COMPLETEP FOR RL: MITIGATING INCONSISTENCIES DURING REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The maximal update parameterization (μP) has been influential in supervised and unsupervised learning, with fixed data distributions, owing to its ability to maintain feature learning across larger parameter scales. This causes more consistent learning dynamics and learned features across model sizes. In addition, optimal hyperparameters such as learning rate approximately transfer from small to larger models, minimizing the computational overhead of hyperparameter sweeps. However, it remains elusive if these benefits readily transfer to the reinforcement learning framework, where the model's learning dynamics are coupled to the shifting data distribution. Reinforcement learning agents must continually adapt to non-stationary data distribution shifts throughout training. We empirically study how two regimes of reinforcement learning agents under the "rich" CompleteP and "lazy" Neural Tangent Kernel (NTK) parameterizations affect hyperparameter transfer, feature and policy consistency. Ultimately, we show that agents trained using the CompleteP parameterization consequentially improves compute and reward efficiency compared to the NTK parameterization over 16 continuous control tasks and variants e.g. normalization and sparse rewards. Hence, we argue that adopting the CompleteP parameterization minimizes learning inconsistencies across model sizes to improve compute efficiency when scaling up.

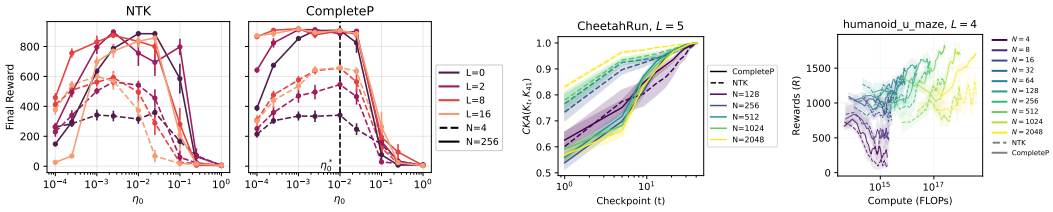


Figure 1: **CompleteP parameterization affords learning rate transfer (Left) and consistently fast feature learning across width and depth (Middle) to improve learning efficiency (Right).**

1 INTRODUCTION

Reinforcement learning (RL) provides a general framework to train agents to solve an environment by learning to maximize rewards. Similar to supervised learning (Hestness et al., 2017) and self-supervised learning (Kaplan et al., 2020; Hoffmann et al., 2022), RL has shown to also benefit from increasing the total parameters in a model (Hilton et al., 2023). However, efficiently scaling up neural networks in a stable manner in this setting remains an open challenge. Unlike supervised and unsupervised learning, RL agents face additional difficulties: (1) training data is nonstationary since the distribution of sampled trajectories depends on the evolving policy, (2) large-scale hyperparameter sweeps are prohibitively expensive due to the sample inefficiency of RL and costly environment interactions. Consequently, RL agents are typically trained with small networks, preventing systematic scaling studies to study generalization capabilities.

Two learning regimes have been identified for understanding the scaling behavior of neural networks: the "lazy" regime, arising from parameterizations such as the Neural Tangent Kernel parameterization (Yang et al., 2024b;a), where feature learning is suppressed at large widths, reducing

the network to a static kernel method (Jacot et al., 2018; Lee et al., 2019), and the "rich" or feature learning regime, where parameterizations such as mean-field and maximal update (μP) preserve the evolution of hidden representations even as width grows (Mei et al., 2018; Yang & Hu, 2020; Bordelon & Pehlevan, 2022; Vyas et al., 2023). CompleteP is μP with depth-dependent residual scaling (Bordelon et al., 2023b; 2024; Dey et al., 2025; Yang et al., 2024b) enabling learning rate transfer and improved generalization across width and depth scaling (Yang et al., 2021; Bordelon et al., 2023b; Yang et al., 2024b).

While these advances have transformed supervised and self-supervised learning, their benefits have not been systematically studied in RL, where optimization dynamics and data distributions differ substantially from supervised contexts. Thus, it is not clear whether the rich feature learning observed in large supervised models also translates to more stable and efficient RL agents as scaling increases. This motivates the core question of our work:

Question: Does rich feature learning improve an RL agent's policy consistency and learning efficiency as networks scale?

