COMPUTE EFFICIENT DEEP REINFORCEMENT LEARNING VIA NORMALIZED GLU BLOCK

Anonymous authors
Paper under double-blind review

ABSTRACT

Massively parallel simulators in robotics and game environments have shifted reinforcement learning (RL) into a wall-clock-constrained regime, where agents must achieve strong performance within limited compute budgets. This creates a fundamental trade-off in network design: larger models provide greater expressive capacity but incur higher per-step computational cost, whereas smaller models train faster but often saturate early. Addressing this challenge requires architectures that achieve higher representational capacity per parameter. We revisit Gated Linear Units (GLUs), which have shown strong parameter efficiency in vision and language models, and investigate their effectiveness in RL. We find that naively applying GLUs to RL leads to unstable training because their multiplicative interactions amplify activation variance, causing feature norm explosion. To overcome this limitation, we propose N-GLU, a simple architectural modification that combines GLUs with pre-activation layer normalization. N-GLU stabilizes feature magnitudes while preserving the expressive benefits of multiplicative gating. We provide both theoretical motivation and empirical evidence that N-GLU produces higher-rank feature representations than standard ReLU blocks, thereby alleviating plasticity loss, a common failure mode in RL. Across a diverse set of benchmarks, including Atari 57, HumanoidBench, and MuJoCo Playground, replacing ReLU blocks with N-GLU in PQN and FastTD3 consistently improves compute efficiency over standard MLP-based architectures.

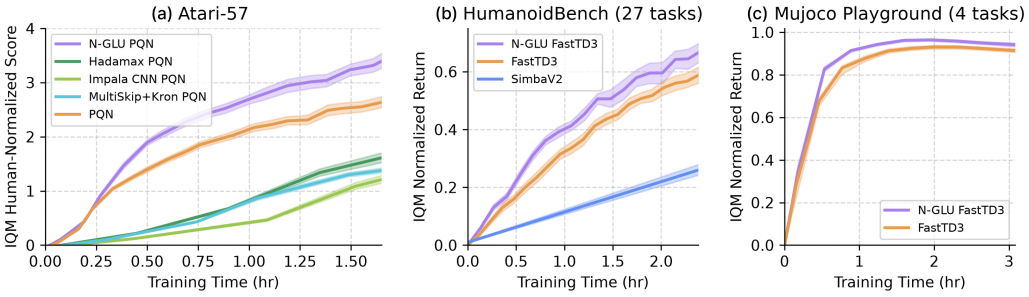

Figure 1: **Compute vs RL Performance.** N-GLU is a simple yet effective architectural block that improves compute efficiency by directly replacing existing network blocks. Across Atari-57 with PQN and HumanoidBench and MuJoCo Playground with FastTD3, N-GLU consistently outperforms strong state-of-the-art baselines. All agents are trained for the same wall-clock time required by the corresponding baseline to complete its standard training budget: 200M frames for PQN and 100k steps for FastTD3.

1 INTRODUCTION

Reinforcement learning (RL) is increasingly deployed in domains where agents are trained in simulation and transferred to real-world applications, including robotics (Levine et al., 2016; Hwangbo et al., 2019; Kaufmann et al., 2023) and complex video games (Vinyals et al., 2019; Berner et al., 2019). Recent advances in massively parallel simulators enable RL agents to collect data at unprecedented scale and speed (Makoviychuk et al., 2021), shifting modern RL into a wall-clock-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

constrained regime where the primary challenge is no longer data collection, but achieving strong performance within limited compute budgets. Accordingly, developing compute-efficient RL algorithms has become a central challenge in modern RL.

Compute efficiency in RL is governed by a fundamental trade-off between representational capacity and computational cost. Larger networks often improve sample efficiency by learning richer and more expressive representations (Hansen et al., 2023; Lee et al., 2025; Castanyer et al., 2025; Ma et al., 2025; Wang et al., 2025; Palenicek et al., 2025a; Kooi et al., 2025), where expressivity is commonly associated with higher feature rank (Kumar et al., 2020; Lyle et al., 2022). However, these gains come at the expense of increased per-step computation, reducing training throughput. In contrast, smaller networks enable faster environment interaction and more updates per unit time (Seo et al., 2025; Li et al., 2023; Petrenko et al., 2020; Espeholt et al., 2020), but their limited capacity often leads to early performance saturation (Schwarzer et al., 2023; Nauman et al., 2024a). This trade-off motivates the search for architectures that provide higher expressive capacity per parameter, allowing small networks to remain computationally efficient while retaining strong asymptotic performance.

A promising architectural component for improving parameter efficiency is the Gated Linear Unit (GLU). GLUs introduce multiplicative interactions that increase representational expressivity with relatively low parameter and computational overhead, and have become a core building block in modern vision and language models (Dauphin et al., 2017; Shazeer, 2020; Liu et al., 2021; Touvron et al., 2023; Chowdhery et al., 2023; Shi, 2024; Ma et al., 2024). Despite their success in supervised learning, GLUs remain largely unexplored in RL. In our preliminary experiments, naively replacing ReLU blocks with GLUs often resulted in unstable or collapsed training. We find that the multiplicative gating mechanism amplifies activation variance, causing uncontrolled feature norm growth during RL training. Since this phenomenon is rarely observed in supervised learning, we hypothesize that the instability arises from the inherently non-stationary optimization dynamics of RL (Kumar et al., 2021).

To address this issue, we propose **N-GLU**, a simple architectural modification that combines GLUs with pre-activation layer normalization. N-GLU stabilizes feature magnitudes while preserving the expressive advantages of multiplicative gating. We provide both theoretical motivation and empirical evidence that N-GLU produces higher-rank feature representations than standard ReLU blocks, thereby mitigating plasticity loss, a common failure mode in RL.

To evaluate its generality, we replace the ReLU-based linear and convolutional blocks in standard RL architectures with N-GLU blocks while leaving all other components unchanged. We study both discrete and continuous control settings using PQN (Gallici et al., 2024) on Atari (Bellemare et al., 2013), and FastTD3 (Seo et al., 2025) on HumanoidBench (Sferrazza et al., 2024) and MuJoCo Playground (Zakka et al., 2025). Across 57 Atari games, 27 HumanoidBench tasks, and 4 MuJoCo Playground tasks, N-GLU consistently improves compute efficiency over standard MLP-based architectures. In particular, N-GLU outperforms vanilla PQN, Hadamax PQN (Kooi et al., 2025), and FastTD3, while also surpassing scaling-oriented approaches such as Multiskip+Kron PQN (Castanyer et al., 2025) and SimbaV2 (Lee et al., 2025). These results demonstrate that carefully stabilizing multiplicative architectures can substantially improve representation quality and training efficiency in RL, highlighting the importance of architectural design for compute-efficient reinforcement learning.

2 Related Work

2.1 Compute Efficiency in Deep RL

Prior work on compute-efficient RL either improves sample efficiency, typically by increasing network capacity (Hansen et al., 2023; Lee et al., 2025; Castanyer et al., 2025; Ma et al., 2025; Wang et al., 2025; Palenicek et al., 2025a; Schwarzer et al., 2023; Obando-Ceron et al., 2024; Kooi et al., 2025; Fu et al., 2025), or improves sample throughput, typically by combining small networks with distributed and parallel environments and normalization methods (Horgan et al., 2018; Espeholt et al., 2018; 2020; Petrenko et al., 2020; Weng et al., 2022; Seo et al., 2025; Li et al., 2023; Gallici et al., 2024; Palenicek et al., 2025b; Kim et al., 2026). This creates a trade-off: larger models learn more from each transition but slow down each update, whereas smaller models iterate faster but

are limited by poor expressive power (Schwarzer et al., 2023; Nauman et al., 2024a). Our N-GLU block addresses this trade-off by combining a GLU-based architecture for expressivity with layer normalization for stability, while preserving the efficiency of small networks.

2.2 Expressivity of Neural Networks in Deep RL

Expressivity refers to a network's ability to preserve diverse and task-relevant distinctions in its hidden features, which determines the complexity of functions it can represent (Mnih et al., 2015; Raghu et al., 2017). Recent scaling results suggest that limited capacity is a major bottleneck for sample-efficient RL in both discrete (Schwarzer et al., 2023; Castanyer et al., 2025) and continuous control (Hansen et al., 2023; Nauman et al., 2024b; Lee et al., 2025; Ma et al., 2025; Wang et al., 2025; Palenicek et al., 2025a; Nauman et al., 2025). Expressivity is further limited by loss of plasticity under the non-stationarity of RL (Nikishin et al., 2022; Lyle et al., 2023; D'Oro et al., 2022; Dohare et al., 2024; Shin & Yun, 2026; Nikishin et al., 2023; Juliani & Ash, 2024; Klein et al., 2024; Ma et al., 2023), which has been addressed through parameter reset (Sokar et al., 2023; Farias & Jozefiak, 2024; Galashov et al., 2024; Ash & Adams, 2020; Lee et al., 2024; Shin et al., 2024; Han et al., 2026; Zhou et al., 2025; Liu et al., 2024; 2025), activation functions (Lewandowski et al., 2024b; Park et al., 2025; Lillo & Cheney, 2025; Obando-Ceron et al., 2025; Delfosse et al., 2021), optimizers (Elsayed & Mahmood, 2024; Tang et al., 2025; Muppidi et al., 2024), and regularization (Lewandowski et al., 2024a; Elsayed et al., 2024; Chung et al., 2024; Kumar et al., 2023; Wu et al., 2026; Saheb Pasand et al., 2026). Following prior work, we measure expressivity by the effective rank of the penultimate feature matrix (Jing et al., 2021; Balestriero & LeCun, 2022; Garrido et al., 2023), a common diagnostic of lost expressivity in RL (Kumar et al., 2020; 2021; Lyle et al., 2023).

2.3 Gated Linear Unit

The Gated Linear Unit (GLU) (Dauphin et al., 2017) computes $\mathrm{GLU}(\mathbf{x}) = (\mathbf{W}_v\mathbf{x}+\mathbf{b}_v)\odot f(\mathbf{W}_g\mathbf{x}+\mathbf{b}_g)$, where $f(\cdot)$ is the sigmoid function and $\odot$ denotes element-wise multiplication, allowing one feature stream to selectively pass or suppress another. Variants such as SwiGLU, which we adopt, replace the sigmoid with other nonlinearities (Shazeer, 2020) and are now standard in Transformer-based language models (Chowdhery et al., 2023; Touvron et al., 2023). GLUs are parameter and compute efficient in both NLP and CV (Dauphin et al., 2017; Liu et al., 2021), and recent vision architectures achieve stronger performance with substantially fewer parameters using GLU-based designs (Shi, 2024; Ma et al., 2024). In RL, however, the few works exploring multiplicative gating mainly focus on plasticity by reducing dormant neurons (Kooi et al., 2024; 2025), whereas we focus on the expressive capacity such gating provides.

