - \alpha)$ decreases, producing progressively stronger curvature and compressing large activations relative to the linear rectifier baseline.

Table 1 summarises the structural properties of the proposed fractional activations and compares them with their classical rectifier counterparts. The table highlights how the fractional order α introduces explicit

curvature control while preserving the basic structural form of conventional rectifier functions. In particular, FReLU, FLReLU, and FPReLU reduce respectively to ReLU, LReLU, and PReLU as $\alpha \to 0$, while enabling continuous curvature modulation for $\alpha > 0$.

3.2.2 Fractional Order Parameter

Fractional activation functions introduce an additional parameter α that determines the order of the fractional transformation and therefore controls the curvature of the resulting nonlinearity. In contrast to conventional rectifier activations, which exhibit fixed piecewise-linear behaviour, the fractional exponent $(1 - \alpha)$ enables continuous modulation of activation growth and gradient behaviour.

As α increases, the positive branch of the activation progressively deviates from the linear behaviour of standard rectifier functions and becomes more curved. This transformation compresses large activation values while maintaining non-zero gradients, which can influence optimisation dynamics during training.

To analyse the influence of the fractional order, we evaluate the proposed activations over the range

$$\alpha \in \{0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5\} \tag{8}$$

This range allows us to explore moderate curvature variations while remaining close to the classical rectifier behaviour obtained when $\alpha = 0$. Preliminary experiments indicated that larger values of α tend to reduce training stability and degrade performance across several tasks. Consequently, the analysis focuses on the interval $\alpha \leq 0.5$, where fractional activations provide stable training behaviour while still introducing meaningful variations in activation curvature.

For each task and experimental configuration, results are analysed both as a function of α and after selecting the best-performing value for that task. This dual analysis enables us to evaluate the sensitivity of RL performance to the fractional order while avoiding the possibility that improvements are obscured when averaging across suboptimal parameter values. Future work will seek to enable representations where α is a learned parameter.

3.3 Network Architectures

To isolate the influence of activation functions from changes in network capacity, the proposed activations are evaluated using two actor–critic architectures commonly employed in continuous-control RL. Both architectures follow the standard multilayer perceptron design used in modern actor–critic algorithms, allowing the effect of activation functions to be studied without introducing additional architectural variations.

Single-hidden-layer architecture. In the first configuration, both the actor and critic networks contain a single hidden layer. This architecture provides a controlled experimental setting in which differences in performance can be attributed primarily to the choice of activation function rather than interactions with deeper hierarchical representations. By limiting network depth, the influence of the nonlinearity itself becomes easier to interpret.

Two-hidden-layer architecture. In the second configuration, both the actor and critic networks employ two hidden layers. Two-hidden-layer multilayer perceptrons are widely used in modern actor–critic algorithms such as TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018). Increasing network depth expands the representational capacity of neural networks and enables hierarchical feature composition, which can influence how nonlinear transformations affect learning dynamics. Bengio (2009); Goodfellow et al. (2016).

3.4 Activation Placement

When multiple hidden layers are present, the placement of nonlinear activation functions can influence gradient propagation and feature transformations in deep networks Glorot et al. (2011); Goodfellow et al. (2016). Different layers often capture representations at different levels of abstraction, and modifying nonlinearities at specific depths can therefore affect how information is processed during training.

This consideration is particularly relevant in actor–critic RL, where the actor learns a policy that maps states to actions while the critic estimates value functions used to guide policy updates Konda & Tsitsiklis (1999); Heess et al. (2015). Changing nonlinear transformations within these components may therefore influence policy representation, value estimation accuracy, and overall optimization stability.

To investigate these effects, we evaluate several activation placement strategies within the two-layer architecture:

- **`all-actor (AA)`**: fractional activations are applied to all hidden layers of the actor network,
- **`all-critic (AC)`**: fractional activations are applied to all hidden layers of the critic network,
- **`all-both (AB)`**: fractional activations are applied to all hidden layers of both the actor and critic networks,
- **`first-both (FB)`**: the fractional activation is applied to the first hidden layer of both networks,
- **`first-actor (FA)`**: the fractional activation is applied only to the first hidden layer of the actor network.

These configurations allow us to analyse how activation functions interact with the distinct roles of the actor and critic. Applying fractional activations only to the actor isolates their effect on policy representation, whereas modifying the critic focuses on value function approximation and the gradients used for policy updates.

The `all-both` configuration evaluates the effect of applying the nonlinear transformation uniformly across the entire actor–critic system. In contrast, the `first-actor` and `first-both` configurations target the first hidden layer, which performs the initial transformation of input state features. Modifying activations at this stage may influence how low-level representations are formed and propagated through subsequent layers.

Across all configurations, the standard ReLU activation used in the original TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018) implementations serves as the baseline.

3.5 Integration into RL Algorithms

To evaluate the effect of the proposed nonlinearities, we integrate the fractional activation functions into two widely used continuous-control algorithms: Soft Actor-Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradient (TD3). In both algorithms, the actor and critic networks are implemented as multilayer perceptrons with hidden layers of width 256.

In the original implementations of SAC and TD3, ReLU is used as the activation function in all hidden layers. In this work, these activations are replaced with one of the proposed fractional variants (FReLU, FLReLU, or FPReLU). No other architectural or algorithmic modifications are introduced, ensuring that any observed performance differences arise solely from the choice of activation function.

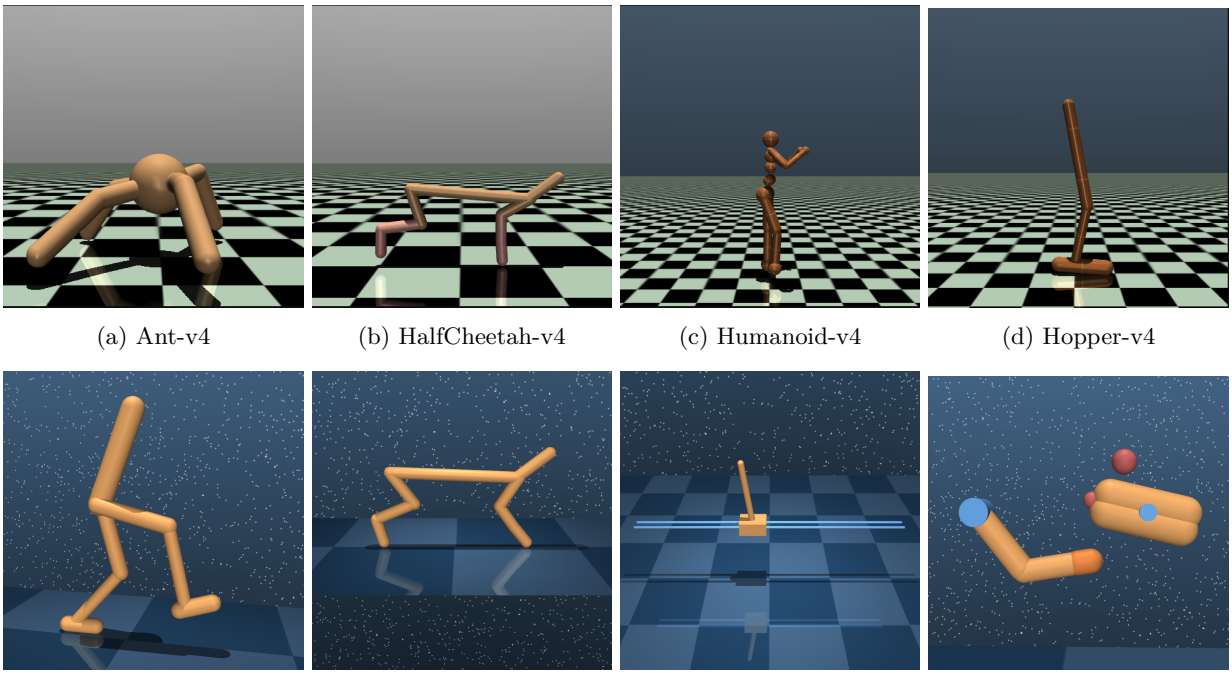
For illustration, consider the single-layer architecture used in our experiments. In this setting, the SAC actor network takes the form

$$\texttt{Linear}(d_{in} \to 256) \to \mathrm{FAct}_\alpha,$$

followed by the output layers that parameterise the stochastic policy. Similarly, the critic network is defined as

$$\texttt{Linear}(d_{in} \to 256) \to \mathrm{FAct}_\alpha \to \texttt{Linear}(256 \to 1).$$

For the standard two-layer architecture, an additional hidden layer with the same activation is inserted:



(a) Ant-v4 (b) HalfCheetah-v4 (c) Humanoid-v4 (d) Hopper-v4

(e) Walker-Walk (f) Cheetah-Run (g) Cartpole-Swingup (h) Finger-Spin

Figure 2: Continuous-control benchmark tasks used in the experiments. The first row shows MuJoCo environments accessed through OpenAI Gym, while the second row shows tasks from the DeepMind Control Suite.

$$\texttt{Linear}(d_{in} \to 256) \to \mathrm{FAct}_\alpha \to \texttt{Linear}(256 \to 256) \to \mathrm{FAct}_\alpha.$$

Here FAct_α denotes one of the proposed fractional activations (FReLU, $\mathrm{FLReLU}_{\alpha,k}$, or $\mathrm{FPReLU}_{\alpha,a}$). All other components of the SAC and TD3 algorithms, including optimisation procedures, target network updates, and policy learning mechanisms, remain unchanged. This design provides a controlled experimental setting in which the impact of activation function design can be analysed independently from other architectural or algorithmic factors.

4 Experiments

This section evaluates the effect of activation function design in off-policy continuous-control RL. We compare conventional rectifier activations with their fractional counterparts within matched TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018) architectures. Specifically, the study includes conventional rectifier variants, namely LReLU and PReLU, together with the proposed fractional variants FReLU, FLReLU, and FPReLU, while the standard ReLU configuration serves as the baseline. This design enables a controlled comparison of how different rectifier families influence learning speed, stability, and final performance while keeping the underlying RL algorithms and network architectures unchanged.

4.1 Benchmarks

We evaluate the proposed activations on continuous-control tasks drawn from two widely used RL benchmark suites: MuJoCo environments accessed through OpenAI Gym Brockman et al. (2016) and the DeepMind Control Suite (DMCS) Tassa et al. (2018). These benchmarks are commonly used in RL research because they include high-dimensional state spaces, complex dynamics, and diverse control objectives, providing a broad testbed for analysing the effect of activation design.

The evaluation includes the following tasks:

MuJoCo (OpenAI Gym) Brockman et al. (2016) *Ant-v4*, *HalfCheetah-v4*, *Hopper-v4*, *Humanoid-v4*. These environments primarily focus on locomotion control with articulated agents. *HalfCheetah-v4* and *Hopper-v4* emphasise forward locomotion efficiency, while *Ant-v4* and *Humanoid-v4* involve higher-dimensional control and coordinated whole-body motion.

DeepMind Control Suite (DMCS) Tassa et al. (2018) *Walker-Walk*, *Cheetah-Run*, *Finger-Spin*, *Cartpole-Swingup*. These tasks include both locomotion and manipulation challenges. *Walker-Walk* and *Cheetah-Run* evaluate locomotion and stability, whereas *Finger-Spin* and *Cartpole-Swingup* focus on precise control and dynamic balancing.

Representative environments from these benchmarks are illustrated in Figure 2.

Together, these tasks cover a wide range of control challenges, including high-dimensional locomotion, under-actuated balance, coordinated whole-body movement, and dynamic control requiring precise timing. They also vary substantially in state dimensionality, transition dynamics, and reward structure. This diversity allows us to assess whether the influence of activation functions is consistent across qualitatively different RL settings rather than being specific to a single task or benchmark.

4.2 Experimental Setup

All experiments follow the standard training configuration commonly used for TD3 and SAC in continuous-control benchmarks. The actor and critic networks are implemented as multilayer perceptrons with hidden layers of width 256, following the architectures described in Section 3.3.

Agents are trained for one million environment steps. During training, policy evaluations are performed periodically using deterministic policies without exploration noise. Each evaluation consists of 10 episodes, and the average return is recorded to construct the learning curves.

Apart from the choice of activation function, all other components of the learning algorithms remain unchanged. In particular, the replay buffer mechanism, target network updates, optimisation procedures, and policy learning strategies follow the original implementations of TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018).

Unless otherwise specified, all hyperparameters remain identical to the standard configurations used in prior continuous-control benchmarks. This ensures that any observed performance differences arise from the activation functions rather than changes in the underlying RL algorithms or training setup.