To address this, we study two scaling parameterizations (NTK and CompleteP) for feedforward residual networks (ResNets) on continuous control RL tasks. Our key contributions are:

- We demonstrate that the learning rate hyperparameter η_0 transfers across width and depth when using CompleteP, reducing hyperparameter sweep overhead.
- We analyze feature inconsistencies in RL across width and random seeds; CompleteP achieves more consistent feature learning for sufficiently wide networks.
- We show that CompleteP significantly improves compute and reward efficiency on 16 continuous control tasks and their variants compared to NTK parameterization.

2 Related Works

Proximal Policy Optimization (PPO) (Schulman et al., 2017) and related algorithms have achieved strong RL performance across locomotion, robotics, and vision-based tasks (Zakka et al., 2025; Tassa et al., 2018; Lin et al., 2023; Huang et al., 2022), often surpassing supervised or heuristic methods (Ouyang et al., 2022). However, most RL agents remain limited in size (widths < 256, depths < 4) due to the expensive nature of environment interactions (Andrychowicz et al., 2020), though GPU-accelerated simulation is beginning to ease these constraints (Freeman et al., 2021).

Scaling in Reinforcement Learning. Recent research shows that increasing model width (Hilton et al., 2023), depth (Wang et al., 2025), or the updates-to-data ratio (Fu et al., 2025; Rybkin et al., 2025) consistently improves RL performance, resulting in higher rewards. Such advances are particularly vital for robotics, where closing the sim2real gap requires agents with strong generalization and robustness. However, the influence of model parameterization on feature learning remains poorly understood. While wider models can sometimes perform well even in the "lazy" learning regime (Kumar et al., 2022; 2024), theoretical work suggests that mean-field scaling could lead to richer feature learning and improved optimization, though these results have only been established for value learning, not policy learning (Yamamoto et al., 2024). Hence, empirical validation of rich parameterizations (e.g. μP) within challenging RL domains remains limited. Our work systematically investigates how scaling model size and parameterization impacts the consistency and generalization of RL agents on the relatively demanding robotics applications.

Transferring from supervised to reinforcement context. Scaling laws, parameterization strategies, and feature learning dynamics are well investigated in supervised settings, where techniques such as mean-field or maximal update parameterizations allow hidden representations to evolve and support better generalization (Yang et al., 2021; Bordelon & Pehlevan, 2022). However, transferring these advances to RL is nontrivial (Yamamoto et al., 2024) due to additional challenges unique to RL, such as stochastic action sampling, nonstationary data distributions, catastrophic forgetting, loss of network plasticity and Temporal Difference error based loss dynamics (Sokar et al., 2023; Dohare et al., 2024; Rybkin et al., 2025; Bordelon et al., 2023a). These issues complicate stable scaling and often lead to inconsistent learning trajectories and final performance. Whether continual feature learning, made possible with parameterizations like μP (Graldi et al., 2025), can mitigate these RL-specific challenges and lead to improved policy consistency remains an open research question.

Techniques to mitigate inconsistencies. A number of modern strategies have emerged to address inconsistency and instability in RL beyond simply increasing scale. Dormant neuron reinitialization (Lee et al., 2024; Dohare et al., 2024) help restore plasticity by reactivating underutilized network units, combating the staleness that can arise during prolonged training. Layer normalization and orthogonal normalization techniques (Lee et al., 2025) have proven effective for stabilizing optimization and reducing variance across different seed runs. Moreover, explicit regularization e.g. weight decay, remains important for controlling capacity and improving generalization. The "BRO" (Bigger, Regularized, Optimistic) approach (Nauman et al., 2024) combines larger models, weight regularization, and mechanisms to balance exploration and exploitation to further reduce performance variability and support consistent learning at scale.

3 METHODOLOGY

Our goal is to compare how NTK and CompleteP parameterizations affect learning rate consistency, feature evolution, and efficiency in RL agents as we scale networks. In this section, we describe the agent architecture, parameterization schemes, and training tasks and diagnostic analysis setup.