3 Normalized GLU Block

In this section, we introduce the Normalized GLU Block, a small yet expressive architecture that enables computationally efficient training of RL. We first describe the detailed structure of the N-GLU block in Section 3.1. We then theoretically demonstrate that N-GLU can improve expressive power in Section 3.2.

3.1 Network Architecture

In this section, we introduce N-GLU Block, an architecture designed to endow compact networks with strong expressivity. N-GLU Block combines a GLU-based design that enhances representational capacity with pre-activation layer normalization, which stabilizes feature representations and helps preserve expressivity.

An illustration of the N-GLU Block is shown in Figure 2. Given an input, the N-GLU Block splits it into two branches. In the first branch, the input is processed by a linear or convolutional layer, followed by layer normalization and a Swish activation. In the second branch, the input is processed by a

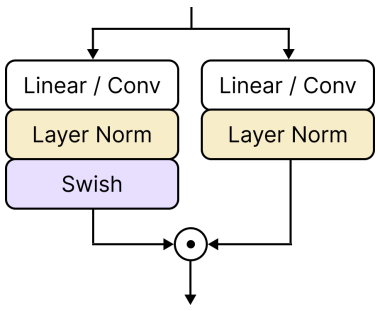

Figure 2: **N-GLU block**

linear or convolutional layer followed by layer normalization.
The outputs of the two branches are then combined through element-wise multiplication. Unlike a standard GLU, which multiplies unnormalized features, N-GLU Block normalizes both branches before the multiplicative interaction. This design is crucial for preventing feature-norm explosion and improving training stability.

3.2 Theoretical Analysis

In this section, we provide a theoretical motivation for why N-GLU can improve expressive power. We compare a linear layer and a GLU layer under a matched parameter budget and show that the multiplicative interaction in GLU yields a substantially larger upper bound on feature rank.

To make our theoretical claim, we first need a measure that quantifies expressivity. Following previous works that used feature rank as a proxy for the expressivity (Jing et al., 2021; Balestriero & LeCun, 2022; Garrido et al., 2023), we quantify expressive power through the rank of the feature matrix. Let $H \in \mathbb{R}^{m\times n}$ denote the representation of n samples, where m is the feature dimension. A higher feature rank $\mathrm{rank}(H)$ indicates that the representation spans a richer subspace and captures more diverse patterns, whereas a lower rank suggests redundancy or representation collapse. Prior work has used feature rank and related effective-rank measures as practical proxies for representation quality and expressive power, since higher-rank features preserve more diverse directions of variation and are less prone to collapse (Kumar et al., 2020; Lyle et al., 2022; Balestriero & LeCun, 2022; Jing et al., 2021; Garrido et al., 2023). Because exact rank can be sensitive to very small singular values, for nonzero H we also consider the stable feature rank measure of Kumar et al. (2020):

$$\mathrm{srank}_\delta(H) = \min\left\{ k : \frac{\sum_{i=1}^{k}\sigma_i(H)}{\sum_{i=1}^{\min\{n,m\}}\sigma_i(H)} \geq 1-\delta \right\}, \tag{1}$$

where $\sigma_i(H)$ denotes the singular values of H in decreasing order. This measure captures how many dominant singular directions are needed to explain most of the feature spectrum.

For simplicity, we consider a single GLU layer and a single linear layer. We define the GLU layer as

$$\mathrm{GLU}(Z) = (W_v Z) \odot (W_g Z), \tag{2}$$

where $W_v, W_g \in \mathbb{R}^{d\times d}$, and Z denotes an arbitrary input matrix. GLU layer gives a total of $2d^2$ parameters. We omit bias terms, and the activation function for simplicity, since they are not needed for the upper-bound argument.

Since the GLU block has two weight matrices, it doubles the number of parameters compared to standard linear layer which has d^2 parameters. Thus, for fair comparison, we enlarge input and output dimension of linear layer by $\sqrt{2}$ to match the number of parameters. Accordingly, we define the linear layer as

$$\mathrm{Linear}(Z) = WZ, \tag{3}$$

where $W \in \mathbb{R}^{\sqrt{2}d\times\sqrt{2}d}$, and Z denotes an arbitrary input matrix. The linear layer has approximately $2d^2$ parameters, which is equal to that of GLU.

For fair comparison, we use the same input matrix for both GLU and linear layer. Let $X \in \mathbb{R}^{d_0\times n}$ be an input matrix with intrinsic rank

$$r_x := \mathrm{rank}(X) \leq d. \tag{4}$$

However, since GLU and linear layer have different input dimensions, we introduce fixed rank-preserving linear maps $P_{\mathrm{glu}} \in \mathbb{R}^{d\times d_0}$ and $P_{\mathrm{linear}} \in \mathbb{R}^{\sqrt{2}d\times d_0}$, and define

$$X_{\mathrm{glu}} = P_{\mathrm{glu}}X, \qquad X_{\mathrm{linear}} = P_{\mathrm{linear}}X, \tag{5}$$

with

$$\mathrm{rank}(X_{\mathrm{glu}}) = \mathrm{rank}(X_{\mathrm{linear}}) = r_x. \tag{6}$$

These auxiliary maps are introduced only to align dimensions and are not counted in the parameter budget.

Applying the layers to the input matrices yields the feature matrices

$$H_{\text{glu}} = \text{GLU}(X_{\text{glu}}), \qquad H_{\text{linear}} = \text{Linear}(X_{\text{linear}}). \tag{7}$$

The key difference between the two layers is that the GLU contains an element-wise product of two linear projections, whereas the linear layer applies a single linear projection. A nice property of the element-wise product (i.e. Hadamard product) is

$$\text{rank}(A \odot B) \leq \text{rank}(A)\,\text{rank}(B), \tag{8}$$

which allows GLU to exhibit a stronger rank upper bound than an ordinary linear layer. By leveraging this property, we introduce Proposition 1 and Corollary 1.

Proposition 1 (Feature-rank upper bounds under a shared input and matched parameter budget)**.** *Under the setup above,*

$$\text{rank}(H_{\text{glu}}) \leq \min\{n, d, r_x^2\}, \qquad \text{rank}(H_{\text{linear}}) \leq \min\{n, r_x\}. \tag{9}$$

Corollary 1 (Upper bounds on srank_δ)**.** *For every nonzero H,* $\text{srank}_\delta(H) \leq \text{rank}(H)$*. Hence,*

$$\text{srank}_\delta(H_{\text{glu}}) \leq \min\{n, d, r_x^2\}, \qquad \text{srank}_\delta(H_{\text{linear}}) \leq \min\{n, r_x\}. \tag{10}$$

Proposition 1 shows an upper bound of standard rank, while Corollary 1 extends it to the stable feature rank. Proof can be found in Appendix A. Since our interest is stable feature rank, we continue our analysis based on Corollary 1.

Three quantities govern the upper bound of stable feature rank: n, d, and r_x. Among these, the batch size n imposes a numerical upper bound on the rank of the feature matrix, but it is a sampling constraint rather than a fundamental change to the intrinsic representation. The underlying function defined by the model's weights and architecture remains the same, independent of how many data points pass through it at once. Increasing the batch size therefore reveals more of the existing latent structure without altering the mapping function itself. Accordingly, we assume $n > d$, so that stable feature rank is measured with a sufficient number of samples and is not constrained by n. Under this assumption, Corollary 1 reduces to

$$\text{srank}_\delta(H_{\text{glu}}) \leq \min\{d, r_x^2\}, \qquad \text{srank}_\delta(H_{\text{linear}}) \leq r_x. \tag{11}$$

In practice, the stable feature rank is also typically below its maximum value, $r_x < d$, even under rank-preserving interventions (Kumar et al., 2020; Lyle et al., 2022; Lee et al., 2023). For any nontrivial rank $r_x > 1$, this implies

$$r_x < \min\{d, r_x^2\}. \tag{12}$$

Therefore, the GLU layer has a strictly larger stable feature rank upper bound than the linear layer:

$$\text{srank}_\delta(H_{\text{glu}}) \leq \min\{d, r_x^2\}, \qquad \text{srank}_\delta(H_{\text{linear}}) \leq r_x < \min\{d, r_x^2\}. \tag{13}$$

Although these results concern upper bounds rather than the rank attained in practice, they identify the mechanism through which N-GLU can improve expressive power. In Section 4.3, we empirically show that N-GLU indeed increases stable feature rank, as illustrated in Figure 6. Furthermore, since we use LayerNorm in practice, in Appendix B we show that this theoretical results still holds in practical setting that uses LayerNorm.

4 Experiments

To assess the effectiveness and generalizability of the N-GLU block, we conduct extensive evaluations across Atari-57 (Bellemare et al., 2013), HumanoidBench (Sferrazza et al., 2024), and MuJoCo Playground (Zakka et al., 2025), as shown in Figure 3. In each benchmark, we keep the learning algorithm and all other components unchanged, modifying only the network architecture by replacing the ReLU-activated linear or convolutional layers in standard architectures with N-GLU blocks, except for the output layer. Specifically, we use PQN (Gallici et al., 2024) for Atari-57 and FastTD3 (Seo et al., 2025) for HumanoidBench and MuJoCo Playground.

We organize our evaluation as follows. In Section 4.1, we evaluate N-GLU on discrete control tasks using PQN. In Section 4.2, we evaluate N-GLU on continuous control tasks using FastTD3. Finally, in Section 4.3, we conduct ablation studies showing that N-GLU improves feature rank in practice and layer normalization is essential for stabilizing training. We provide a code snippet and detailed settings in Appendix C, D, and E.