4.3 Evaluation Protocol

During training, agents are evaluated periodically, and the returns obtained at each evaluation point are averaged following established RL evaluation practice Henderson et al. (2018). The resulting learning curves, therefore, represent the mean episode return as training progresses.

To quantify overall performance, we compute the AUC over a training horizon of one million environment steps. For each run, the evaluation returns are integrated using the trapezoidal rule and normalised by the total training budget. This produces a single scalar metric that captures both learning speed and final performance, making it suitable for comparing activation functions across the full course of training rather than only at convergence.

Because reward scales differ substantially across environments, results are reported relative to a matched ReLU baseline. For each algorithm and task, the baseline corresponds to the standard TD3 or SAC implementation using ReLU activations under identical architectures and training conditions. Relative improvement is computed as

$$\Delta\% = \frac{\mathrm{AUC}_{\mathrm{config}} - \mathrm{AUC}_{\mathrm{ReLU}}}{|\mathrm{AUC}_{\mathrm{ReLU}}|} \times 100 \tag{9}$$

This normalisation allows results to be compared across tasks with different reward magnitudes while preserving the interpretation of gains and losses relative to the standard rectifier baseline.

Each configuration is trained with five independent random seeds to account for stochastic variation in training and evaluation. Aggregate results are summarised across seeds to provide a more reliable estimate of performance.

5 Results

This section reports the empirical evaluation of fractional activation functions in continuous-control RL. We analyse performance across benchmark tasks using TD3 and SAC under both single-layer and two-layer actor–critic architectures. Results are first presented in terms of overall performance across tasks, followed by analyses of fractional-order sensitivity and the interaction between activation functions and network structure in deeper actor–critic architectures. Unless stated otherwise, performance is reported relative to the ReLU baseline using the $\Delta\%$ metric defined in Section 4.3.

5.1 Overall Performance Across Tasks

The results indicate that fractional extensions of rectifier activations provide a flexible and effective modification for deep RL networks. Across multiple algorithms, architectures, and benchmark tasks, fractional activations frequently achieve stronger learning performance than standard rectifier functions.

The results indicate that fractional extensions of rectifier activations provide a flexible and effective modification for deep RL networks. Across multiple algorithms, architectures, and benchmark tasks, fractional activations frequently achieve stronger learning performance than standard rectifier functions.

Representative learning curves are shown in Figures 3, 4, 5, and 6. Each plot shows the average episode return obtained during training over one million environment steps. For each task, the curves correspond to the best-performing configuration of each activation family according to the experimental protocol. To facilitate direct comparison, conventional rectifier activations are plotted using solid lines, while their fractional counterparts are shown using dashed lines. This visual pairing highlights the effect of introducing the fractional order parameter α while preserving the overall structure of the corresponding rectifier activation.

Across many environments, fractional activations lead to faster early learning and improved final performance. These effects are particularly evident in locomotion tasks such as *HalfCheetah-v4*, *Hopper-v4*, and *Walker-Walk*, where fractional variants often reach stronger performance earlier during training. The results also reveal clear differences between the single-hidden-layer and two-hidden-layer architectures. In the single-layer setting, conventional rectifier activations remain competitive in some environments, although fractional variants still achieve improvements in several tasks. In contrast, in the two-layer architecture, fractional activations consistently outperform their non-fractional counterparts across most tasks. This pattern suggests that the additional flexibility introduced by the fractional order parameter α becomes increasingly beneficial as network depth increases.

To provide a clearer quantitative comparison, Figures 7a–8b present heatmap visualizations of the relative AUC improvement ($\Delta\%$) across activation functions and fractional orders. Each heatmap cell reports the best mean improvement relative to the ReLU baseline for each activation–task pair. For the two-layer architecture, the heatmaps additionally indicate the placement of the activation function within the actor and critic networks. These visualizations therefore illustrate the combined influence of activation type, fractional order, and placement strategy on performance.

Across most environments, fractional variants of rectifier activations frequently appear among the strongest-performing configurations. Particularly large improvements are observed in several locomotion tasks, including *HalfCheetah-v4*, *Hopper-v4*, and *Cheetah-Run*. For example, under TD3 with the single-layer architecture, fractional activations achieve improvements of up to +79.7% in *HalfCheetah-v4* and +65.3% in *Hopper-v4*. Similarly, in the two-layer SAC configuration, FPReLU achieves an improvement of +48.7% in *Ant-v4*. These results suggest that fractional nonlinearities can substantially improve learning efficiency in several continuous-control environments.

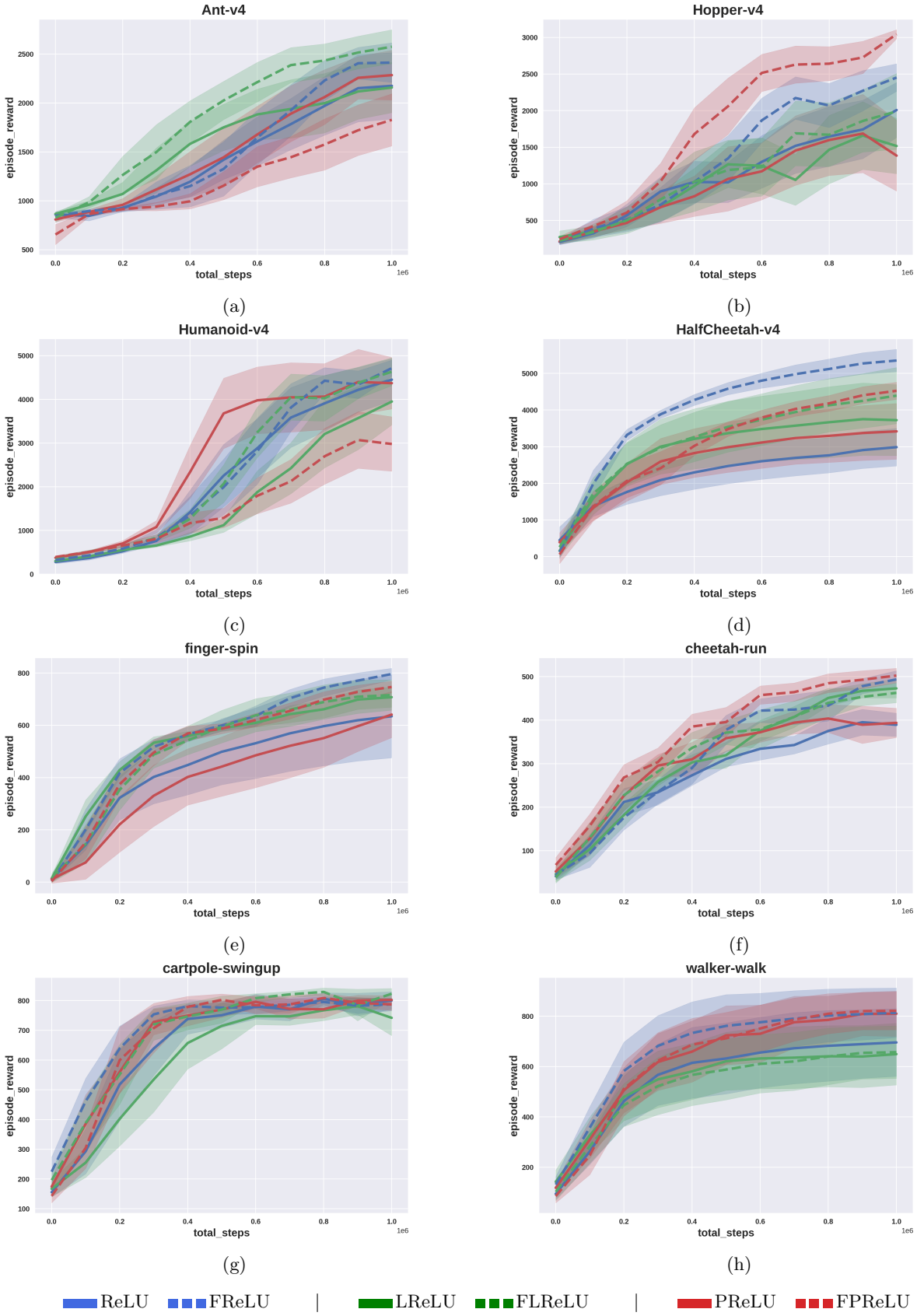


Figure 3: Learning curves for **TD3 with a single hidden layer**. Episode return is shown over one million training steps. Curves represent the mean across five seeds and shaded regions indicate standard deviation.

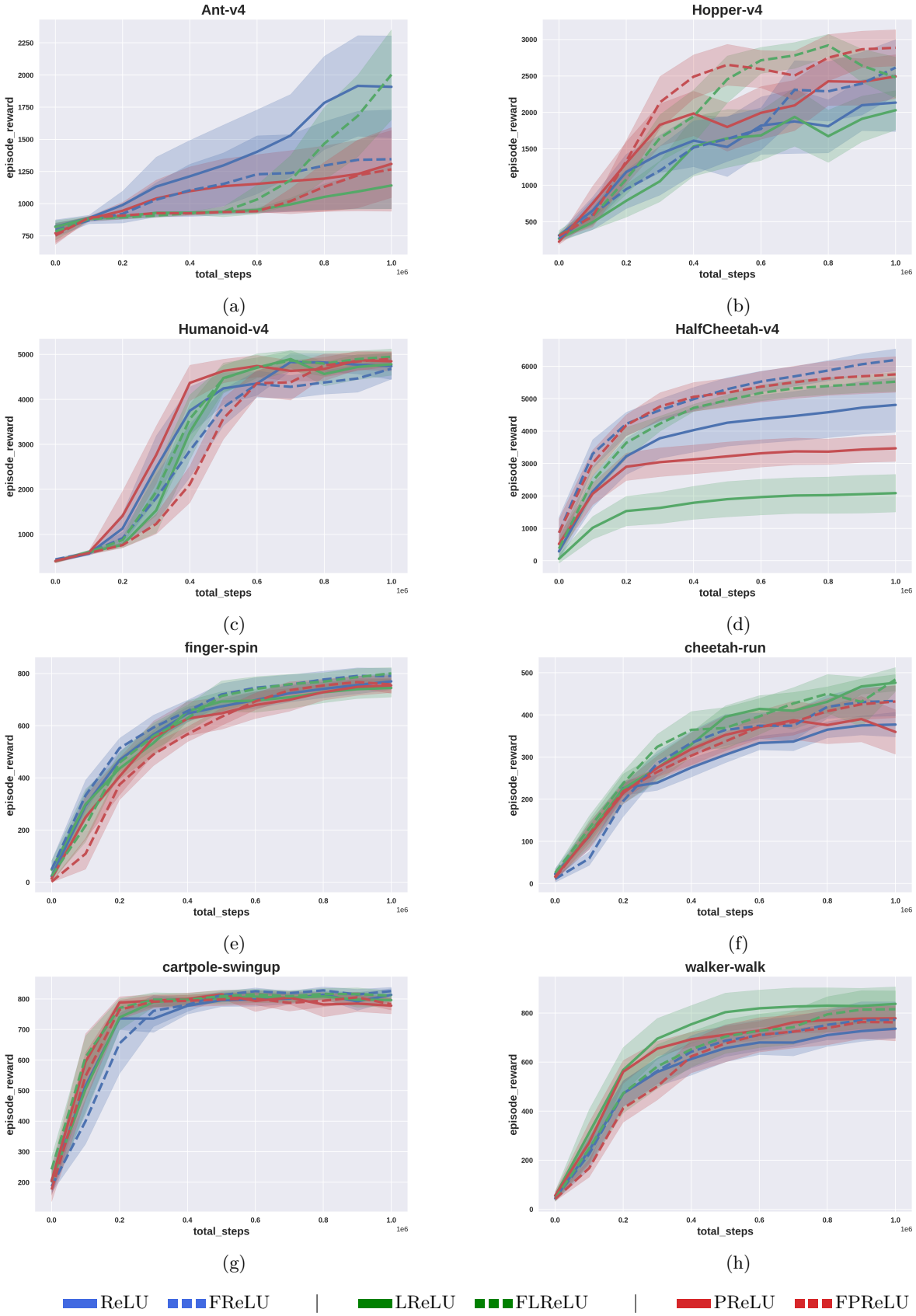
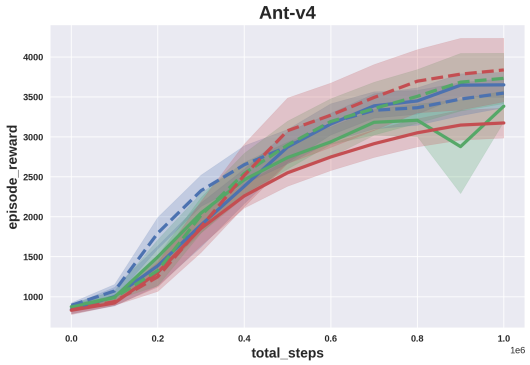
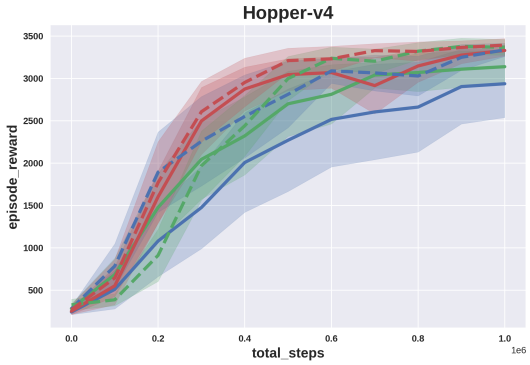


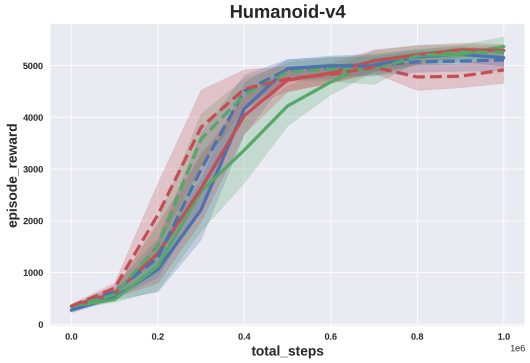
Figure 4: Learning curves for **SAC with a single hidden layer**. Performance is averaged across five random seeds and smoothed with a sliding window of 10 evaluation steps.