3.1 RESIDUAL NETWORKS FOR REINFORCEMENT LEARNING

Figure 2 shows the feedforward residual network architecture we used to study the influence of network parameterization during reinforcement learning. The agent receives an observation vector $\boldsymbol{o}_t$ from the environment, which is first processed by an input layer parameterized by weights W^0 with a scaling factor $1/\sqrt{D}$ and bias $\boldsymbol{b}^0$. This produces a preactivation vector, $\boldsymbol{h}_t^1$ which is fed through a stack of L residual blocks (ResNet $\times L$). $L = 0$ means there is no residual layer but a single feature layer and an actor/critic layer as in (Kumar et al., 2022). Each residual layer applies a Rectified Linear Unit (ReLU) nonlinearity $\phi(\cdot)$, followed by weight matrix W^ℓ and bias $\boldsymbol{b}^\ell$, and two scaling factors (N and L). A skip connection adds the input preactivation $\boldsymbol{h}_t^\ell$ to the output of the layer to form $\boldsymbol{h}_t^{\ell+1}$. After passing through the final residual layer, the penultimate preactivation is processed by a nonlinearity and then to the actor layer which has a normalizing scale factor governed by ΩN.

$$\begin{aligned} \boldsymbol{h}_t^1 &= \frac{1}{\sqrt{D}} \sum_j^D W_{ij}^0 \boldsymbol{o}_t + \boldsymbol{b}^0 \,, \quad \boldsymbol{h}_t^{\ell+1} = \boldsymbol{h}_t^\ell + \frac{1}{L^\alpha} \left[\frac{1}{N} \sum_j^N W_{ij}^l \phi(\boldsymbol{h}_t^\ell) + \boldsymbol{b}^\ell \right] \,, \\ v_t &= \frac{1}{\Omega N} \sum_j^N \boldsymbol{w}_j^L \phi(\boldsymbol{h}_t^{L,v}) + b_v^L \,, \quad \boldsymbol{a}_t = \frac{1}{\Omega N} \sum_j^N W_{ij}^L \phi(\boldsymbol{h}_t^{L,\pi}) + \boldsymbol{b}_\pi^L \,. \end{aligned} \tag{1}$$

We leverage on two separate networks h^π and h^v to learn the policy and value function respectively, although a single deep network with shared features that splits into the actor and critic layers is possible using the same parameterization (Fig. 2, gray arrow). The actor head is parameterized by W^L and $\boldsymbol{b}_\pi^L$ to output the action vector $\boldsymbol{a}_t$, while the critic head uses parameters $\boldsymbol{w}^L$ and b_v^L to output the value estimate v_t. Notably, Ω exists as a knob to smoothly control the level of "feature learning" in the model. We note that this parameter is often referred to as γ in other works (Atanasov et al., 2025), but we chose Ω as to avoid confusion with the discount factor of an environment's reward.

3.2 NTK AND COMPLETEP PARAMETERIZATIONS

A central methodological choice in scaling deep networks is the selection of parameterization schemes—rules for initializing parameters and scaling hyperparameters across width (N) and depth (L). These schemes strongly affect the dynamics of learning, the preservation of meaningful features, and the transferability of hyperparameters. Our goal is to ensure desirable properties ("desiderata") as networks are scaled, especially in reinforcement learning. Refer to Appendix A for the desiderata. The NTK parameterization is designed such that as $N \to \infty$, the learned features change minimally: "lazy" learning with a rate of $\Theta(1/\sqrt{N})$. Conversely, maximal update (μP) modifies the scaling rules so feature learning capacity is retained even as width and depth grows: the change in intermediate representations remains $\Theta(1) \sim \boldsymbol{h}(t) - \boldsymbol{h}(0)$ (Yang & Hu, 2020). Additionally,

Table 1: Network initialization and hyperparameter scaling rules for width, depth and learning rate. Ω_0 and η_0 are free hyperparameters.

Network Initialization	**Input, Output**	**Hidden**
Weight variance (σ^2)	1	LN
Bias variance (σ_b^2)	1	L
Hyperparameter scaling	**NTK**	**CompleteP**
Width Scaling Parameter (Ω)	$\Omega_0/\sqrt{N}$	Ω_0
Depth Scaling Exponent (α)	0	1
Learning Rate (η)	$\eta_0\Omega$	$\eta_0\Omega$

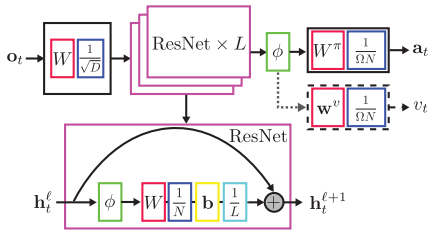


Figure 2: **Residual agent architecture.** Value function can be learned using separate or shared (grey) ResNets.

introducing a $1/L$ scaling in the residual branch affords consistent feature evolution with depth scaling. Normalizing for both width and depth scaling is the CompleteP parameterization (Dey et al., 2025), which allows all three desiderata to be simultaneously satisfied.