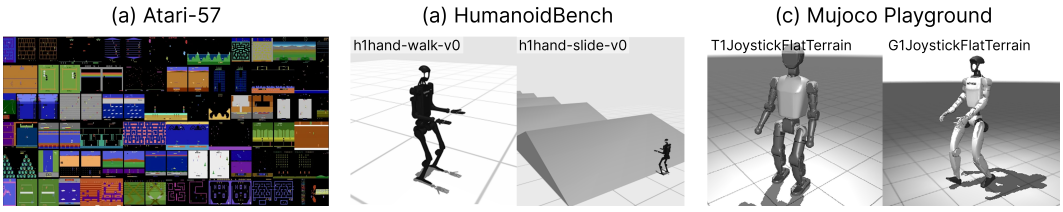

Figure 3: **Overview of benchmark environments.** **(a)** Atari-57 (Bellemare et al., 2013). **(b)** HumanoidBench (Sferrazza et al., 2024). **(c)** Mujoco Playground (Zakka et al., 2025).

4.1 Discrete Control

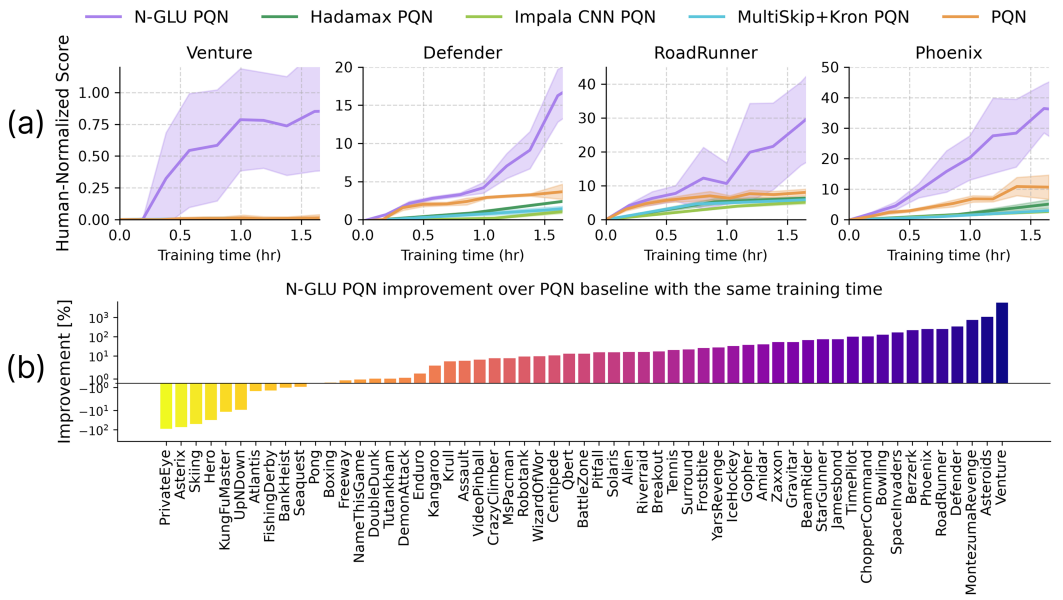

Figure 4: **Atari-57 results.** **(a)** N-GLU PQN exhibits strong compute efficiency across four representative Atari environments. **(b)** Across Atari-57, N-GLU PQN outperforms the base PQN in 84.2% of environments, achieving an 13.5% median relative improvement. All agents are trained for the same wall-clock time that the base PQN requires to complete training for the standard 200 million frames.

Setup. We evaluate N-GLU on Atari-57 (Bellemare et al., 2013), with PQN (Gallici et al., 2024) as the base algorithm and modify only the network architecture. Specifically, we replace the ReLU-LayerNorm-linear and convolutional blocks in the standard PQN network with N-GLU blocks, except for the final layer, while keeping all other settings unchanged. As a result, the baseline PQN network and N-GLU have the same number of hidden layers and hidden dimension. However, since each N-GLU block uses two linear projections, N-GLU has approximately twice as many parameters as the baseline network. We train each agent for the same wall-clock time required by the baseline PQN to complete training for the standard 200 million frames, which corresponds to 1 hour and 39 minutes on our hardware setup. Our implementation is based on PyTorch (Paszke et al., 2019).

Baselines. We compare N-GLU against several PQN-based baselines on Atari-57. First, we use the base PQN agent with the standard network architecture as our primary baseline. In addition, we include Hadamax PQN, a strong baseline that leverages feature encoding (Kooi et al., 2025), and Impala CNN PQN, which combines PQN with the Impala CNN architecture (Espeholt et al., 2018). Both Hadamax PQN and Impala CNN PQN exploit enhanced feature representations and have demonstrated strong performance on Atari-57. We also compare against MultiSkip+Kron PQN,

a scaling-based approach that uses a large multi-skip residual network together with the Kron optimizer (Castanyer et al., 2025). MultiSkip+Kron PQN leverages the expressive power of a large network to achieve superior performance. Although N-GLU (3.4M) has roughly twice as many parameters as the baseline PQN network (1.7M), it remains smaller than the other architectural baselines, including MultiSkip (4.6M), Impala CNN (5.8M), and Hadamax (4.1M).

Results. The results of full Atari-57 benchmark is shown in Figure 1 (a). Feature-encoding-based methods that rely on stronger but larger encoders, such as Hadamax PQN and Impala CNN PQN, exhibit poor compute efficiency. Similarly, the scaling-based approach, MultiSkip+Kron PQN, also shows limited compute efficiency, achieving only about half the performance of the base PQN by the end of training. In contrast, N-GLU PQN outperforms all baselines with strong compute efficiency, simply by replacing the standard blocks in the PQN network with N-GLU blocks.

This strong performance becomes even more pronounced when examining individual environments. As shown in Figure 4(a), N-GLU PQN demonstrates high compute efficiency across four Atari environments: Venture, Defender, RoadRunner, and Phoenix. In particular, on Venture, while all other baselines struggle to learn, N-GLU PQN reaches human-level performance, corresponding to a human-normalized score of 1, within one hour of training.

We further visualize the improvement of N-GLU PQN over the base PQN under the same training time in Figure 4(b). Across the Atari-57 benchmark, N-GLU PQN outperforms the base PQN in 84.2% of environments, with an 13.5% median relative improvement.

4.2 Continuous Control

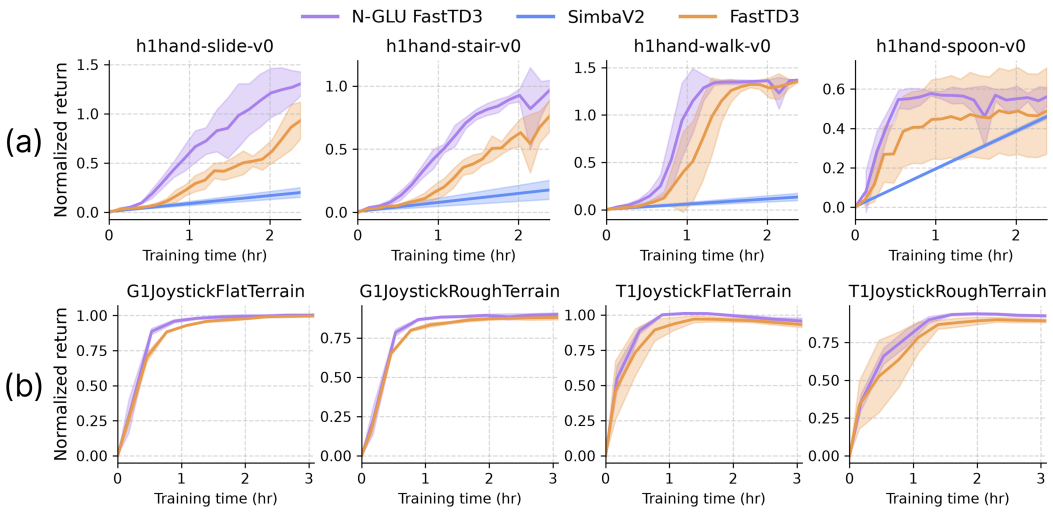

Figure 5: **FastTD3 results.** N-GLU FastTD3 demonstrates strong compute efficiency across four representative HumanoidBench tasks **(a)** and MuJoCo Playground tasks **(b)**.

Setup. We evaluate N-GLU across 27 HumanoidBench tasks (Sferrazza et al., 2024) and 4 MuJoCo Playground tasks (Zakka et al., 2025), using FastTD3 (Seo et al., 2025) as the base algorithm. These benchmarks cover a range of challenging high-degree-of-freedom control tasks. We replace the ReLU-linear blocks in the standard FastTD3 network with N-GLU blocks, except for the final layer. This modification is applied only to the critic, since the critic is the main component through which increased expressive power drives performance improvements (Nauman et al., 2024a). Each agent is trained for the same wall-clock time required by the baseline FastTD3 to complete the standard 100k training steps, corresponding to 2 hours and 22 minutes on HumanoidBench and 3 hours and 5 minutes on MuJoCo Playground in our hardware setup. Our implementation is based on the official FastTD3 codebase (Seo et al., 2025).

Baselines. We compare our approach against the base FastTD3 agent with the standard network architecture. In addition, we include SimbaV2 (Lee et al., 2025), a strong scaling-based baseline that has demonstrated strong performance across a wide range of continuous control tasks.

Results. The HumanoidBench results are shown in Figure 1(b), and the MuJoCo Playground results are shown in Figure 1(c). As shown in the figure, N-GLU FastTD3 achieves strong compute efficiency and outperforms all baselines on both HumanoidBench and MuJoCo Playground. In contrast, SimbaV2, which improves expressive power by scaling the network, exhibits poor compute efficiency. We further report results on four representative HumanoidBench environments in Figure 5(a), and per-environment results on MuJoCo Playground in Figure 5(b). Across all environments, N-GLU FastTD3 consistently surpasses the base FastTD3.