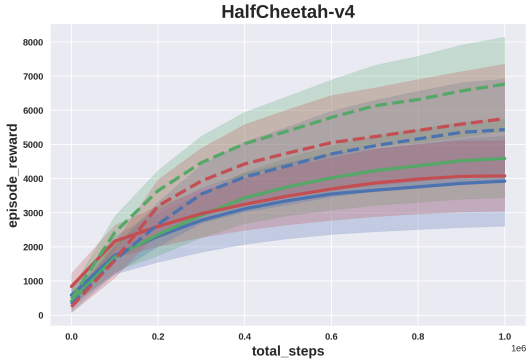
(a)



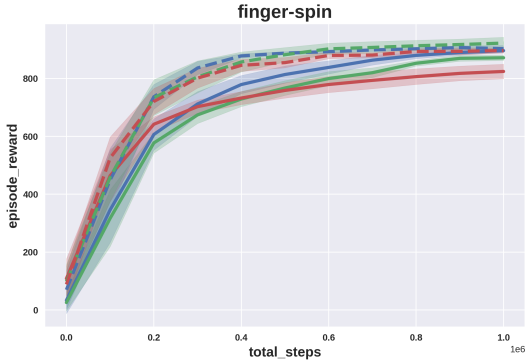
(b)



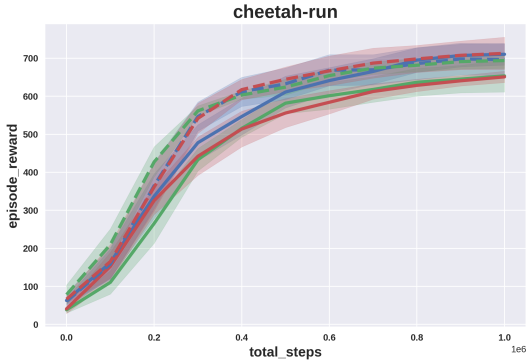
(c)



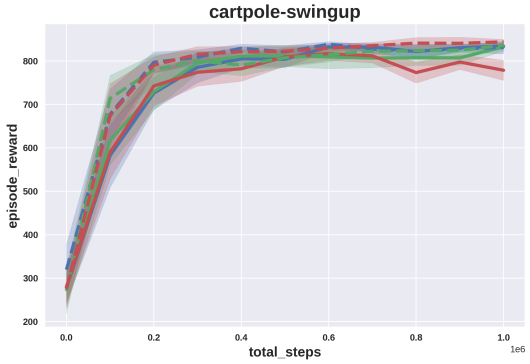
(d)



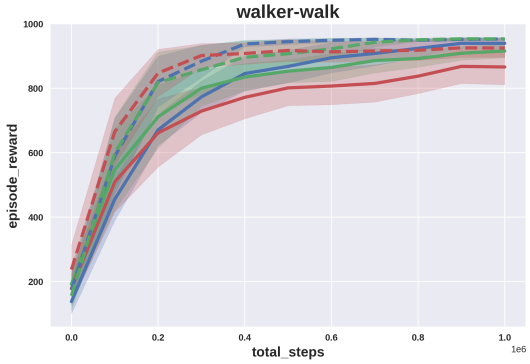
(e)



(f)



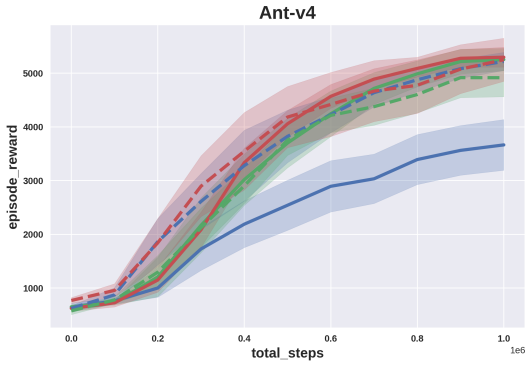
(g)



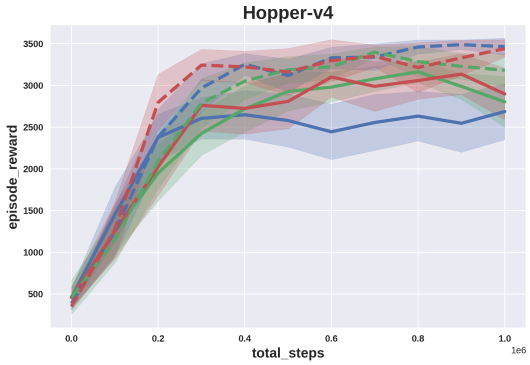
(h)

ReLU FReLU | LReLU FLReLU | PReLU FPReLU

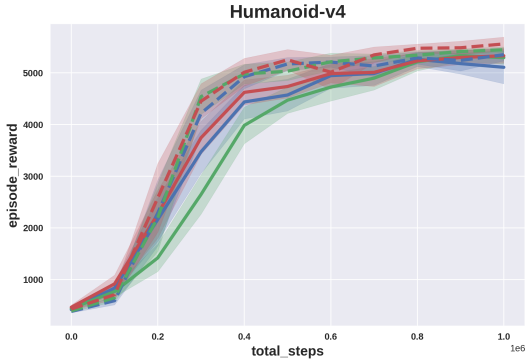
Figure 5: Learning curves for **TD3 with two hidden layers**. For each task, the best-performing activation configuration (including placement and α) is compared against the ReLU baseline.



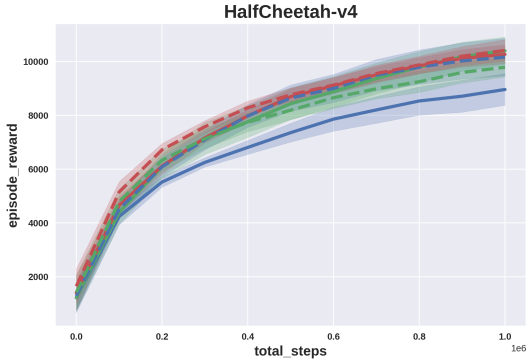
(a)



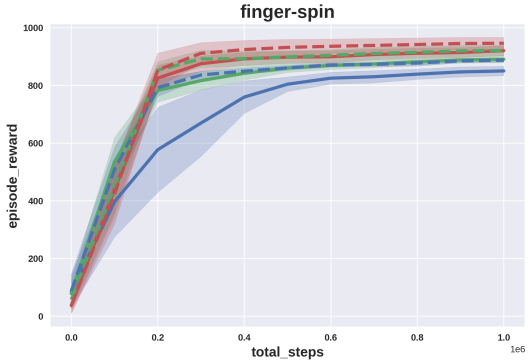
(b)



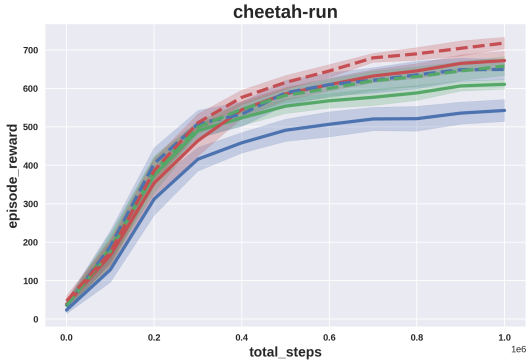
(c)



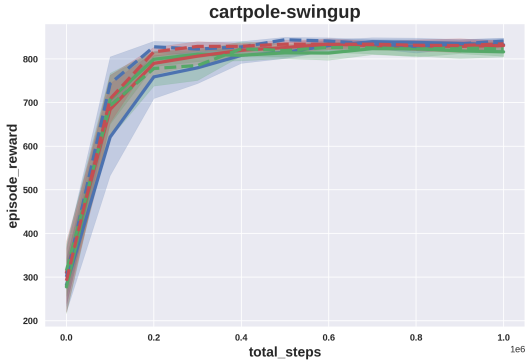
(d)



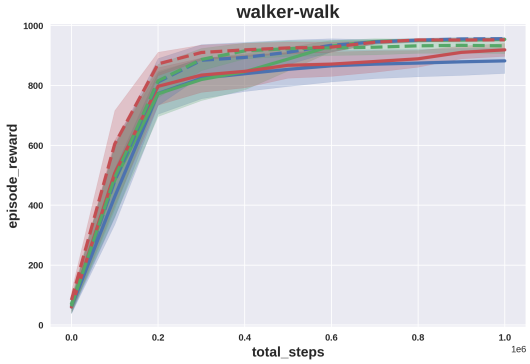
(e)



(f)



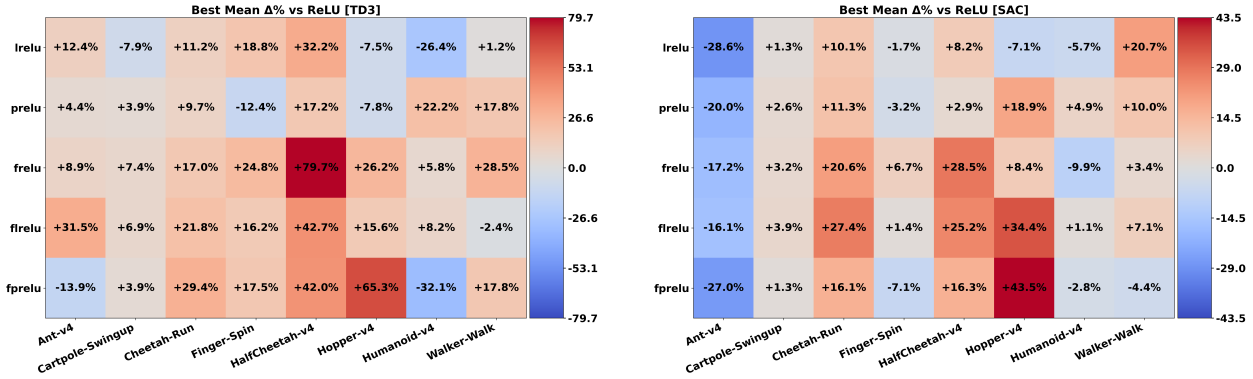
(g)



(h)

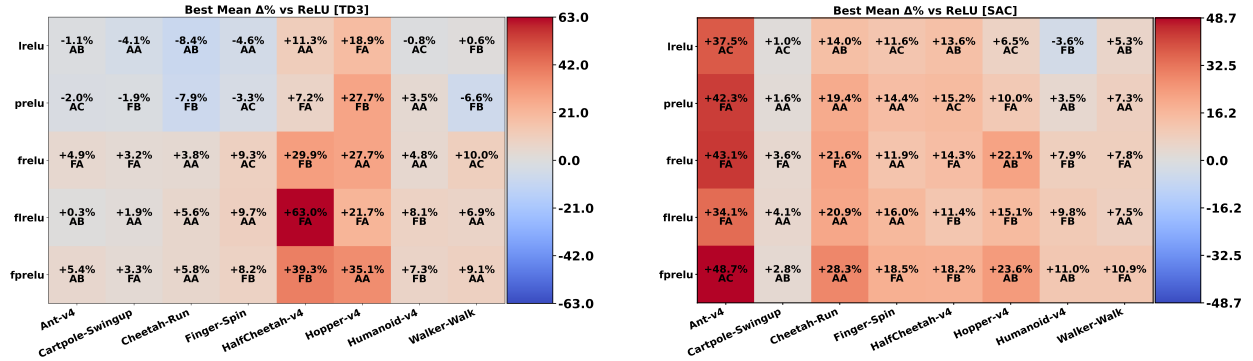
ReLU FReLU | LReLU FLReLU | PReLU FPReLU

Figure 6: Learning curves for **SAC with two hidden layers**. For each task, the best-performing activation configuration (including placement and α) is compared against the ReLU baseline.