Table 1 summarizes the parameterizations: (1) initialize weights and biases from a zero-mean Gaussian and layer-specific variances; (2) set normalization constants (for preactivation and learning rate) according to the chosen parameterization, with Ω_0 and η_0 as meta-hyperparameters. NTK scales Ω as $1/\sqrt{N}$ and uses $\alpha = 0$ for the depth exponent, whereas CompleteP keeps Ω constant and sets $\alpha = 1$. Note that the precursor μP parameterization uses $\alpha = 0$, while scaling the other hyperparameters similarly to CompleteP. Learning rates are always scaled consistently with Ω.

For each environment an optimal learning rate η_0^* was determined for a specific width and depth configuration. Performing hyperparmeter sweeps for each configuration is computationally expensive. For NTK agents, we scaled the learning rate by a factor of $1/\sqrt{N}$ according to the learning rate scaling rule in Table 1 and used by Hilton et al. (2023). This heuristic ensures learning rates for NTK agents are close to optimal without needing additional hyperparmeter sweeps when scaling. For CompleteP agents, learning rate was a constant. We set $\Omega_0 = 1$ for most experiments, except in Appendix F to determine if there was an optimal value as seen in (Graldi et al., 2025; Atanasov et al., 2025). Note that the scaling rules in Table 1 are specifically for the ADAM optimizer which was used in all of our experiments. Though the scaling rule derivations are simpler for SGD optimizer, training agents using SGD results in lower maximum reward and slower learning performance. Hence, we focused on developing the scaling rules for ADAM optimizer instead as that is the current SOTA optimizer for RL research. Refer to Appendix A for SGD optimizer rules.

Comparatively, Standard Parameterization (SP) schemes (e.g., Kaiming, Xavier, LeCun) only maintain stable signals (desideratum 1), but not stable output learning or feature learning (desiderata 2 & 3) when scaling networks. By contrast, NTK and CompleteP both provide stable signal and output learning (desiderata 1 & 2). However, they differ in their ability to maintain feature learning (desiderata 3), where the representations of the intermediate layers change throughout training. Namely, when scaling width, the NTK scheme loses its feature learning capabilities. The CompleteP scheme maintains stable feature learning even when increasing width and depth. We empirically verify the stability of output learning and feature learning in (Appendix J).

3.3 Training on Continuous Control Tasks

We evaluated the effects of network parameterization on learning and representation by training agents on eight continuous control environments, ranging from standard DeepMind Control Suite tasks (Tassa et al., 2018) to more difficult manipulation in MuJoCo Playground (Zakka et al., 2025) and humanoid maze navigation problems (Wang et al., 2025; Bortkiewicz et al., 2024). In each setting, agents observe high-dimensional sensory inputs and learn to modulate joint torques to maximize cumulative reward. Most tasks use dense rewards to encourage learning at every step (reward shaping), but we also considered task variants with sparse rewards (Appendix N.4) given only upon successful completion, providing a greater test of representation and exploration.

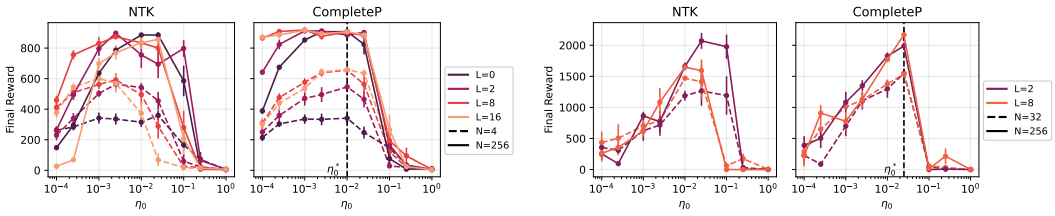
All experiments used Proximal Policy Optimization (PPO) with the Adam optimizer (Freeman et al., 2021). This is an online learning method that iteratively gathers trajectories with the current policy, updates the network, and then collects new data under the updated policy. We chose to study an on-policy learning algorithm because it reflects the most extreme case of data distribution shift in reinforcement learning algorithms: Changes in the networks from learning are immediately realized in the new collection of trajectories as data for the next update. This is in contrast to off-policy algorithms such as Soft Actor-Critic (SAC), which slow distributional shift through experience replay. Additional details on the effects of policy updates and data distribution can be found in the Appendix H. Models with varying hyperparameters, widths and depths were trained on a mix of A100s and H100s, with a total compute of 98,000 GPU hours.