4.3 Ablation Study

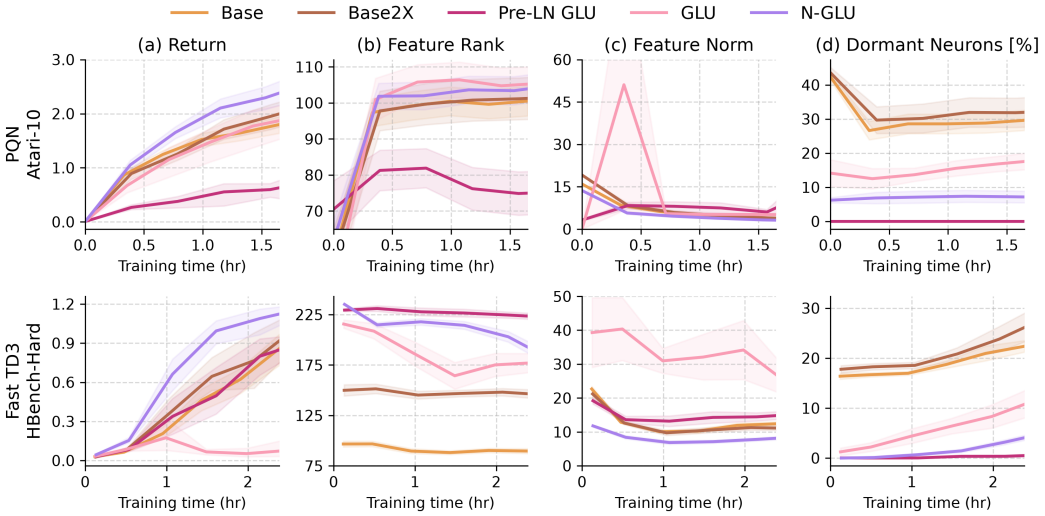

Figure 6: **Ablation study on N-GLU.** GLU suffers from feature-norm explosion, which leads to poor performance. N-GLU prevents this with pre-activation layer normalization, enabling stable learning and strong expressive power.

In this section, we provide empirical evidence that N-GLU improves feature rank in practice and that layer normalization is essential for stable training of GLU-based architectures.

Setup. We conduct our analysis on both discrete and continuous control tasks. For discrete control, we use the Atari-10 benchmark, a representative subset of 57 Atari games (Aitchison et al., 2023). For continuous control, we consider HBench-Hard, a challenging subset defined for this analysis, which consists of five environments: `h1hand-walk-v0`, `h1hand-slide-v0`, `h1hand-stair-v0`, `h1hand-balance_hard-v0`, and `h1hand-sit_hard-v0`.

To analyze both representational expressivity and learning dynamics, we evaluate multiple metrics computed from the penultimate-layer features. As our main metric, we use feature rank, defined as the stable rank in Eq. 1, to measure the expressivity of learned representations. We also report the feature norm to examine the training instability observed when GLU is naively used in RL. Large feature norms directly inflate both the magnitude and the variance of gradients, and once the gradient variance dominates the gradient signal, the signal to noise ratio of parameter updates collapses, causing learning to stall or diverge Faghri et al. (2020); Liu et al. (2020); McCandlish et al. (2018). In addition, we measure the proportion of dormant neurons (Sokar et al., 2023), which are neurons with consistently low activation. A high proportion of dormant neurons indicates a loss of plasticity and, consequently, a loss of the expressivity of learned representations. Dormant neurons are computed over the features of the entire network.

Baselines. We compare five architectures: Base, Base2X, Pre-LN GLU, GLU, and N-GLU. Since N-GLU has twice as many parameters as Base, we include Base2X, which widens the standard network to match the parameter budget of the GLU-based architectures. This comparison allows us to examine whether the improvement of N-GLU comes merely from an increased number of parameters. We also compare GLU and N-GLU to analyze the feature-norm explosion that can occur when GLU is naively applied to RL, and to verify whether layer normalization can prevent this instability. Here, N-GLU combines GLU with pre-activation layer normalization. We additionally include Pre-LN GLU, as placing LayerNorm before each layer (Pre-LN) is the standard configuration in Transformer architectures (Xiong et al., 2020).

Results. The results in Figure 6 support the main implications of our theoretical analysis. GLU surpasses Base2X in feature rank under the same parameter budget, suggesting that multiplicative gating improves expressive capacity beyond merely increasing network width. In addition, GLU improves the plasticity of the network by reducing dormant neurons. The improvement in feature rank is more prominent on HBench-Hard, while it is modest on Atari-10.

However, GLU exhibits poor optimization behavior. Although it achieves high feature rank, it also produces excessively large feature norms and unstable learning dynamics, ultimately leading to poor performance. This learning instability can be observed on both Atari-10 and HBench-Hard. While Pre-LN GLU attempts to mitigate this instability, its feature norm is still larger than those of the Base and Base2X models, and on Atari-10 in particular, adding Pre-LN leads to poor performance. This indicates that adding Pre-LN is not sufficient to eliminate the problem. Regarding this learning instability of GLU, we provide additional analysis on synthetic non-stationary setting in Appendix F.

N-GLU resolves this issue by introducing layer normalization before the multiplicative activation of GLU. As a result, it preserves the high feature rank induced by GLU while substantially reducing feature-norm explosion, allowing the model to combine strong representational capacity with stable optimization.

Taken together, these results provide empirical evidence that N-GLU enhances representational expressivity in practice. Under a matched parameter budget, its higher feature rank than Base2X indicates that the gain comes from multiplicative gating rather than increased parameters, while pre-activation layer normalization stabilizes GLU by preventing feature-norm explosion. Thus, N-GLU offers an effective mechanism for achieving higher expressive capacity per parameter in deep RL.

5 Conclusion

In this work, we proposed N-GLU, a simple yet effective architectural block for compute-efficient reinforcement learning. Motivated by the trade-off between sample efficiency and sample throughput, N-GLU aims to improve the expressive power of small networks without relying on costly network scaling. By combining GLU with pre-activation layer normalization, N-GLU enhances representation capacity while maintaining stable optimization. We provided both theoretical and empirical evidence that N-GLU improves expressive capacity. Our theoretical analysis shows that multiplicative gating can increase the upper bound of feature rank under a matched parameter budget, while our ablation study confirms that N-GLU achieves higher feature rank in practice.

Across extensive experiments on discrete and continuous control benchmarks, including Atari-57 with PQN and HumanoidBench and MuJoCo Playground with FastTD3, N-GLU consistently improves compute efficiency under the same wall-clock training budget. Despite its simplicity, N-GLU outperforms strong state-of-the-art baselines, suggesting that simply aligning existing architectural components in an appropriate manner can bring significant improvement. We believe that N-GLU provides a promising direction for building efficient and expressive RL agents, and future work may further explore its applicability to broader algorithms, larger-scale environments.

Our theoretical analysis is limited to deriving an upper bound on the feature rank. Future work may investigate corresponding lower bounds to more tightly characterize the actual rank. In addition, our work focuses on network architecture and does not address other factors affecting computational efficiency, such as parallel environments.

AI USE STATEMENT

In this work, we used generative AI tools for assist in the writing of proofs. We have not used generative AI tools for Generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, interpret results, and assist with translation, clean and reformat dataset are not applicable to this work. Additionally, we used generative AI tools for edit a research paper to improve readability. We have reviewed all AI-assisted work. We checked all paragraphs and proofs that polished by LLMs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This work proposes a network architecture for improving the compute efficiency of reinforcement learning and does not involve human subjects, personal data, or crowdsourced annotation, and therefore did not require IRB approval. All experiments are conducted on publicly available simulated benchmarks (Atari-57, HumanoidBench, and MuJoCo Playground) and the public MNIST dataset, used in accordance with their licenses, and no new dataset is released. We are not aware of any potentially harmful insights, discrimination, bias, fairness, privacy, or security concerns specific to this work, nor of any conflicts of interest or sponsorship that could have influenced the results. As a general purpose architectural component, N-GLU could in principle be applied to any downstream RL application, but it does not introduce risks beyond those of existing RL methods. By reducing the wall-clock time required for training, it may also help lower the computational and energy cost of RL research.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure that all results in this paper are reproducible. The architecture of the N-GLU block is described in Section 3.1, and a complete PyTorch implementation of both the linear and convolutional variants, including the initialization used for FastTD3, is provided in Appendix C, together with the public codebases and licenses on which our experiments are built. All hyperparameters, the number of seeds per method, and the source of baseline scores are listed in Appendix D, and the hardware, training time, number of parameters, FLOPs, and SPS of every agent are reported in Appendix E. For the theoretical results, all assumptions are stated in Section 3.2, and complete proofs, including the extension to LayerNorm, are given in Appendices A and B. The setup of the synthetic random label MNIST experiment is fully described in Appendix F. All benchmarks used in this work (Atari-57, HumanoidBench, MuJoCo Playground, and MNIST) are publicly available.

REFERENCES

Matthew Aitchison, Penny Sweetser, and Marcus Hutter. Atari-5: Distilling the arcade learning environment down to five games. In *International Conference on Machine Learning*, pp. 421–438. PMLR, 2023.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

Randall Balestriero and Yann LeCun. Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods. *Advances in Neural Information Processing Systems*, 35:26671–26685, 2022.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47: 253–279, 2013.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

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

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

Quentin Delfosse, Patrick Schramowski, Martin Mundt, Alejandro Molina, and Kristian Kersting. Adaptive rational activations to boost deep reinforcement learning. *arXiv preprint arXiv:2102.09407*, 2021.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep reinforcement learning workshop neurIPS 2022*, 2022.

Mohamed Elsayed and A Rupam Mahmood. Addressing loss of plasticity and catastrophic forgetting in continual learning. *arXiv preprint arXiv:2404.00781*, 2024.

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

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1407–1416, 2018.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. In *International Conference on Learning Representations*, 2020.

Fartash Faghri, David Duvenaud, David J Fleet, and Jimmy Ba. A study of gradient variance in deep learning. *arXiv preprint arXiv:2007.04532*, 2020.

Vivek F Farias and Adam D Jozefiak. Self-normalized resets for plasticity in continual learning. *arXiv preprint arXiv:2410.20098*, 2024.

Preston Fu, Oleh Rybkin, Zhiyuan Zhou, Michal Nauman, Pieter Abbeel, Sergey Levine, and Aviral Kumar. Compute-optimal scaling for value-based deep rl. *arXiv preprint arXiv:2508.14881*, 2025.

Alexandre Galashov, Michalis Titsias, András György, Clare Lyle, Razvan Pascanu, Yee Whye Teh, and Maneesh Sahani. Non-stationary learning of neural networks with automatic soft parameter reset. *Advances in Neural Information Processing Systems*, 37:83197–83234, 2024.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

Quentin Garrido, Randall Balestriero, Laurent Najman, and Yann Lecun. Rankme: Assessing the downstream performance of pretrained self-supervised representations by their rank. In *International conference on machine learning*, pp. 10929–10974. PMLR, 2023.

Isaac Han, Sangyeon Park, Seungwon Oh, Donghu Kim, Hojoon Lee, and Kyung-Joong Kim. Fire: Frobenius-isometry reinitialization for balancing the stability-plasticity tradeoff. *arXiv preprint arXiv:2602.08040*, 2026.

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

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science robotics*, 4(26):eaau5872, 2019.

Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. *arXiv preprint arXiv:2110.09348*, 2021.

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

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Donghu Kim, Youngdo Lee, Minho Park, Kinam Kim, I Nahendra, Takuma Seno, Sehee Min, Daniel Palenicek, Florian Vogt, Danica Kragic, et al. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

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

Jacob E Kooi, Mark Hoogendoorn, and Vincent François-Lavet. Hadamard representations: Augmenting hyperbolic tangents in rl. *arXiv preprint arXiv:2406.09079*, 2024.

Jacob E Kooi, Zhao Yang, and Vincent François-Lavet. Hadamax encoding: Elevating performance in model-free atari. *arXiv preprint arXiv:2505.15345*, 2025.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *arXiv preprint arXiv:2010.14498*, 2020.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. *arXiv preprint arXiv:2112.04716*, 2021.

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

Hojoon Lee, Koanho Lee, Dongyoon Hwang, Hyunho Lee, Byungkun Lee, and Jaegul Choo. On the importance of feature decorrelation for unsupervised representation learning in reinforcement learning. In *International Conference on Machine Learning*, pp. 18988–19009. PMLR, 2023.

Hojoon Lee, Hyeonseo Cho, Hyunseung Kim, Donghu Kim, Dugki Min, Jaegul Choo, and Clare Lyle. Slow and steady wins the race: Maintaining plasticity with hare and tortoise networks. *arXiv preprint arXiv:2406.02596*, 2024.

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

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Alex Lewandowski, Michał Bortkiewicz, Saurabh Kumar, András György, Dale Schuurmans, Mateusz Ostaszewski, and Marlos C Machado. Learning continually by spectral regularization. *arXiv preprint arXiv:2406.06811*, 2024a.

Alex Lewandowski, Dale Schuurmans, and Marlos C Machado. Plastic learning with deep fourier features. *arXiv preprint arXiv:2410.20634*, 2024b.

Zechu Li, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal. Parallel q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *International Conference on Machine Learning*, pp. 19440–19459. PMLR, 2023.

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

Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. Pay attention to mlps. *Advances in neural information processing systems*, 34:9204–9215, 2021.

Jiashun Liu, Johan Obando-Ceron, Aaron Courville, and Ling Pan. Neuroplastic expansion in deep reinforcement learning. *arXiv preprint arXiv:2410.07994*, 2024.

Jiashun Liu, Zihao Wu, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Ling Pan. Measure gradients, not activations! enhancing neuronal activity in deep reinforcement learning. *arXiv preprint arXiv:2505.24061*, 2025.

Jinlong Liu, Guoqing Jiang, Yunzhi Bai, Ting Chen, and Huayan Wang. Understanding why neural networks generalize well through gsnr of parameters. *arXiv preprint arXiv:2001.07384*, 2020.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. *arXiv preprint arXiv:2204.09560*, 2022.

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

Guozheng Ma, Lu Li, Sen Zhang, Zixuan Liu, Zhen Wang, Yixin Chen, Li Shen, Xueqian Wang, and Dacheng Tao. Revisiting plasticity in visual reinforcement learning: Data, modules and training stages. *arXiv preprint arXiv:2310.07418*, 2023.

Guozheng Ma, Lu Li, Zilin Wang, Li Shen, Pierre-Luc Bacon, and Dacheng Tao. Network sparsity unlocks the scaling potential of deep reinforcement learning. *arXiv preprint arXiv:2506.17204*, 2025.

Xu Ma, Xiyang Dai, Yue Bai, Yizhou Wang, and Yun Fu. Rewrite the stars. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5694–5703, 2024.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

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.

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

Michal Nauman, Michał Bortkiewicz, Piotr Miłoś, Tomasz Trzciński, Mateusz Ostaszewski, and Marek Cygan. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. *arXiv preprint arXiv:2403.00514*, 2024a.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. *Advances in neural information processing systems*, 37:113038–113071, 2024b.

Michal Nauman, Marek Cygan, Carmelo Sferrazza, Aviral Kumar, and Pieter Abbeel. Bigger, regularized, categorical: High-capacity value functions are efficient multi-task learners. *arXiv preprint arXiv:2505.23150*, 2025.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847, 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html.

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

Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep rl. *arXiv preprint arXiv:2402.08609*, 2024.

Johan Obando-Ceron, Walter Mayor, Samuel Lavoie, Scott Fujimoto, Aaron Courville, and Pablo Samuel Castro. Simplicial embeddings improve sample efficiency in actor-critic agents. *arXiv preprint arXiv:2510.13704*, 2025.

Daniel Palenicek, Florian Vogt, Joe Watson, and Jan Peters. Scaling off-policy reinforcement learning with batch and weight normalization. *arXiv preprint arXiv:2502.07523*, 2025a.

Daniel Palenicek, Florian Vogt, Joe Watson, Ingmar Posner, and Jan Peters. Xqc: Well-conditioned optimization accelerates deep reinforcement learning. *arXiv preprint arXiv:2509.25174*, 2025b.

Sangyeon Park, Isaac Han, Seungwon Oh, and KyungJoong Kim. Activation by interval-wise dropout: A simple way to prevent neural networks from plasticity loss. In *Forty-second International Conference on Machine Learning*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pp. 7652–7662. PMLR, 2020.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2847–2854, 2017. URL https://proceedings.mlr.press/v70/raghu17a.html.

Ali Saheb Pasand, Johan Obando-Ceron, Aaron Courville, Pouya Bashivan, and Pablo Samuel Castro. Stable deep reinforcement learning via isotropic gaussian representations. *arXiv e-prints*, pp. arXiv–2602, 2026.

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

Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. Fasttd3: Simple, fast, and capable reinforcement learning for humanoid control. *arXiv preprint arXiv:2505.22642*, 2025.

Carmelo Sferrazza, Dun-Ming Huang, Xingyu Lin, Youngwoon Lee, and Pieter Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Dai Shi. Transnext: Robust foveal visual perception for vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 17773–17783, 2024.

Baekrok Shin and Chulhee Yun. Implicit bias and loss of plasticity in matrix completion: Depth promotes low-rankness. *arXiv preprint arXiv:2603.04703*, 2026.

Baekrok Shin, Junsoo Oh, Hanseul Cho, and Chulhee Yun. Dash: Warm-starting neural network training in stationary settings without loss of plasticity. *Advances in Neural Information Processing Systems*, 37:43300–43340, 2024.

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

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

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

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. Envpool: A highly parallel reinforcement learning environment execution engine. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22409–22421, 2022.

Zihao Wu, Hongyao Tang, Yi Ma, Jiashun Liu, Yan Zheng, and Jianye Hao. The rank and gradient lost in non-stationarity: Sample weight decay for mitigating plasticity loss in reinforcement learning. *arXiv preprint arXiv:2604.01913*, 2026.

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

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

Huaicheng Zhou, Zifeng Zhuang, and Donglin Wang. Stay hungry, keep learning: Sustainable plasticity for deep reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025.

A PROOFS

Proof of Proposition 1. For the linear layer, by definition,

$$H_{\text{linear}} = \text{Linear}(X_{\text{linear}}) = WX_{\text{linear}}.$$

By the standard rank inequality for matrix products,

$$\text{rank}(H_{\text{linear}}) = \text{rank}(WX_{\text{linear}}) \leq \min\{\text{rank}(W), \text{rank}(X_{\text{linear}})\}.$$

Since $W \in \mathbb{R}^{\sqrt{2}d \times \sqrt{2}d}$ and

$$\text{rank}(X_{\text{linear}}) = r_x,$$

we have

$$\text{rank}(H_{\text{linear}}) \leq \min\{\sqrt{2}d, r_x\}.$$

By the setup, $r_x \leq d \leq \sqrt{2}d$, and hence

$$\min\{\sqrt{2}d, r_x\} = r_x.$$

Therefore,

$$\text{rank}(H_{\text{linear}}) \leq r_x.$$

Moreover, since

$$H_{\text{linear}} \in \mathbb{R}^{\sqrt{2}d \times n},$$

we also have

$$\text{rank}(H_{\text{linear}}) \leq n.$$

Combining the two bounds gives

$$\text{rank}(H_{\text{linear}}) \leq \min\{n, r_x\}.$$

For the GLU layer, define

$$A := W_v X_{\text{glu}}, \qquad B := W_g X_{\text{glu}}.$$

Then, by the definition of the GLU layer in the main text,

$$H_{\text{glu}} = \text{GLU}(X_{\text{glu}}) = A \odot B.$$

Using the standard rank inequality for the Hadamard product,

$$\text{rank}(A \odot B) \leq \text{rank}(A)\,\text{rank}(B),$$

we obtain

$$\text{rank}(H_{\text{glu}}) \leq \text{rank}(A)\,\text{rank}(B).$$

For the first factor,

$$\text{rank}(A) = \text{rank}(W_v X_{\text{glu}}) \leq \min\{\text{rank}(W_v), \text{rank}(X_{\text{glu}})\} \leq r_x.$$

Similarly,

$$\text{rank}(B) = \text{rank}(W_g X_{\text{glu}}) \leq \min\{\text{rank}(W_g), \text{rank}(X_{\text{glu}})\} \leq r_x.$$

Hence,

$$\text{rank}(H_{\text{glu}}) \leq r_x^2.$$

Moreover, since

$$H_{\text{glu}} \in \mathbb{R}^{d \times n},$$

we also have

$$\text{rank}(H_{\text{glu}}) \leq \min\{d, n\}.$$

Therefore,

$$\text{rank}(H_{\text{glu}}) \leq \min\{n, d, r_x^2\}.$$

This completes the proof. □

Proof of Corollary 1. Let $r = \operatorname{rank}(H)$ for a nonzero feature matrix $H \in \mathbb{R}^{m\times n}$, following the notation in the main text. Then $\sigma_i(H) = 0$ for all $i > r$. Therefore,

$$\sum_{i=1}^{r} \sigma_i(H) = \sum_{i=1}^{\min\{n,m\}} \sigma_i(H),$$

which implies

$$\frac{\sum_{i=1}^{r} \sigma_i(H)}{\sum_{i=1}^{\min\{n,m\}} \sigma_i(H)} = 1 \geq 1 - \delta.$$

By the definition of $\operatorname{srank}_\delta$, we have

$$\operatorname{srank}_\delta(H) \leq r = \operatorname{rank}(H).$$

Applying Proposition 1 to H_{glu} and H_{linear} yields

$$\operatorname{srank}_\delta(H_{\text{glu}}) \leq \min\{n, d, r_x^2\}, \qquad \operatorname{srank}_\delta(H_{\text{linear}}) \leq \min\{n, r_x\}.$$

This completes the proof. □

B Additional Theoretical Analysis with LayerNorm

In practice, we apply LayerNorm to the GLU branches for stable training, so it is natural to ask whether our theoretical result still holds in this practical setting. We show below that it does. We keep the notation of Section 3.2: $X_{\text{glu}} \in \mathbb{R}^{d\times n}$ is the input for n samples with $r_x = \text{rank}(X_{\text{glu}})$, and $W_v, W_g \in \mathbb{R}^{d\times d}$ are the value and gate projections.