(a) TD3 (b) SAC

Figure 7: Relative performance (Δ% AUC vs. ReLU baseline) for the **single-layer architecture**. Each cell shows the best-performing fractional order α for the corresponding activation function and task. Positive values indicate improvement over the ReLU baseline.



(a) TD3 (b) SAC

Figure 8: Relative performance (Δ% AUC vs. ReLU baseline) for the **two-layer architecture** using the best placement configuration for each task. Results highlight how fractional activations interact with deeper representations and activation placement. The placement of the activation function is indicated using the abbreviations defined in Section 3.4.

While the heatmaps provide a global view of performance across activation functions and fractional orders, Tables 2 and 3 identify the strongest configuration for each task. Table 2 reports results for the single-layer actor–critic architecture. Under TD3, fractional activations dominate most environments, with the best configurations consistently involving FReLU, FLReLU, or FPReLU. Large improvements appear in tasks such as *HalfCheetah-v4* (+79.7%) and *Hopper-v4* (+65.3%), demonstrating that fractional nonlinearities can significantly enhance performance even in relatively shallow networks.

Under SAC, the improvements are generally smaller but remain consistent across several environments. Fractional variants achieve the best results in tasks such as *Cheetah-Run*, *HalfCheetah-v4*, and *Hopper-v4*. An exception occurs in *Ant-v4*, where the ReLU baseline performs best under the single-layer SAC architecture and all activation variants produce negative Δ% values. However, this behaviour changes in the deeper architecture. In the two-layer setting, fractional activations substantially outperform the ReLU baseline in the same environment, with FPReLU achieving an improvement of +48.7%. This contrast highlights the importance of network depth and activation placement in exploiting the flexibility introduced by the fractional order parameter.

Table 2: Best activation configuration per task for the **single-layer architecture** under **TD3** and **SAC**. The table reports the activation, fractional order α, and relative AUC improvement ($\Delta\%$) compared with the ReLU baseline.

	TD3			**SAC**		
Task	**Act.**	$\boldsymbol{\alpha}$	$\boldsymbol{\Delta}\%$	**Act.**	$\boldsymbol{\alpha}$	$\boldsymbol{\Delta}\%$
Ant-v4	FLReLU	0.3	+31.5	ReLU	–	+0.0
Cartpole-Swingup	FReLU	0.2	+7.4	FLReLU	0.2	+3.9
Cheetah-Run	FPReLU	0.3	+29.4	FLReLU	0.1	+27.4
Finger-Spin	FReLU	0.1	+24.8	FReLU	0.3	+6.7
HalfCheetah-v4	FReLU	0.3	+79.7	FReLU	0.2	+28.5
Hopper-v4	FPReLU	0.1	+65.3	FPReLU	0.1	+43.5
Humanoid-v4	PReLU	–	+22.2	PReLU	–	+4.9
Walker-Walk	FReLU	0.1	+28.5	LReLU	–	+20.7

Table 3: Best activation configuration per task for the **two-layer architecture** under **TD3** and **SAC**. The table reports the activation, fractional order α, placement strategy, and relative AUC improvement ($\Delta\%$) compared with the ReLU baseline.

	TD3				**SAC**			
Task	**Act.**	$\boldsymbol{\alpha}$	**Place.**	$\boldsymbol{\Delta}\%$	**Act.**	$\boldsymbol{\alpha}$	**Place.**	$\boldsymbol{\Delta}\%$
Ant-v4	FPReLU	0.1	all-both	+5.38	FPReLU	0.2	all-critic	+48.72
Cartpole-Swingup	FPReLU	0.1	first-actor	+3.30	FLReLU	0.3	all-actor	+4.08
Cheetah-Run	FPReLU	0.4	all-actor	+5.79	FPReLU	0.3	all-actor	+28.34
Finger-Spin	FLReLU	0.1	all-actor	+9.72	FPReLU	0.3	first-actor	+18.49
HalfCheetah-v4	FLReLU	0.1	first-actor	+63.04	FPReLU	0.3	first-both	+18.15
Hopper-v4	FPReLU	0.1	all-actor	+35.09	FPReLU	0.2	all-both	+23.65
Humanoid-v4	FLReLU	0.2	first-both	+8.07	FPReLU	0.3	all-both	+10.96
Walker-Walk	FReLU	0.1	all-critic	+9.97	FPReLU	0.5	first-actor	+10.89

Table 3 presents the corresponding results for the two-layer architecture. Fractional activations again appear frequently among the best-performing configurations across both algorithms. The results also show that activation placement within the actor and critic networks can influence performance. For example, in SAC the strongest improvement in *Ant-v4* occurs when FPReLU is applied to the critic network, whereas actor-focused placements yield the largest gains in tasks such as *HalfCheetah-v4* and *Hopper-v4*. These observations suggest that the interaction between activation design and network structure becomes increasingly important as model depth increases.

Overall, the empirical results show that fractional rectifier activations improve learning behaviour across a wide range of RL settings. The consistent gains observed across architectures and environments indicate that introducing the fractional order parameter α enables more adaptable nonlinear transformations within actor–critic networks, improving both function approximation and training efficiency in deep RL..

5.2 Alpha Sensitivity and Robustness

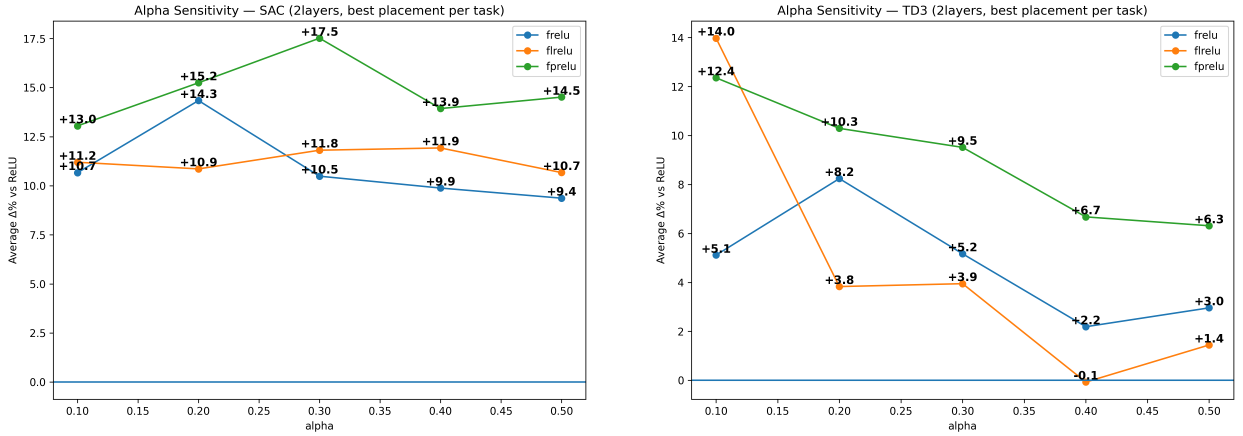
In this section, we evaluate the sensitivity of fractional activation functions to the fractional order parameter α. The parameter was introduced in Section 3.2.2, and here we analyse how different values influence learning performance.

Figure 9 presents the results for $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ for the three fractional activation families (FReLU, FLReLU, and FPReLU) in the two-layer architecture. The curves show the average improvement in nor-

Table 4: Preferred fractional order α per task in the two-layer experiments. For each task, the value corresponds to the fractional order that appears most often among the best-performing fractional activations.

Task	SAC Preferred α	TD3 Preferred α
Ant-v4	0.2	0.1
Cartpole-Swingup	0.5	0.5
Cheetah-Run	0.1	0.1
Finger-Spin	0.3	0.1
HalfCheetah-v4	0.3	0.1
Hopper-v4	0.2	0.1
Humanoid-v4	0.5	0.2
Walker-Walk	0.1	0.1

malised AUC ($\Delta\%$ relative to ReLU) aggregated across the benchmark tasks. To isolate the influence of the fractional parameter, the best activation placement configuration is selected for each task when computing the curves.



(a) SAC

(b) TD3

Figure 9: Sensitivity of performance to the fractional order α for SAC and TD3 in the two-layer architecture. Curves show the average improvement in normalised AUC relative to the ReLU baseline across tasks.

The results show that fractional activations maintain positive improvements across a range of α values, indicating stable behaviour with respect to the fractional order. However, the trends differ between the two RL algorithms. For TD3, the largest gains occur for small fractional orders, with the best performance typically around $\alpha = 0.1$ to $\alpha = 0.2$. For example, FLReLU achieves its highest improvement at $\alpha = 0.1$, while FReLU peaks near $\alpha = 0.2$. Performance gradually decreases as α increases, suggesting that TD3 benefits most from relatively mild deviations from the standard ReLU shape. In contrast, SAC exhibits a broader optimum across fractional orders. Moderate values of α, typically between 0.2 and 0.3, frequently produce the largest improvements. For example, FPReLU achieves its strongest improvement at $\alpha = 0.3$, while FReLU peaks at $\alpha = 0.2$. Unlike TD3, SAC maintains relatively stable performance even for larger values of α, indicating that the algorithm is less sensitive to the precise fractional order.

To further examine task-level behaviour, Table 4 reports the preferred fractional order for each environment individually. Across tasks, small fractional orders dominate the optimal configurations. In TD3, $\alpha = 0.1$ appears most frequently across environments, whereas SAC exhibits a slightly broader distribution with values typically between $\alpha = 0.2$ and $\alpha = 0.5$ depending on the task.

Table 5: Preferred fractional order α for each activation family in the two-layer experiments. Values correspond to the fractional order that most frequently appears among the best-performing configurations across the benchmark tasks.

Activation	SAC Preferred α	TD3 Preferred α
FReLU	0.2 (3 tasks)	0.1 (3 tasks)
FLReLU	0.3 (3 tasks)	0.1 (5 tasks)
FPReLU	0.3 (4 tasks)	0.1 (4 tasks)

Table 5 summarizes these trends at the activation-family level by reporting the preferred fractional order for each activation under SAC and TD3. A clear difference emerges between the algorithms. For TD3, $\alpha = 0.1$ consistently appears as the preferred value across all activation families, confirming that TD3 benefits from relatively mild fractional transformations. For SAC, slightly larger fractional orders are preferred, with $\alpha = 0.2$ most common for FReLU and $\alpha = 0.3$ appearing most frequently for both FLReLU and FPReLU.

Overall, these results indicate that fractional activations remain robust across a moderate range of α values. Nevertheless, the strongest improvements typically occur for small fractional orders, where the activation function remains close to the standard ReLU shape. Across both algorithms and tasks, values in the range $\alpha \in [0.1, 0.3]$ provide the most consistent improvements over the ReLU baseline.

5.3 Activation Functions and Placement Effects

While the previous sections analysed overall performance trends and the sensitivity to the fractional parameter α, this section examines how activation functions interact with network structure. In particular, we analyse two complementary aspects: the relative performance of different activation families across tasks, and the influence of activation placement within deeper actor–critic networks. Detailed numerical results for all configurations are reported in the Appendix.

5.3.1 Activation Family Comparison

We first compare the overall behaviour of the different activation families across the two-layer experiments. Table 6 summarises the average improvement relative to the ReLU baseline, the number of tasks where each activation outperforms ReLU, and the maximum observed gain across environments.

Table 6: Summary statistics of activation performance in the two-layer experiments. For each activation family we report the average improvement relative to the ReLU baseline across tasks, the number of tasks where the activation outperforms ReLU, and the maximum observed improvement.