3.4 Kernel and action distribution analysis

For feature and policy analyses, we constructed a synthetic evaluation dataset by varying each state dimension individually across a standardized range while holding all other dimensions at zero. Since our PPO implementation (Freeman et al., 2021) employs observation normalization, a majority of observation values are processed between the range of -3 and 3, with more extreme values taking larger magnitudes. As such, for each of the D observation dimensions, we sampled S linearly spaced values from -3 to 3, resulting in DS total samples—each with only a single dimension nonzero. (See Appendix B for an example.) This range of values ensures that we cover inputs across different levels of extremity. Ultimately the purpose of this synthetic dataset is to enable consistent and interpretable comparison of feature kernels and policy outputs across training, network configurations, and random seeds, similar to kernel analyses in supervised and unsupervised learning.

4 Learning rate hyperparameter transfer across scalings

Before scaling agent parameters, we want to reduce the need to sweep for the optimal learning rate hyperparameter, especially when training agents on highly challenging tasks like Humanoid Maze. An ideal setting is when we only need to perform a single learning rate sweep with a small width (e.g. $N = 32$) and depth (e.g. $L = 2$) model and transfer it to train a larger parameter model. We will call this the optimal learning rate η_0^*. In this section, we demonstrate that agents initialized in the CompleteP parameterization demonstrate improved learning rate hyperparameter transfer across width and depth compared to the NTK parameterization. Refer to Fig. 9d and Appendix U for learning rate transfer using Standard Parameterization.



(a) HalfCheetah (b) HumanoidMaze

Figure 3: **Learning rate hyperparameter transfers across width and depth more consistently with CompleteP parameterization.** Across environments, NTK agents require a general reduction in the learning rate with width and depth scaling, although this is done through the Ω hyperparameter. Comparatively, the same learning rate η_0^* is optimal across different scales of CompleteP agents to show a improvements in final rewards. Refer to Appendix E for other environments. Error bars indicate standard error across 10 and 5 seeds for HalfCheetah and HumanoidMaze respectively.

Figure 3 demonstrates how the final reward achieved by the agent (after the last gradient update) changes based on the learning rate (η_0) hyperparameter across different depths and widths for the NTK and CompleteP parametrized agents. For both HalfCheetah and HumanoidMaze we see that increasing either the model width and depth leads to a general increase in overall performance, supporting the trend that scaling agent parameters improves learning performance. However, naively increasing the depth in NTK agents without learning rate optimization causes a general decrease in

the final rewards achieved. NTK agents with deeper models require a lower learning rate to achieve similar performance. Instead, CompleteP agents demonstrate improved consistency where a single learning rate (η_0^*) transfers to deeper and wider models for reliable increase in learning performance. Refer to Appendix E for learning rate transfer experiments for other environments.

Hence, the transferability of learning rates across different model widths and depths applies even to the reinforcement learning context. While NTK agents can achieve similar final reward as CompleteP agents, only the latter reliably achieves our goal to reduce compute costs incurred for hyperparameter optimization. Additionally, while we do see learning rate transferring in NTK agents for some environments, CompleteP agents demonstrate a more robust transfer (Appendix E). We can now use this transferability to perform a single learning rate sweep for smaller width and depth models to determine the optimal learning rate to train reinforcement learning agents to solve significantly harder environments.

5 Inconsistencies in Representation Learning

We have seen that the CompleteP parameterization robustly improves learning performance when we scale width and depth. However, reinforcement learning is notorious for having high variance among seeds. As such, improving the consistency of feature learning could reduce sources of variance during the noisy training process. We will now analyze how features and logits evolve during training with scale.