LayerNorm in matrix form. LayerNorm is applied per sample, so for a feature matrix $A \in \mathbb{R}^{d\times n}$ with columns a_j it is defined as

$$\text{LN}(a_j) = \gamma \odot \frac{a_j - \mu_j \mathbf{1}_d}{\sigma_j} + \beta, \tag{14}$$

where $\mu_j = \frac{1}{d}\mathbf{1}_d^\top a_j$ and σ_j are the mean and standard deviation of the entries of a_j, and $\gamma, \beta \in \mathbb{R}^d$ are the learnable scale and shift. Collecting the columns, this is equivalent to

$$\text{LN}(A) = \text{diag}(\gamma)\, CAD + \beta \mathbf{1}_n^\top, \tag{15}$$

where $C = I_d - \frac{1}{d}\mathbf{1}_d\mathbf{1}_d^\top$ is the fixed centering projection and $D = \text{diag}(1/\sigma_1, \ldots, 1/\sigma_n)$ collects the per sample scales. With the usual stabilizing constant ϵ, σ_j is replaced by $\sqrt{\sigma_j^2 + \epsilon}$, which does not affect the argument below. The data dependence of LayerNorm is therefore confined to the diagonal matrix D, and the operation as a whole consists of a fixed left multiplication, a diagonal right multiplication, and a rank one additive shift.

For any $A \in \mathbb{R}^{d\times n}$, $\text{rank}(\text{LN}(A)) \le \text{rank}(A) + 1$. Without the bias term ($\beta = 0$), $\text{rank}(\text{LN}(A)) \le \text{rank}(A)$.

Proof. Left and right multiplication cannot increase rank, so $\text{rank}(\text{diag}(\gamma)\, CAD) \le \text{rank}(A)$. Since $\text{rank}(\beta \mathbf{1}_n^\top) \le 1$, subadditivity of rank applied to Eq. equation 15 gives the result. □

Proposition 2. *Let $H_{\text{glu}} = \text{LN}(W_v X_{\text{glu}}) \odot \text{LN}(W_g X_{\text{glu}})$, and let H_{linear} be the output of the parameter matched linear layer followed by LayerNorm. Then*

$$\text{rank}(H_{\text{glu}}) \le \min\{n, d, (r_x + 1)^2\}, \qquad \text{rank}(H_{\text{linear}}) \le \min\{n, d, r_x + 1\}. \tag{16}$$

Proof. Since $\text{rank}(W_v X_{\text{glu}}), \text{rank}(W_g X_{\text{glu}}) \le r_x$, Lemma B yields $\text{rank}(\text{LN}(W_v X_{\text{glu}})), \text{rank}(\text{LN}(W_g X_{\text{glu}})) \le r_x + 1$. The Hadamard rank inequality $\text{rank}(A \odot B) \le \text{rank}(A)\,\text{rank}(B)$ then gives the bound on H_{glu}. The bound on H_{linear} follows directly from Lemma B. □

The separation between the two bounds is thus preserved whenever $r_x \ge 1$ and $\min\{n, d\} > r_x + 1$, which is the same nontrivial rank condition as in Proposition 1 shifted by one.

C Implementation Details

In case of FastTD3, we initialized weights and biases of layer normalization into near zero values on the linear branch, for early training stability.

Listing 1: A PyTorch implementation of N-GLU block.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class NGLU_Linear(nn.Module):
    def __init__(
        self,
        input_size,
        output_size,
        use_zero_init=False,
        device='cpu',
    ):
        super(NGLU_Linear, self).__init__()
        self.output_size = output_size
        self.linear = nn.Linear(input_size, output_size * 2, device=device)
        self.ln_a = nn.LayerNorm(output_size, device=device)
        self.ln_b = nn.LayerNorm(output_size, device=device)

        if use_zero_init:
            with torch.no_grad():
                self.ln_b.weight.mul_(1e-3)
                self.ln_b.bias.fill_(1.0)

    def forward(self, x):
        ft = self.linear(x)
        a, b = ft.chunk(2, dim=-1)
        a = F.silu(self.ln_a(a))
        b = self.ln_b(b)
        return a * b

class NGLU_Conv(nn.Module):
    def __init__(
        self,
        in_channels,
        out_channels,
        kernel_size,
        stride,
        padding,
        device='cpu',
    ):
        super(NGLU_Conv, self).__init__()

        self.conv = nn.Conv2d(
            in_channels, out_channels * 2,
            kernel_size=kernel_size, stride=stride, padding=padding, device=device
        )

        self.ln_a = nn.GroupNorm(1, out_channels, device=device)
        self.ln_b = nn.GroupNorm(1, out_channels, device=device)

    def forward(self, x):
        ft = self.conv(x)
        a, b = ft.chunk(2, dim=1)
```

```
        a = F.silu(self.ln_a(a))
        b = self.ln_b(b)