Activation	Avg $\Delta\%$	Tasks > ReLU	Max $\Delta\%$
LReLU	+6.0	11 / 16	+37.54
PReLU	+9.2	11 / 16	+42.33
FReLU	+12.7	16 / 16	+43.08
FLReLU	+14.9	16 / 16	+63.04
FPReLU	+17.9	16 / 16	+48.72

Several clear patterns emerge. Fractional activations consistently outperform the standard ReLU baseline across all evaluated tasks, while conventional variants such as LReLU and PReLU occasionally underperform relative to ReLU. Among the tested nonlinearities, FPReLU achieves the highest average improvement (+17.9%), followed by FLReLU and FReLU.

The largest improvement occurs on the *HalfCheetah-v4* task, where FLReLU achieves a gain of +63.04%. These results indicate that fractional nonlinearities provide greater representational flexibility and can improve optimisation behaviour in deeper actor–critic architectures.

5.3.2 Effect of Network Depth and Placement

Network depth changes how activation functions influence learning. In the single-layer architecture, activation placement is fixed, so performance differences arise solely from the choice of nonlinearity. Even in this simple setting, fractional activations often improve learning behaviour compared to conventional variants, as illustrated in Figures 3 and 4.

Across most tasks, fractional activations lead to faster learning and higher final returns. An interesting exception appears in the *Ant* task under SAC. Although the AUC comparison in Figure 7 shows limited improvement relative to ReLU, the learning curves in Figure 4 reveal that fractional activations such as FLReLU achieve higher episode returns during the later stages of training. This suggests that performance differences measured by AUC may not fully capture the long-term advantages of certain nonlinearities.

When moving to the two-layer architecture, activation functions can be placed in different parts of the actor–critic network. As described in Section 3.4, we evaluate five placement strategies: *all-actor*, *all-critic*, *all-both*, *first-actor*, and *first-both*. This design allows us to analyse how the benefits of fractional activations depend not only on their functional form but also on where they are introduced within the network.

Table 7 summarises how often each placement strategy appears among the best-performing configurations across tasks.

Table 7: Frequency of activation placement appearing as the best configuration across tasks in the two-layer experiments. Counts are aggregated across all activations and tasks for each algorithm.

Placement Strategy	SAC (2-layer)	TD3 (2-layer)
all-actor	7	7
first-actor	8	3
all-critic	5	4
all-both	5	3
first-both	3	7

For SAC, the strongest improvements are typically observed when fractional activations are applied within the actor network. Placements such as `all-actor` and `first-actor` appear most frequently among the best configurations, suggesting that improving the policy representation plays a dominant role in entropy-regularised policy optimisation.

In contrast, TD3 shows a more balanced placement distribution. Configurations affecting both actor and critic networks, particularly `first-both`, appear more frequently among the best-performing setups. This pattern is consistent with the stronger dependence of TD3 on accurate critic estimation during policy updates.

5.3.3 Task-Dependent Placement Behaviour

We further analyse whether particular tasks consistently favour specific activation placements within the actor–critic architecture. Table 8 reports how often each placement strategy appears among the best configurations for each task across both algorithms and activation families.

Several patterns emerge. Locomotion tasks such as *Cheetah-Run* and *HalfCheetah-v4* frequently favour placements affecting the actor network, particularly `all-actor` or `first-actor`. This suggests that improving policy representations is especially important for tasks requiring smooth continuous control.

In contrast, more complex environments such as *Humanoid-v4* show a higher frequency of placements affecting both networks, including `first-both` and `all-both`. These tasks involve higher-dimensional state spaces and more challenging value estimation, making critic improvements more beneficial.

Overall, these results indicate that the impact of fractional activations depends not only on the activation family itself but also on how nonlinearities are distributed across the actor and critic networks.

Table 8: Most frequent best placement per task across both algorithms and activation families in the two-layer experiments.

Task	all-actor	first-actor	all-critic	all-both	first-both
Ant-v4	2	4	3	2	1
Cartpole-Swingup	3	2	2	1	2
Cheetah-Run	4	2	0	1	1
Finger-Spin	3	2	1	0	2
HalfCheetah-v4	2	3	1	1	1
Hopper-v4	2	2	1	2	1
Humanoid-v4	1	1	1	2	3
Walker-Walk	2	2	1	1	2

6 Conclusion

This paper investigated fractional-order extensions of ReLU-based activation functions in deep reinforcement learning. Three fractional variants, FReLU, FLReLU, and FPReLU, were evaluated alongside conventional rectifier activations, ReLU, LReLU, and PReLU, within two widely used actor–critic algorithms, TD3 and SAC, across continuous-control benchmarks from the MuJoCo and DeepMind Control Suite environments.

The experimental results show that fractional activations frequently outperform the standard ReLU baseline and its conventional variants across tasks and architectures. Among the proposed methods, FPReLU provides the most consistent improvements. Further analysis reveals that performance depends on both the fractional order parameter α and the placement of activation functions within deeper actor–critic networks. In particular, small fractional orders generally provide the most stable improvements, while activation placement influences how fractional nonlinearities interact with policy and value representations.

Taken together, these findings demonstrate that fractional rectifier activations offer a simple yet effective architectural modification for improving neural function approximation in RL. More broadly, the results highlight the potential of exploring alternative nonlinearities beyond conventional rectifier functions in deep RL architectures.

Broader Impact Statement

This work investigates fractional-order activation functions for RL in continuous-control tasks. Improvements in activation design may enhance the learning efficiency and representational capacity of neural networks used in RL algorithms, potentially benefiting applications in robotics, control systems, and autonomous decision-making.

RL methods may also be applied in sensitive domains such as autonomous systems, where responsible development and appropriate oversight are important. This research does not involve human subjects or personal data, and all experiments are conducted in standard simulated benchmark environments.

References

Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE signal processing magazine*, 34(6):26–38, 2017.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Finance*, 2(1):1–127, 2009.

A Bhatt, D Palenicek, B Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and J Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *International Conference on Learning Representations*, 2024.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

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

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323. JMLR Workshop and Conference Proceedings, 2011.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

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*, pp. 1861–1870. PMLR, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). In *International Conference on Learning Representations*, 2017.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning. In *International Conference on Learning Representations*, 2022.

Zhimo Jian, Chaoqian Luo, and Yajun Yin. Design and control of fractional-order systems based on fractal operators. *Fractal and Fractional*, 9(8):528, 2025.

Megha S Job, Priyanka H Bhateja, Muskan Gupta, Kishore Bingi, and B Rajanarayan Prusty. Fractional rectified linear unit activation function and its variants. *Mathematical Problems in Engineering*, 2022(1): 1860779, 2022.

Anatoly A. Kilbas, Hari M. Srivastava, and Juan J. Trujillo. *Theory and Applications of Fractional Differential Equations*, volume 204 of *North-Holland Mathematics Studies*. Elsevier Science, 2006.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Meshach Kumar, Utkal Mehta, and Giansalvo Cirrincione. Enhancing neural network classification using fractional-order activation functions. *AI Open*, 5:10–22, 2024.

Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International conference on machine learning*, pp. 5556–5566. PMLR, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Lu Lu, Yeonjong Shin, Yanhui Su, and George E. Karniadakis. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, 2020. doi: 10.4208/cicp.OA-2020-0165.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Atlanta, GA, 2013.

Kenneth S Miller and Bertram Ross. An introduction to the fractional calculus and fractional differential equations. *(No Title)*, 1993.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Vojtech Molek and Zahra Alijani. Fractional concepts in neural networks: Enhancing activation functions. *Pattern Recognition Letters*, 190:126–132, 2025.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Keith Oldham and Jerome Spanier. *The fractional calculus theory and applications of differentiation and integration to arbitrary order*, volume 111. Elsevier, 1974.

Manuel Duarte Ortigueira and Fernando Coito. From differences to derivatives. *Fractional Calculus and Applied Analysis*, 7(4):459, 2004.

Igor Podlubny. *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*, volume 198. elsevier, 1998.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017. URL `https://arxiv.org/abs/1710.05941`.

Stefan G Samko. Fractional integrals and derivatives. *Theory and applications*, 1993.

Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. second. *A Bradford Book*, 2018.

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 28694–28698, 2025.

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.

Juan Terven. Deep reinforcement learning: a chronological overview and methods. *AI*, 6(3):46, 2025.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

A Appendix

This appendix provides additional material supporting the experimental results presented in the main paper. Specifically, it reports complete performance tables, additional analyses of the fractional order parameter, and detailed task-level configurations for the evaluated activation functions. These extended results complement the summary analyses presented in the main text and improve the transparency and reproducibility of the experimental study.

A.1 Complete Experimental Results

This section presents the full performance tables for the single-layer and two-layer architectures evaluated in this study. For each task and activation variant, we report the area under the learning curve (AUC) accumulated over the full training horizon.

Performance values are aggregated across five random seeds using the interquartile mean (IQM), which provides a robust estimate of central performance while reducing sensitivity to outliers. Within each fractional activation family (FReLU, FLReLU, and FPReLU), the best-performing fractional order α is indicated in **bold**. The overall best configuration for each task, including the ReLU baseline, is highlighted in **purple**.

A.1.1 Single-Layer Architecture

Table 9 reports the complete results for the single-layer architecture under both TD3 and SAC agents.

A.1.2 Two-Layer Architectures

The following tables report the complete results for the two-layer networks across all evaluated activation placement strategies. Each table corresponds to a specific placement configuration within the actor–critic architecture.

A.2 Fractional Order Sensitivity Analysis

This section analyses the sensitivity of the proposed fractional activation functions to the fractional order parameter α. Fractional activations introduce a continuous parameter that controls the curvature of the nonlinear transformation and can therefore influence the behaviour of the neural networks used in the actor–critic agents.

To investigate this effect, each activation family (FReLU, FLReLU, and FPReLU) is evaluated across fractional orders $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For each value of α, the best performance improvement ($\Delta\%$) relative to the ReLU baseline is identified across all benchmark tasks. This procedure highlights the strongest achievable improvement associated with each fractional order while avoiding averaging effects that may obscure task-specific gains.

The resulting sensitivity curves for TD3 and SAC are shown in Figures 11 and 10. For visual comparability across activation families, the improvements are normalised with respect to the maximum absolute improvement observed for each activation type. This normalisation preserves the relative shape of the sensitivity curves while enabling the behaviour of different activation families to be compared on a common scale.

For TD3, the results indicate that the effectiveness of fractional activations depends on both the fractional order and the activation placement within the actor–critic architecture. Moderate fractional orders generally produce the most stable improvements, with values around $\alpha \approx 0.2$ frequently corresponding to the highest normalised performance for FReLU and FPReLU. Larger fractional orders often lead to reduced performance in several placements, suggesting that small fractional deviations from standard rectifier behaviour can provide beneficial nonlinear transformations while maintaining training stability.

A similar but more pronounced sensitivity is observed for SAC. Several activation families achieve stronger improvements at slightly larger fractional orders, typically around $\alpha \approx 0.3$–0.4. This behaviour suggests that SAC can benefit from stronger nonlinear transformations introduced by fractional activations, possibly due

Table 9: **Full results for the 1-layer architecture.** Normalised area under the learning curve (AUC, $\times 10^9$) up to total training steps (interquartile mean over five seeds). Best α per activation type (FReLU, FLReLU, FPReLU) is shown in **bold**; the overall best configuration for each task is highlighted in **purple**.