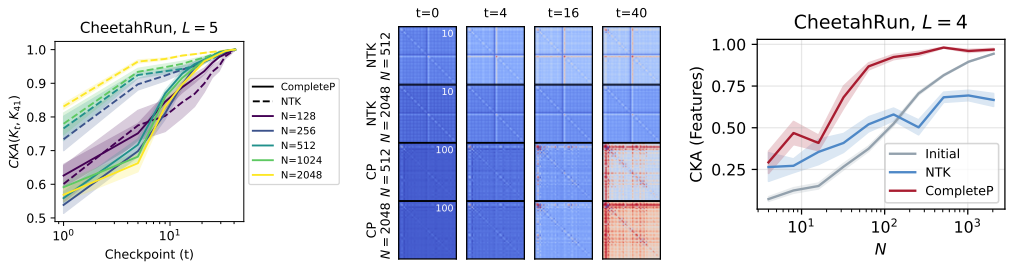


Figure 4: **CompleteP maintains consistent speed of feature learning across width scaling. Left:** As width increases, NTK agents demonstrate gradual and width-dependent feature evolution-larger widths evolve slower and hence more aligned with the feature kernel at the end of training while smaller width models demonstrate a faster rate of change in alignment. Conversely, CompleteP agents demonstrate width-consistent feature evolution–the features of larger width agents aligns with the feature kernel at the final checkpoint at the same or faster speed than smaller width agents. **Middle:** Example NTK and CompleteP feature kernels evolving at width-dependent and width-independent rates. **Right:** Feature kernels across different seeds align at a faster rate for CompleteP agents compared to NTK agents. Refer to Appendix K for other tasks. Shaded area indicates standard error over 10 seeds.

We obtained a feature matrix $H \in R^{DS \times N}$ by feeding in the synthetic dataset with DS samples (refer to section 3.4) to agents at different checkpoints of training (t). A kernel (defined as the similarity matrix $HH^\top$) of the penultimate layer activations (h^L) is then constructed at different checkpoints. Each kernel's alignment with the the kernel at the last checkpoint is visualized in Figure 4. NTK agents demonstrate a gradual increase in alignment with the final kernel. However, this alignment is width-dependent. Larger width NTK agents demonstrate a smaller rate of change across training compared to smaller width agents. This is expected as NTK agents demonstrate slower feature evolution with width scaling (Refer to Appendix J) in the order of $\Theta(1/\sqrt{N})$. Hence, the difference between the final and intermediate kernels will be persistently high due to a lack of feature learning. Whereas smaller width NTK agents demonstrate a slightly faster feature evolution. Refer to Appendix P for feature consistency in other environments.

Conversely, CompleteP agents demonstrate width-independent feature evolution where agents with different widths demonstrate similar rates of change in feature evolution (also see Appendix J) due to our feature learning parameterization. The example kernels demonstrate the difference in feature

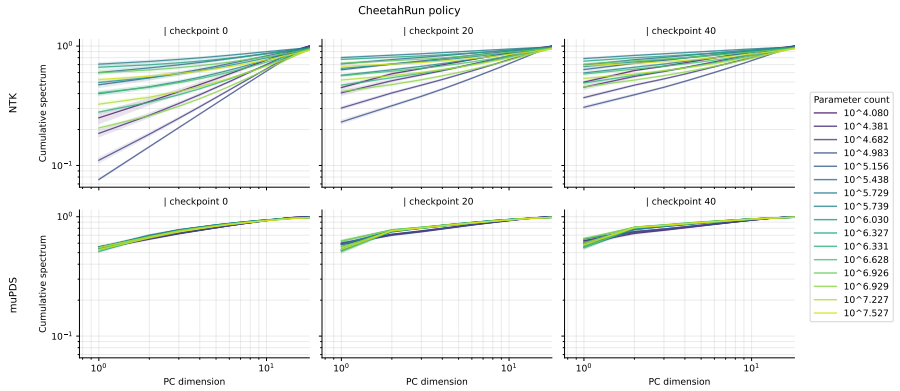


Figure 5: Combined policy eigenspectrum across checkpoints for *CheetahRun*. The synthetic tiled dataset is fed in as input, and the eigenspectrum is calculated from the resulting output features. The eigenspectrum of features of NTK agents across different parameter counts (which reflect different choices in width and depth) are spread out over the course of training. They start to converge over training but remain spread out. Notice the consistency across different parameter counts in the CompleteP agents as opposed to the NTK agents.

evolution with training by the NTK and CompleteP agents. Additionally, features in CompleteP agents initialized with different seeds demonstrate increasing consistency with width scaling compared to NTK agents (see Fig. 22), suggesting that CompleteP might be able to reduce the variance in feature learning, even in the reinforcement learning framework. It is expected for random feature kernels (gray) to align at large widths.

Additionally, NTK agents of different widths and depths demonstrate highly variable eigenspectrum in the early onset of training, but gradually learn low-diemsnional representations (Fig. 5). Conversely, since the early onset of training, CompleteP agents with different number of parameters demonstrate consistent eigenspectrum, and maintain this low-diemensional representation throughout training.