        return a * b
```

We performed our experiments based on codes from Castanyer et al. (2025) (https://github.com/roger-creus/stable-deep-rl-at-scale, license N/A), Lee et al. (2025) (https://github.com/DAVIAN-Robotics/SimbaV2, Apache 2.0 license), and Seo et al. (2025) (https://github.com/younggyoseo/FastTD3, MIT license).

D Detailed Experiment Settings

We ran 6 seeds per environment for our method (N-GLU) and the strongest baselines (base PQN and base FastTD3). For Hadamax PQN, we used offical score report (https://huggingface.co/datasets/Zhao666/Hadamax_Atari) that contains 5 seeds. All other agents are trained with 3 seeds per environment.

Table 1: PQN hyperparameters.

Hyperparameter	**Value (Description)**
Learning rate	2.5e-4
Anneal lr	False (no learning rate annealing)
Num envs	128 (parallel environments)
Num steps	32 (steps per rollout per environment)
Gamma	0.99 (discount factor)
Num minibatches	32
Update epochs	2 (policy update epochs)
Max grad norm	10.0 (gradient clipping)
Start ϵ	1.0 (initial exploration rate)
End ϵ	0.005 (final exploration rate)
Exploration fraction	0.10 (exploration annealing fraction)
Q λ	0.65 (Q(λ) parameter)

Table 2: FastTD3 hyperparameters.

Hyper-parameter	**Value**
Critic Hidden Dim	1024
Actor Hidden Dim	512
Critic Learning Rate	3e-4
Actor Learning Rate	3e-4
Discount Factor	0.99
Number of Environments	128
Number of Atoms	101

E Compute Details

In Atari-57 the agents are trained using NVIDIA TITAN RTX GPU. In HumanoidBench and MuJoCo playground, the agents are trained using NVIDIA A100 GPU. Here, we report training time, number of layers, number of parameters, FLOPs, and SPS (Steps per Second).

Table 3: Training time of PQN agents in Atari-57.

Network	**Training time**
Base	1h 39m
N-GLU	1h 55m
MultiSkip+Kron	7h 26m
Hadamax	8h 32m
Impala CNN	10h 53m

Table 4: Training time of FastTD3 agents in HumanoidBench.

Network	**Training time**
Base	2h 23m
N-GLU	2h 41m
SimbaV2	30h 59m

Table 5: Training time of FastTD3 agents in MuJoCo Playground.

Network	**Training time**
Base	3h 5m
N-GLU	3h 33m

Table 6: Computational cost comparison on PQN.

Method	Layers	Params	FLOPs	SPS
Base PQN	5	1.69 M	18.7 M	8,417
N-GLU PQN	5	3.38 M	37.3 M	7,246
Hadamax PQN	5	4.13 M	325.9 M	1,628
Multiskip+Kron PQN	16	4.60 M	24.4 M	1,868
Impala PQN	17	5.84 M	417.5 M	1,276

Table 7: Computational cost comparison on FastTD3.

Method	Layers	Params	FLOPs	SPS
FastTD3	4	1.80 M	3.5 M	1,491
N-GLU FastTD3	4	3.50 M	7.0 M	1,325
SimbaV2	7	4.62 M	9.2 M	119

F Analysis of Optimization Instability of GLU on Synthetic Setting

This section provides a more thorough account of the connection between GLUs and training instability, and in particular justifies why large feature norms are themselves harmful to learning.

Why do large feature norms impair learning? Large feature and weight norms directly inflate both the magnitude and the variance of gradients. Prior work has established that when gradient variance dominates the gradient signal, the effective signal to noise ratio of parameter updates collapses, and learning stalls or diverges McCandlish et al. (2018); Faghri et al. (2020); Liu et al. (2020).

Validating stable training of N-GLU in a synthetic setting. To further investigate this phenomenon, we conduct an additional synthetic experiment using the random label MNIST setting, in which the labels are periodically reshuffled at random and the network must maintain high training accuracy despite the changing data distribution. This setting is widely used to model the nonstationary nature of RL and to study loss of plasticity, *i.e.*, the ability to keep learning from new data Lyle et al. (2023); Lewandowski et al. (2024a). It isolates training from confounding factors such as exploration and environment stochasticity, which allows us to attribute any degradation in learning ability directly to the architecture and its optimization dynamics rather than to the RL algorithm or the environment.