(a) TD3

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**1.51**	1.47	1.40	**1.62**	1.30	1.26	1.47	1.52	**1.93**	1.38	1.13	1.18	**1.23**	0.96	0.55	-0.02
Cartpole-Swingup	**0.66**	0.66	**0.71**	0.63	0.66	0.62	0.69	0.70	**0.70**	0.66	0.65	0.63	**0.70**	0.69	0.61	0.65
Cheetah-Run	**0.28**	0.29	0.31	0.32	**0.33**	0.30	0.32	0.31	0.31	**0.35**	0.31	0.29	0.29	0.37	**0.37**	0.31
Finger-Spin	**0.54**	**0.56**	0.50	0.50	0.44	0.52	0.51	**0.52**	0.40	0.18	0.21	**0.52**	0.37	0.32	0.26	0.14
HalfCheetah-v4	**2.40**	3.87	3.00	**4.00**	3.56	2.92	2.18	3.39	2.28	2.06	**3.66**	**3.19**	2.39	3.10	3.10	2.42
Hopper-v4	**1.17**	**1.32**	0.97	0.69	0.35	0.58	1.12	**1.27**	0.75	0.68	0.41	**1.79**	1.31	0.91	0.83	1.19
Humanoid-v4	**2.38**	2.03	2.23	**2.40**	0.91	0.76	2.36	**2.55**	2.28	1.37	0.83	1.60	1.06	0.81	**1.62**	0.73
Walker-Walk	**0.59**	**0.69**	0.55	0.60	0.42	0.60	0.50	0.50	0.44	**0.55**	0.27	0.62	**0.65**	0.47	0.52	0.34

(b) SAC

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**1.22**	**0.94**	0.91	0.92	0.91	0.90	**1.11**	0.92	0.92	0.92	0.91	0.91	**0.92**	0.91	0.91	0.90
Cartpole-Swingup	**0.73**	0.74	**0.75**	0.74	0.70	0.72	0.73	**0.76**	0.72	0.70	0.66	0.72	0.70	**0.74**	0.72	0.70
Cheetah-Run	**0.27**	**0.33**	0.32	0.31	0.32	0.30	**0.34**	0.31	0.32	0.32	0.31	**0.32**	0.26	0.30	0.30	0.27
Finger-Spin	**0.60**	0.59	0.59	**0.63**	0.60	0.57	0.54	0.58	0.59	**0.63**	0.57	**0.57**	0.53	0.56	0.53	0.55
HalfCheetah-v4	**4.30**	4.81	**4.89**	4.53	4.01	4.28	2.93	4.31	4.63	**4.86**	3.12	4.48	**4.65**	4.40	4.31	3.63
Hopper-v4	**1.42**	1.22	1.59	**1.64**	1.15	0.81	1.59	**1.99**	1.56	1.48	1.47	**2.10**	1.80	1.67	1.99	1.46
Humanoid-v4	**3.33**	**3.17**	2.97	2.94	2.61	2.61	**3.40**	3.11	3.11	3.39	2.63	3.08	**3.30**	2.41	2.49	2.93
Walker-Walk	**0.53**	**0.56**	0.51	0.56	0.49	0.54	**0.60**	0.50	0.59	0.54	0.48	0.50	**0.53**	0.46	0.49	0.41

Table 10: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **SAC** with **2layers** under placement **all-actor**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.37**	2.69	**2.92**	2.50	2.84	2.52	2.43	2.84	2.80	**3.37**	3.19	2.48	2.99	2.74	**3.41**	3.27
Cartpole-Swingup	**0.77**	0.76	0.78	0.78	**0.79**	0.77	0.78	0.77	**0.80**	0.78	0.78	0.77	0.77	0.76	**0.78**	0.77
Cheetah-Run	**0.44**	0.45	0.48	0.46	0.46	**0.50**	0.50	0.47	**0.50**	0.47	0.47	0.50	0.48	**0.54**	0.47	0.52
Finger-Spin	**0.73**	0.77	0.74	0.74	0.75	**0.79**	0.76	0.75	**0.81**	0.79	0.76	0.75	0.76	0.76	0.76	**0.78**
HalfCheetah-v4	**6.81**	7.18	6.52	**7.60**	6.82	5.80	6.75	**7.33**	5.43	6.45	6.97	6.57	7.45	7.36	**7.52**	7.45
Hopper-v4	**2.35**	2.27	2.39	2.34	2.50	**2.66**	2.56	2.40	2.53	**2.63**	2.35	2.19	2.53	**2.85**	2.37	2.67
Humanoid-v4	**3.83**	3.57	3.70	**3.91**	3.66	3.87	3.67	3.58	3.76	3.82	**3.95**	3.72	3.63	3.32	**3.81**	3.71
Walker-Walk	**0.78**	0.79	0.76	**0.81**	0.78	0.70	0.84	**0.86**	0.82	0.77	0.81	0.83	0.77	**0.85**	0.84	0.84

Table 11: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **SAC** with **2layers** under placement **all-both**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.37**	**2.45**	2.36	2.07	1.47	1.03	**2.70**	2.21	2.49	1.61	1.27	2.31	**2.99**	1.41	1.14	1.06
Cartpole-Swingup	**0.77**	**0.78**	0.74	0.77	0.77	0.76	**0.78**	0.77	0.75	0.78	0.77	0.74	0.77	0.76	0.77	**0.79**
Cheetah-Run	**0.44**	**0.44**	0.43	0.39	0.39	0.33	0.45	**0.46**	0.41	0.41	0.37	0.44	**0.45**	0.41	0.39	0.36
Finger-Spin	**0.73**	0.76	0.77	**0.78**	0.77	0.68	**0.78**	0.75	0.74	0.77	0.74	0.71	**0.78**	0.73	0.74	0.73
HalfCheetah-v4	**6.81**	**6.90**	6.47	6.17	5.72	5.04	7.19	6.28	**7.25**	6.51	6.66	6.52	**7.12**	6.14	4.90	4.74
Hopper-v4	**2.35**	2.51	**2.86**	2.38	2.53	2.48	2.63	2.34	1.97	2.44	**2.66**	2.56	**2.99**	2.74	2.71	2.88
Humanoid-v4	**3.83**	3.73	3.52	3.58	**3.87**	3.56	3.98	3.65	3.70	**4.12**	3.71	3.98	3.87	**4.25**	4.07	4.17
Walker-Walk	**0.78**	0.77	**0.81**	0.75	0.77	0.59	0.77	0.75	**0.80**	0.75	0.57	0.67	0.63	0.77	0.67	**0.78**

Table 12: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **SAC** with **2layers** under placement **all-critic**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.37**	**3.17**	2.39	2.14	1.78	1.20	**2.38**	2.07	2.04	1.12	1.27	3.17	**3.76**	3.45	1.36	1.20
Cartpole-Swingup	**0.77**	0.77	0.77	0.78	**0.78**	0.78	0.78	0.78	0.76	**0.78**	0.77	0.74	0.76	0.77	**0.78**	0.78
Cheetah-Run	**0.44**	**0.46**	0.43	0.41	0.37	0.38	**0.45**	0.44	0.42	0.38	0.38	0.39	**0.42**	0.39	0.38	0.36
Finger-Spin	**0.73**	0.78	**0.79**	0.73	0.74	0.69	0.76	0.76	**0.79**	0.73	0.74	0.70	0.73	0.76	**0.76**	0.73
HalfCheetah-v4	**6.81**	7.13	**7.31**	6.11	7.01	6.79	7.39	**7.67**	6.91	6.46	5.91	**6.45**	5.76	6.05	3.78	5.48
Hopper-v4	**2.35**	**2.76**	2.64	2.52	2.21	2.27	2.39	**2.55**	2.36	2.38	2.35	**2.76**	2.75	2.43	2.36	2.68
Humanoid-v4	**3.83**	3.50	3.53	3.39	**3.60**	3.50	3.45	3.53	3.57	3.52	**3.76**	3.97	3.92	4.04	3.80	**4.10**
Walker-Walk	**0.78**	**0.75**	0.74	0.73	0.54	0.69	0.78	0.73	**0.83**	0.71	0.69	0.62	**0.71**	0.67	0.68	0.63

Table 13: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **SAC** with **2layers** under placement **first-actor**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.37**	3.15	3.48	**3.48**	2.99	2.58	2.35	3.12	2.56	3.18	**3.22**	2.16	**3.37**	2.75	2.57	3.26
Cartpole-Swingup	**0.77**	0.77	0.78	0.75	0.79	**0.79**	0.77	0.78	0.78	**0.80**	0.78	0.78	0.78	0.78	**0.79**	0.77
Cheetah-Run	**0.44**	**0.52**	0.49	0.49	0.50	0.49	0.48	0.48	0.47	0.49	**0.50**	0.43	0.48	0.48	**0.50**	0.48
Finger-Spin	**0.73**	0.76	0.76	0.76	0.72	**0.78**	**0.77**	0.76	0.76	0.75	0.75	0.74	0.75	**0.84**	0.79	0.78
HalfCheetah-v4	**6.81**	7.56	**8.11**	6.90	6.96	7.28	7.24	7.13	**7.26**	7.08	7.19	7.34	7.35	**7.58**	7.38	6.83
Hopper-v4	**2.35**	2.58	2.41	2.38	**2.74**	2.49	2.28	2.49	2.51	**2.54**	2.36	2.54	**2.61**	2.56	2.51	2.61
Humanoid-v4	**3.83**	3.62	**3.82**	3.57	3.60	3.68	3.59	3.59	**3.66**	3.63	3.51	3.66	3.68	3.73	3.57	**3.87**
Walker-Walk	**0.78**	**0.85**	0.81	0.74	0.83	0.84	0.83	0.80	**0.84**	0.78	0.70	0.80	0.71	0.81	0.81	**0.85**

Table 14: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **SAC** with **2layers** under placement **first-both**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.37**	**3.07**	2.88	1.87	1.38	1.59	**2.88**	2.76	1.42	1.15	1.19	**3.34**	2.11	2.39	1.24	1.41
Cartpole-Swingup	**0.77**	0.78	0.78	0.78	0.78	**0.78**	**0.79**	0.77	0.78	0.77	0.79	0.76	**0.79**	0.78	0.77	0.78
Cheetah-Run	**0.44**	**0.51**	0.46	0.36	0.48	0.43	0.47	0.43	0.46	**0.48**	0.44	0.47	0.46	0.46	**0.49**	0.46
Finger-Spin	**0.73**	0.78	0.75	0.78	0.76	**0.79**	**0.78**	0.77	0.76	0.78	0.78	0.77	0.77	**0.79**	0.79	0.71
HalfCheetah-v4	**6.81**	**7.23**	6.85	7.01	6.70	5.89	7.35	6.54	**7.75**	7.25	6.49	7.39	7.63	**8.22**	7.49	6.91
Hopper-v4	**2.35**	2.47	2.39	2.11	2.61	**2.64**	2.64	**2.77**	2.72	2.67	2.67	2.33	**2.87**	2.53	2.31	2.69
Humanoid-v4	**3.83**	3.70	3.72	3.72	3.76	**4.15**	3.56	3.75	3.67	3.83	**4.19**	**4.25**	4.05	4.19	4.06	4.06
Walker-Walk	**0.78**	0.79	0.80	**0.81**	0.75	0.68	**0.85**	0.80	0.83	0.82	0.78	0.74	**0.82**	0.80	0.72	0.60

Table 15: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **TD3** with **2layers** under placement **all-actor**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.54**	2.19	**2.44**	2.23	2.12	2.17	2.39	**2.44**	2.43	2.22	2.39	2.33	**2.33**	2.15	2.15	2.17
Cartpole-Swingup	**0.75**	0.75	0.76	0.74	**0.77**	0.75	0.75	0.74	0.76	0.75	**0.77**	0.76	**0.76**	0.74	0.76	0.75
Cheetah-Run	**0.52**	**0.56**	0.52	0.51	0.55	0.53	**0.55**	0.48	0.55	0.52	0.49	0.47	0.52	0.54	**0.57**	0.52
Finger-Spin	**0.72**	0.69	0.73	0.77	**0.78**	0.76	**0.78**	0.77	0.76	0.77	0.76	0.23	0.61	0.60	0.66	**0.71**
HalfCheetah-v4	**2.69**	2.41	**3.54**	2.63	2.80	2.04	1.54	**2.70**	2.44	2.63	2.67	2.60	3.56	**4.47**	2.69	3.61
Hopper-v4	**2.16**	1.76	2.28	1.91	1.67	**2.62**	1.90	2.25	**2.25**	2.18	1.57	**2.72**	2.43	2.36	2.32	2.47
Humanoid-v4	**3.51**	3.60	3.65	3.62	**3.74**	3.46	3.51	3.78	3.47	**3.78**	3.76	0.92	3.04	3.27	**3.71**	3.32
Walker-Walk	**0.79**	0.70	0.77	**0.86**	0.80	0.77	**0.87**	0.62	0.81	0.67	0.82	0.85	0.81	**0.89**	0.87	0.89

Table 16: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **TD3** with **2layers** under placement **all-both**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.54**	**2.41**	1.97	2.37	1.05	0.90	2.42	**2.43**	1.94	1.07	0.86	**2.44**	2.28	1.55	0.24	-0.02
Cartpole-Swingup	**0.75**	0.75	**0.76**	0.75	0.76	0.74	0.75	0.73	**0.77**	0.75	0.76	0.75	0.75	0.73	**0.76**	0.74
Cheetah-Run	**0.52**	0.50	0.47	0.50	**0.51**	0.42	0.48	**0.51**	0.48	0.47	0.43	0.47	**0.50**	0.49	0.46	0.41
Finger-Spin	**0.72**	0.71	**0.76**	0.69	0.63	0.55	**0.78**	0.73	0.42	0.62	0.52	0.63	**0.69**	0.58	0.67	0.46
HalfCheetah-v4	**2.69**	2.75	1.53	2.31	**3.74**	2.61	1.56	1.60	**3.00**	1.20	2.99	**3.00**	1.71	2.62	2.76	2.79
Hopper-v4	**2.16**	**2.21**	1.45	1.96	1.52	1.37	1.94	1.90	1.69	**2.03**	1.79	**2.72**	2.43	2.36	2.32	2.47
Humanoid-v4	**3.51**	3.50	**3.88**	3.16	2.26	1.11	**3.64**	3.44	3.15	3.35	2.48	2.36	**3.60**	1.87	2.38	0.99
Walker-Walk	**0.79**	0.81	0.75	0.76	**0.81**	0.67	0.79	**0.80**	0.77	0.79	0.61	0.78	**0.85**	0.70	0.73	0.68

Table 17: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **TD3** with **2layers** under placement **all-critic**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.54**	**2.60**	2.50	2.13	2.27	1.33	2.33	**2.43**	2.38	2.19	1.21	**2.48**	2.43	1.97	0.82	0.40
Cartpole-Swingup	**0.75**	0.75	0.74	0.76	**0.76**	0.76	0.76	**0.77**	0.77	0.76	0.74	0.75	0.76	**0.76**	0.74	0.73
Cheetah-Run	**0.52**	0.47	0.49	**0.51**	0.49	0.49	**0.53**	0.52	0.50	0.48	0.48	0.47	**0.47**	0.46	0.43	0.39
Finger-Spin	**0.72**	0.78	**0.79**	0.70	0.63	0.53	0.70	0.74	**0.77**	0.63	0.58	**0.76**	0.72	0.74	0.66	0.50
HalfCheetah-v4	**2.69**	**1.56**	1.54	1.34	1.40	1.15	1.66	1.56	2.21	**2.21**	1.29	**4.15**	3.01	3.29	4.00	1.91
Hopper-v4	**2.16**	1.95	1.49	**2.37**	1.95	1.63	1.31	**2.05**	1.94	1.85	1.70	**2.45**	2.29	2.33	2.25	2.42
Humanoid-v4	**3.51**	**3.82**	3.32	3.03	2.32	0.54	3.47	**3.51**	3.24	2.16	0.67	**3.67**	2.02	3.20	2.22	0.36
Walker-Walk	**0.79**	**0.88**	0.85	0.79	0.68	0.66	0.73	0.81	**0.85**	0.79	0.62	**0.87**	0.81	0.86	0.85	0.75

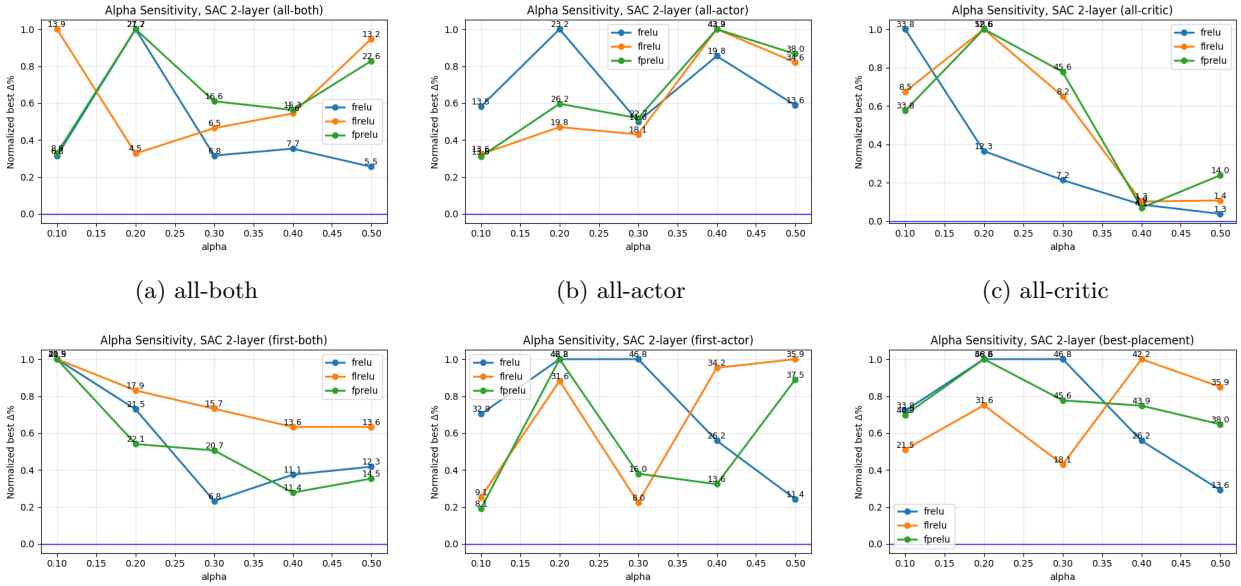
Table 18: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **TD3** with **2layers** under placement **first-actor**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.54**	2.34	2.31	**2.59**	2.24	2.52	2.32	2.35	2.42	**2.51**	2.08	2.31	2.36	2.33	2.38	**2.54**
Cartpole-Swingup	**0.75**	0.76	0.75	0.75	0.77	**0.78**	0.70	**0.73**	0.71	0.70	0.70	**0.78**	0.75	0.76	0.77	0.76
Cheetah-Run	**0.52**	**0.54**	0.52	0.51	0.49	0.49	0.49	0.49	**0.54**	0.51	0.53	0.51	0.53	**0.54**	0.46	0.48
Finger-Spin	**0.72**	0.71	**0.75**	0.72	0.72	0.70	0.73	0.71	0.76	0.73	**0.76**	0.70	0.72	0.73	0.72	**0.75**
HalfCheetah-v4	**2.69**	1.51	2.62	2.97	**3.81**	2.73	**5.47**	2.73	1.69	2.46	2.61	1.60	1.71	2.44	1.93	**2.66**
Hopper-v4	**2.16**	1.65	**2.26**	1.53	1.73	2.16	**2.35**	2.12	2.14	1.83	2.08	**2.33**	1.87	2.08	2.19	1.92
Humanoid-v4	**3.51**	1.75	1.31	1.20	1.45	**1.95**	3.38	3.43	**3.68**	3.43	3.42	3.45	3.36	**3.74**	3.53	3.58
Walker-Walk	**0.79**	0.76	**0.87**	0.84	0.74	0.75	**0.80**	0.75	0.78	0.76	0.78	0.83	0.84	**0.86**	0.84	0.81

Table 19: Normalised area under the learning curve (AUC) ($\times 10^9$) (interquartile mean over five seeds) for **TD3** with **2layers** under placement **first-both**. Best α per activation type in **bold**; overall best per task in **purple**.

Task	ReLU	FReLU					FLReLU					FPReLU				
		α=0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
Ant-v4	**2.54**	**2.56**	2.33	2.26	1.99	1.95	**2.50**	2.39	2.37	1.95	1.30	2.43	**2.49**	2.49	1.83	1.23
Cartpole-Swingup	**0.75**	0.75	**0.76**	0.74	0.75	0.75	0.75	0.75	**0.75**	0.74	0.74	0.76	0.77	0.74	**0.77**	0.75
Cheetah-Run	**0.52**	**0.53**	0.53	0.50	0.51	0.47	0.49	0.45	0.44	**0.52**	0.49	0.52	0.49	**0.53**	0.52	0.49
Finger-Spin	**0.72**	0.70	**0.72**	0.66	0.52	0.68	**0.75**	0.71	0.74	0.69	0.64	0.75	0.71	**0.78**	0.69	0.62
HalfCheetah-v4	**2.69**	1.53	**3.87**	1.46	1.51	1.40	2.60	**2.80**	2.23	1.52	1.37	3.22	4.09	**4.12**	1.55	2.07
Hopper-v4	**2.16**	1.79	**2.10**	1.57	1.70	1.73	2.05	**2.09**	1.88	1.80	1.57	**2.55**	2.32	2.32	2.40	1.91
Humanoid-v4	**3.51**	3.38	3.35	**3.50**	3.00	2.07	3.68	**3.84**	3.30	3.42	2.42	**3.94**	3.76	3.25	3.23	1.66
Walker-Walk	**0.79**	**0.82**	0.68	0.74	0.74	0.75	0.82	0.71	**0.84**	0.74	0.82	0.81	0.73	**0.82**	0.68	0.71

to the entropy-regularised optimisation dynamics of the algorithm. Across placement strategies, the first-actor configuration frequently yields strong improvements, indicating that modifying the early nonlinear transformations in the policy network can significantly influence learning behaviour.



(a) all-both (b) all-actor (c) all-critic

(d) first-both (e) first-actor (f) best-placement

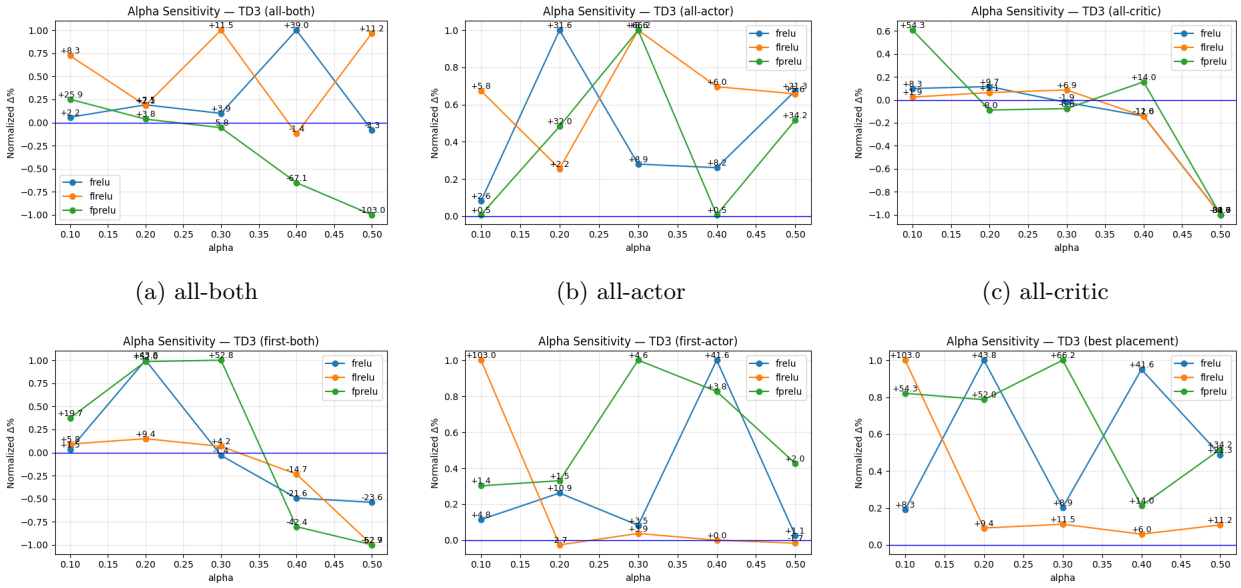
Figure 10: Task-wise fractional order sensitivity for SAC with two-layer networks across different placement strategies. Each panel reports the best relative improvement over the ReLU baseline across tasks as a function of α for FReLU, FLReLU, and FPReLU. Values are normalised within each activation family to highlight relative sensitivity patterns.

A.3 Best Task-Level Configurations for TD3 and SAC

This section reports the best-performing configuration for each activation family on every benchmark task for both TD3 and SAC agents. The tables summarise the fractional order α, the activation placement within the network, and the resulting performance improvement ($\Delta\%$) relative to the ReLU baseline.

Unlike the aggregate analyses presented in the main text, which summarise results across environments, these tables provide a task-level view of the experimental outcomes. For each task and activation family, the configuration that yields the largest improvement over the ReLU baseline is reported. This representation highlights the strongest achievable gains and allows differences in optimal configurations across tasks, algorithms, and network architectures to be examined more clearly.

The results indicate that the preferred fractional order and activation placement vary across environments and between TD3 and SAC. In several tasks, fractional activations achieve notable improvements over conventional rectifier functions, particularly when combined with appropriate placement strategies in deeper networks. Providing these detailed configurations improves the transparency of the experimental evaluation and facilitates reproducibility of the reported results.