We train a 5 layer MLP built from GLU, Pre-LN GLU, or N-GLU layers on 1,000 MNIST samples for 1,000 epochs, reshuffling the labels every 100 epochs. All results are averaged over 5 seeds, and we report mean $\pm$ standard deviation in Figure 7.

The results show that the large feature norm of GLU is accompanied by a sharp increase in both the magnitude and the variance of the gradient. Once the variance dominates the gradient signal, the signal to noise ratio of parameter updates collapses, and learning stalls or diverges: accuracy falls to near chance level and all five seeds diverge by epoch 700. In contrast, N-GLU keeps the feature norm bounded near 20 and its gradient norm on the order of 10^{-2} with small variance, recovering 100% training accuracy after every relabeling throughout all 1,000 epochs.

Pre-LN GLU also maintains 100% training accuracy throughout training. However, its feature norm keeps increasing as it experiences more relabeling, growing from 32.1 at epoch 100 to 45.8 at epoch 900, roughly twice that of N-GLU, whose feature norm stays flat at around 20 over all 1,000 epochs. We do not observe explicit performance degradation for Pre-LN GLU in this setting. We suspect this is because memorizing random labels only requires sufficient capacity, so the network can reach 100% accuracy even with an inflated feature norm. In RL, which is considerably more complex, the same feature norm growth does translate into degraded performance (Section 4.3).

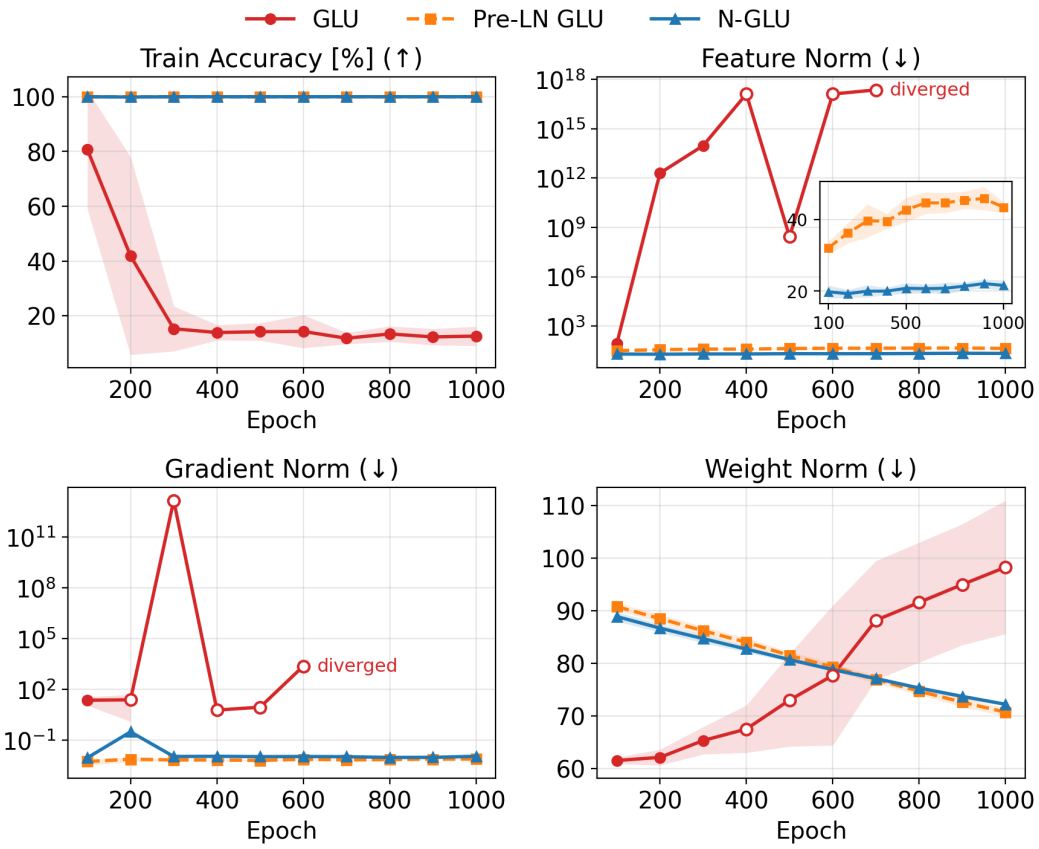

Figure 7: Training accuracy, feature norm, gradient norm, and weight norm on random label MNIST (mean ± standard deviation over 5 seeds). Hollow markers denote epochs where only a subset of seeds produced finite values. The inset shows the feature norm of Pre-LN GLU and N-GLU on a linear scale.

G CHOICE OF ACTIVATION FUNCTION IN N-GLU

Here we test different activations for our N-GLU block (Swish, Sigmoid, and ReLU) on the Atari-5 benchmark Aitchison et al. (2023), a representative subset of the full Atari-57 benchmark. Table 8 reports the final Atari-5 score of N-GLU PQN after 200M frames for each activation function.

Table 8: Final Atari-5 score of N-GLU PQN with different activation functions (200M frames).

Method	Atari-5 Score (↑)
Swish (Ours)	2.400 ± 0.060
Sigmoid	2.572 ± 0.104
ReLU	2.274 ± 0.340

As a result, Swish and Sigmoid show similar performance, while ReLU performs relatively worse. However, since this is not a full benchmark result, we expect the performance gap between Swish and Sigmoid to be negligible. This is supported by a pairwise significance test on the final Atari-5 scores (exact permutation test, $n = 3$ seeds per method): Sigmoid vs. ReLU (mean diff $= +0.4296$, $p = 0.100$), Sigmoid vs. Swish (mean diff $= +0.3848$, $p = 0.100$), and ReLU vs. Swish (mean diff $= -0.0448$, $p = 0.900$). None of these reach statistical significance, indicating that the performance gaps among the three activations are not statistically significant.

We choose Swish because SwiGLU is the most widely used GLU variant, adopted across both computer vision and language models.

H ADDITIONAL RESULTS

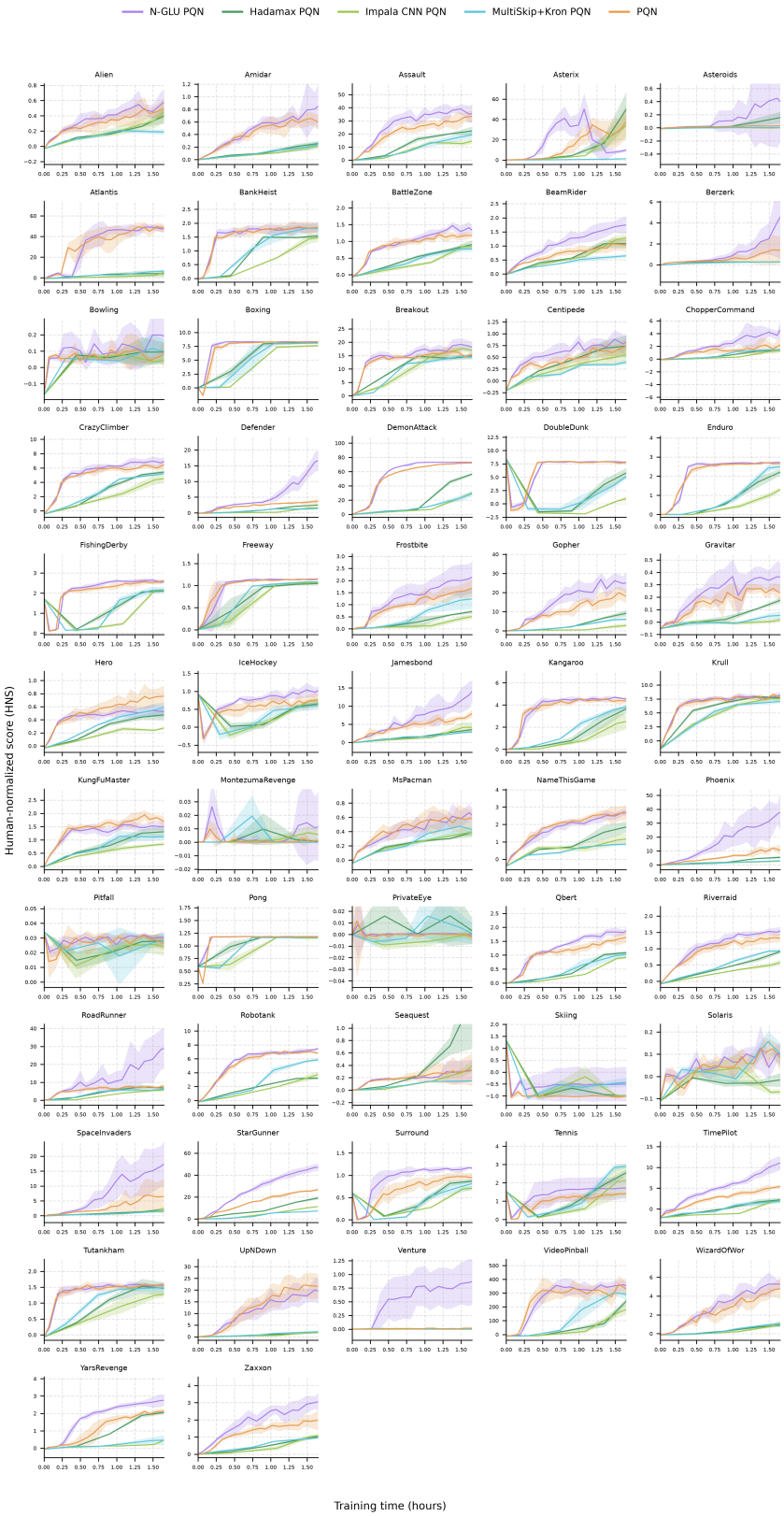

Figure 8: **Compute efficiency in Atari-57**

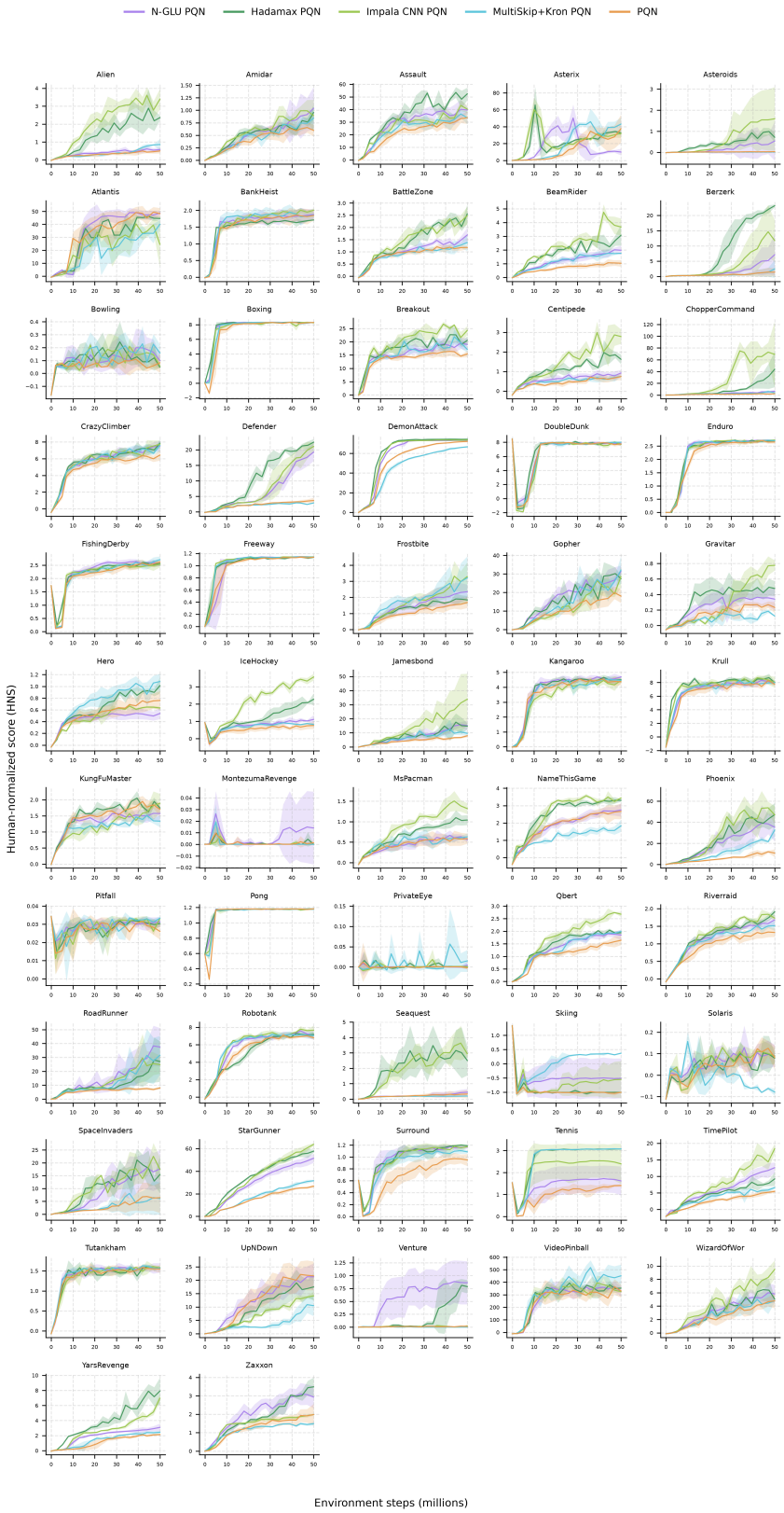

Figure 9: **Sample efficiency in Atari-57**

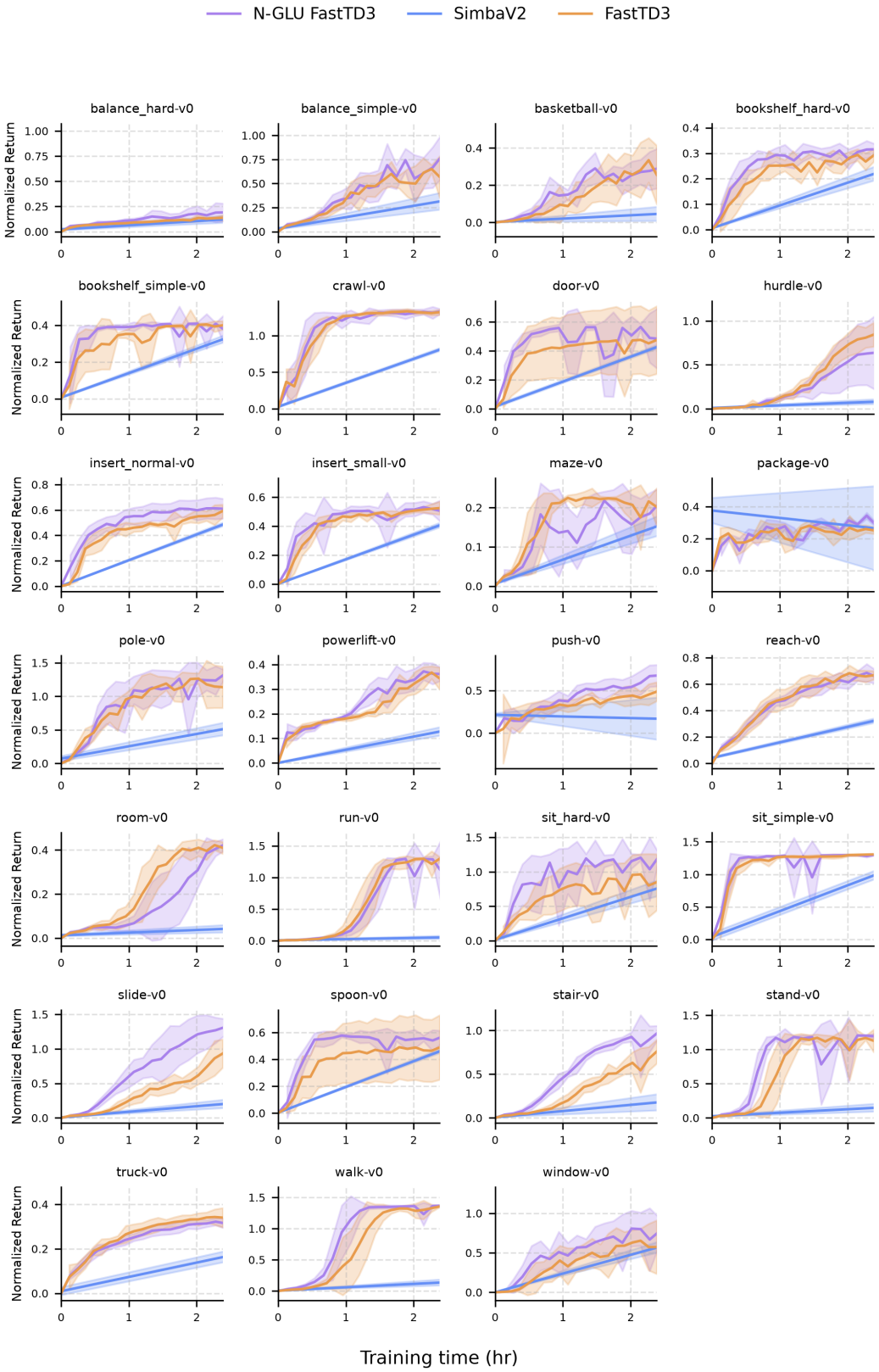

Figure 10: **Compute efficiency in HumanoidBench**

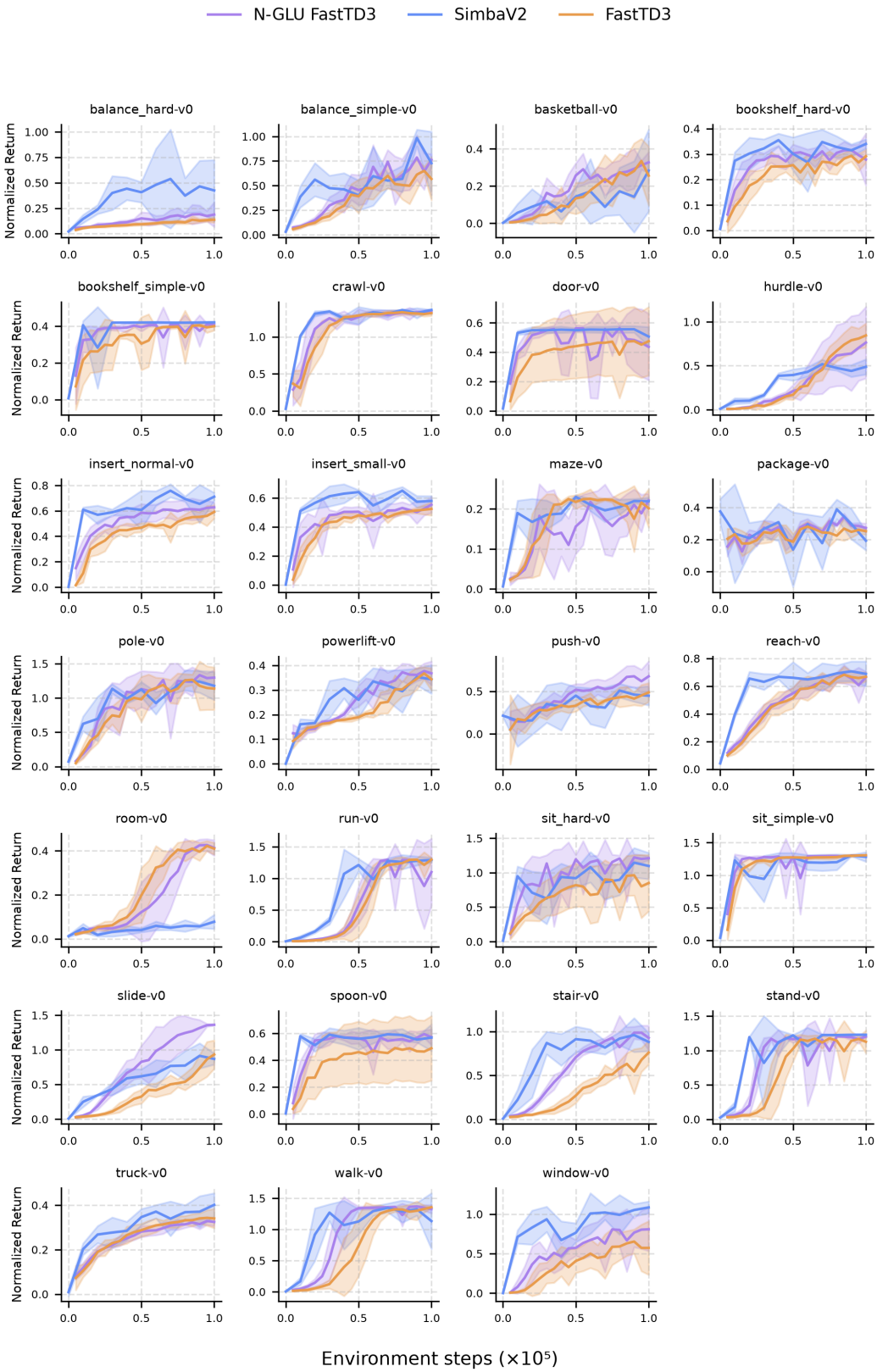

Figure 11: **Sample efficiency in HumanoidBench**