(a) all-both (b) all-actor (c) all-critic

(d) first-both (e) first-actor (f) best-placement

Figure 11: Task-wise fractional order sensitivity for TD3 with two-layer networks across different placement strategies. Each panel shows the best relative improvement over the ReLU baseline across tasks as a function of α for FReLU, FLReLU, and FPReLU. Values are normalised within each activation family to highlight relative sensitivity patterns.

Table 20: Best configuration per activation per task (TD3, 1-layer), mean AUC vs ReLU baseline (placement r when available).

Task	Activation	Best α	Best placement	$\Delta\%$
Ant-v4	relu	base	none	+0.00
Ant-v4	lrelu	base	none	+12.41
Ant-v4	prelu	base	none	+4.38
Ant-v4	frelu	0.3	none	+8.89
Ant-v4	flrelu	0.3	none	+31.54
Ant-v4	fprelu	0.2	none	-13.94
Cartpole-Swingup	relu	base	none	+0.00
Cartpole-Swingup	lrelu	base	none	-7.89
Cartpole-Swingup	prelu	base	none	+3.91
Cartpole-Swingup	frelu	0.2	none	+7.42
Cartpole-Swingup	flrelu	0.3	none	+6.93
Cartpole-Swingup	fprelu	0.2	none	+3.90
Cheetah-Run	relu	base	none	+0.00
Cheetah-Run	lrelu	base	none	+11.25
Cheetah-Run	prelu	base	none	+9.66
Cheetah-Run	frelu	0.4	none	+16.99
Cheetah-Run	flrelu	0.4	none	+21.78
Cheetah-Run	fprelu	0.3	none	+29.43
Finger-Spin	relu	base	none	+0.00
Finger-Spin	lrelu	base	none	+18.85
Finger-Spin	prelu	base	none	-12.42
Finger-Spin	frelu	0.1	none	+24.80
Finger-Spin	flrelu	0.2	none	+16.17
Finger-Spin	fprelu	0.1	none	+17.53
HalfCheetah-v4	relu	base	none	+0.00
HalfCheetah-v4	lrelu	base	none	+32.22
HalfCheetah-v4	prelu	base	none	+17.17
HalfCheetah-v4	frelu	0.3	none	+79.67
HalfCheetah-v4	flrelu	0.5	none	+42.69
HalfCheetah-v4	fprelu	0.3	none	+42.03
Hopper-v4	relu	base	none	+0.00
Hopper-v4	lrelu	base	none	-7.55
Hopper-v4	prelu	base	none	-7.84
Hopper-v4	frelu	0.1	none	+26.17
Hopper-v4	flrelu	0.2	none	+15.63
Hopper-v4	fprelu	0.1	none	+65.26
Humanoid-v4	relu	base	none	+0.00
Humanoid-v4	lrelu	base	none	-26.40
Humanoid-v4	prelu	base	none	+22.17
Humanoid-v4	frelu	0.3	none	+5.76
Humanoid-v4	flrelu	0.2	none	+8.15
Humanoid-v4	fprelu	0.1	none	-32.10
Walker-Walk	relu	base	none	+0.00
Walker-Walk	lrelu	base	none	+1.16
Walker-Walk	prelu	base	none	+17.85
Walker-Walk	frelu	0.1	none	+28.48
Walker-Walk	flrelu	0.4	none	-2.41
Walker-Walk	fprelu	0.2	none	+17.76

Table 21: Best configuration per activation per task (TD3, 2-layer), mean AUC vs ReLU baseline (placement r when available).

Task	Activation	Best α	Best placement	$\Delta\%$
Ant-v4	relu	base	r	+0.00
Ant-v4	lrelu	base	all-both	-1.14
Ant-v4	prelu	base	all-critic	-2.03
Ant-v4	frelu	0.3	first-actor	+4.88
Ant-v4	flrelu	0.1	all-both	+0.33
Ant-v4	fprelu	0.1	all-both	+5.38
Cartpole-Swingup	relu	base	r	+0.00
Cartpole-Swingup	lrelu	base	all-actor	-4.10
Cartpole-Swingup	prelu	base	first-both	-1.89
Cartpole-Swingup	frelu	0.5	first-actor	+3.24
Cartpole-Swingup	flrelu	0.5	all-actor	+1.94
Cartpole-Swingup	fprelu	0.1	first-actor	+3.30
Cheetah-Run	relu	base	r	+0.00
Cheetah-Run	lrelu	base	all-both	-8.42
Cheetah-Run	prelu	base	first-both	-7.86
Cheetah-Run	frelu	0.1	all-actor	+3.83
Cheetah-Run	flrelu	0.1	all-actor	+5.64
Cheetah-Run	fprelu	0.4	all-actor	+5.79
Finger-Spin	relu	base	r	+0.00
Finger-Spin	lrelu	base	all-actor	-4.65
Finger-Spin	prelu	base	all-critic	-3.31
Finger-Spin	frelu	0.1	all-critic	+9.34
Finger-Spin	flrelu	0.1	all-actor	+9.72
Finger-Spin	fprelu	0.3	first-both	+8.19
HalfCheetah-v4	relu	base	r	+0.00
HalfCheetah-v4	lrelu	base	all-actor	+11.27
HalfCheetah-v4	prelu	base	first-actor	+7.17
HalfCheetah-v4	frelu	0.2	first-both	+29.91
HalfCheetah-v4	flrelu	0.1	first-actor	+63.04
HalfCheetah-v4	fprelu	0.2	first-both	+39.27
Hopper-v4	relu	base	r	+0.00
Hopper-v4	lrelu	base	first-actor	+18.86
Hopper-v4	prelu	base	first-both	+27.69
Hopper-v4	frelu	0.5	all-actor	+27.73
Hopper-v4	flrelu	0.1	first-actor	+21.73
Hopper-v4	fprelu	0.1	all-actor	+35.09
Humanoid-v4	relu	base	r	+0.00
Humanoid-v4	lrelu	base	all-critic	-0.81
Humanoid-v4	prelu	base	all-actor	+3.51
Humanoid-v4	frelu	0.4	all-actor	+4.79
Humanoid-v4	flrelu	0.2	first-both	+8.07
Humanoid-v4	fprelu	0.1	first-both	+7.26
Walker-Walk	relu	base	r	+0.00
Walker-Walk	lrelu	base	first-both	+0.58
Walker-Walk	prelu	base	first-both	-6.64
Walker-Walk	frelu	0.1	all-critic	+9.97
Walker-Walk	flrelu	0.1	all-actor	+6.93
Walker-Walk	fprelu	0.5	all-actor	+9.10

Table 22: Best configuration per activation per task (SAC, 1-layer), mean AUC vs ReLU baseline (placement r when available).

Task	Activation	Best α	Best placement	$\Delta\%$
Ant-v4	relu	base	none	+0.00
Ant-v4	lrelu	base	none	-28.55
Ant-v4	prelu	base	none	-19.98
Ant-v4	frelu	0.2	none	-17.20
Ant-v4	flrelu	0.1	none	-16.10
Ant-v4	fprelu	0.3	none	-27.00
Cartpole-Swingup	relu	base	none	+0.00
Cartpole-Swingup	lrelu	base	none	+1.29
Cartpole-Swingup	prelu	base	none	+2.55
Cartpole-Swingup	frelu	0.2	none	+3.21
Cartpole-Swingup	flrelu	0.2	none	+3.94
Cartpole-Swingup	fprelu	0.3	none	+1.27
Cheetah-Run	relu	base	none	+0.00
Cheetah-Run	lrelu	base	none	+10.15
Cheetah-Run	prelu	base	none	+11.33
Cheetah-Run	frelu	0.1	none	+20.59
Cheetah-Run	flrelu	0.1	none	+27.43
Cheetah-Run	fprelu	0.1	none	+16.06
Finger-Spin	relu	base	none	+0.00
Finger-Spin	lrelu	base	none	-1.70
Finger-Spin	prelu	base	none	-3.25
Finger-Spin	frelu	0.3	none	+6.70
Finger-Spin	flrelu	0.4	none	+1.35
Finger-Spin	fprelu	0.5	none	-7.10
HalfCheetah-v4	relu	base	none	+0.00
HalfCheetah-v4	lrelu	base	none	+8.17
HalfCheetah-v4	prelu	base	none	+2.87
HalfCheetah-v4	frelu	0.2	none	+28.54
HalfCheetah-v4	flrelu	0.4	none	+25.21
HalfCheetah-v4	fprelu	0.1	none	+16.28
Hopper-v4	relu	base	none	+0.00
Hopper-v4	lrelu	base	none	-7.08
Hopper-v4	prelu	base	none	+18.87
Hopper-v4	frelu	0.3	none	+8.44
Hopper-v4	flrelu	0.2	none	+34.44
Hopper-v4	fprelu	0.1	none	+43.51
Humanoid-v4	relu	base	none	+0.00
Humanoid-v4	lrelu	base	none	-5.71
Humanoid-v4	prelu	base	none	+4.88
Humanoid-v4	frelu	0.2	none	-9.93
Humanoid-v4	flrelu	0.4	none	+1.15
Humanoid-v4	fprelu	0.2	none	-2.78
Walker-Walk	relu	base	none	+0.00
Walker-Walk	lrelu	base	none	+20.65
Walker-Walk	prelu	base	none	+9.99
Walker-Walk	frelu	0.1	none	+3.36
Walker-Walk	flrelu	0.1	none	+7.09
Walker-Walk	fprelu	0.2	none	-4.43

Table 23: Best configuration per activation per task (SAC, 2-layer), mean AUC vs ReLU baseline (placement r when available).

Task	Activation	Best α	Best placement	$\Delta\%$
Ant-v4	relu	base	r	+0.00
Ant-v4	lrelu	base	all-critic	+37.54
Ant-v4	prelu	base	first-actor	+42.33
Ant-v4	frelu	0.2	first-actor	+43.08
Ant-v4	flrelu	0.4	first-actor	+34.09
Ant-v4	fprelu	0.2	all-critic	+48.72
Cartpole-Swingup	relu	base	r	+0.00
Cartpole-Swingup	lrelu	base	all-critic	+0.96
Cartpole-Swingup	prelu	base	all-actor	+1.61
Cartpole-Swingup	frelu	0.5	first-actor	+3.58
Cartpole-Swingup	flrelu	0.3	all-actor	+4.08
Cartpole-Swingup	fprelu	0.5	all-both	+2.78
Cheetah-Run	relu	base	r	+0.00
Cheetah-Run	lrelu	base	all-both	+14.04
Cheetah-Run	prelu	base	all-actor	+19.43
Cheetah-Run	frelu	0.1	first-actor	+21.59
Cheetah-Run	flrelu	0.1	all-actor	+20.86
Cheetah-Run	fprelu	0.3	all-actor	+28.34
Finger-Spin	relu	base	r	+0.00
Finger-Spin	lrelu	base	all-critic	+11.61
Finger-Spin	prelu	base	all-actor	+14.44
Finger-Spin	frelu	0.5	all-actor	+11.85
Finger-Spin	flrelu	0.3	all-actor	+15.97
Finger-Spin	fprelu	0.3	first-actor	+18.49
HalfCheetah-v4	relu	base	r	+0.00
HalfCheetah-v4	lrelu	base	all-both	+13.58
HalfCheetah-v4	prelu	base	all-critic	+15.16
HalfCheetah-v4	frelu	0.2	first-actor	+14.33
HalfCheetah-v4	flrelu	0.3	first-both	+11.38
HalfCheetah-v4	fprelu	0.3	first-both	+18.15
Hopper-v4	relu	base	r	+0.00
Hopper-v4	lrelu	base	all-critic	+6.45
Hopper-v4	prelu	base	first-actor	+9.95
Hopper-v4	frelu	0.2	all-both	+22.06
Hopper-v4	flrelu	0.3	first-both	+15.13
Hopper-v4	fprelu	0.2	all-both	+23.65
Humanoid-v4	relu	base	r	+0.00
Humanoid-v4	lrelu	base	first-both	-3.55
Humanoid-v4	prelu	base	all-both	+3.48
Humanoid-v4	frelu	0.5	first-both	+7.88
Humanoid-v4	flrelu	0.5	first-both	+9.75
Humanoid-v4	fprelu	0.3	all-both	+10.96
Walker-Walk	relu	base	r	+0.00
Walker-Walk	lrelu	base	all-both	+5.32
Walker-Walk	prelu	base	all-actor	+7.28
Walker-Walk	frelu	0.1	first-actor	+7.76
Walker-Walk	flrelu	0.1	all-actor	+7.49
Walker-Walk	fprelu	0.5	first-actor	+10.89