One of the key differences between the supervised and reinforcement learning framework is that the data distribution that is used to update a network's parameters is always changing in the latter (Fig. 13). The data distribution stops changing once policy learning has concluded. Non-stationary data distributions accelerate the loss of plasticity phenomenon where networks lose the ability to learn the changing distribution (Dohare et al., 2024).

Since NTK agents have reduced feature learning capabilities, they can maintain a stable proportion of dormant neurons and plasticity to adapt to changing data distribution whereas CompleteP agents produce $\Theta(1)$ feature evolution at sufficiently large widths (Fig. 16). Nevertheless, CompleteP agents demonstrate a similar (Fig. 14) proportion of dormant neurons to NTK agents, especially at larger widths. Whether CompleteP agents can continually adapt to changing data distributions needs further analysis. Still, the ability to maintain plasticity even with maximal feature updates might be a possible mechanism for continual reinforcement learning.

6 Inconsistencies in Policy Learning

To determine how logits evolve, we plot the difference in logits generated (mean squared error) at each model checkpoint (t) and the final logit distribution using the synthetic dataset. Both NTK and CompleteP agents demonstrate width independent logit evolution (Figure 6, left). As learning proceeds, we observe a similar rate of logit evolution between the largest width and smaller width agents. This systematic divergence is observed in both NTK and CompleteP agents across different environments (Fig. 23). This is expected since there are no restrictions placed on logit output evolution for NTK agents and they can evolve at $\Theta(1)$ (Fig. 17) based on our desiderata 2 (Appendix A). However, the speed of logit evolution is environment specific, with CompleteP agents showing increased rate of evolution (Fig. 23).

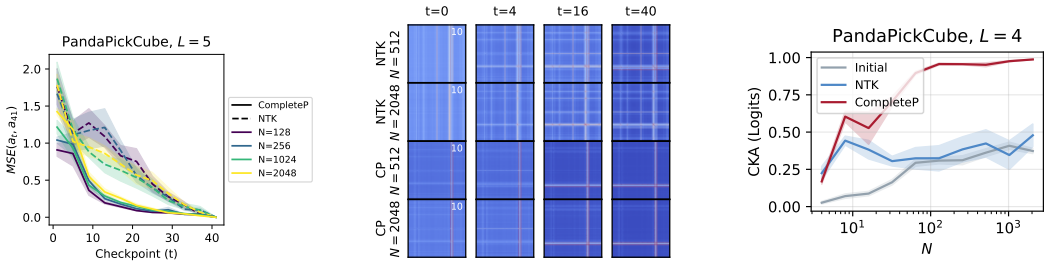
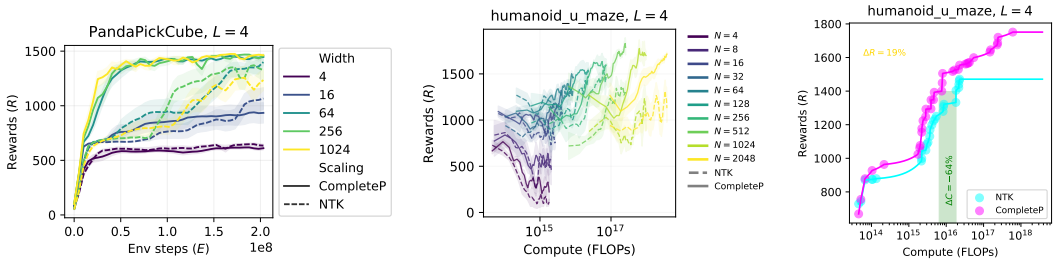


Figure 6: **Both NTK and CompleteP demonstrate consistent logit evolution, with the latter demonstrate seed-specific consistency. Left:** MSE of policy logits between the last model checkpoint and intermediate checkpoints (t). Both NTK and CompleteP agents demonstrate a similar rate of logit evolution across different width. **Middle:** Both NTK and CompleteP demonstrated consistent differences in logit kernels across widths during learning. **Right:** CompleteP shows improved logit kernels consistency across different seed initializations compared to NTK agents. Shaded area shows standard error for 10 seeds.

However, NTK agents with different seed initialization converge to policy kernels that are more distinct (Fig. 24), causing a lower kernel alignment despite width scaling (Fig. 6). Comparatively, CompleteP agents demonstrate a significantly higher consistency in seed specific policy kernels with width scaling. This is interesting in the reinforcement learning framework as two different sequences of actions that produces the same reward magnitude are equally likely to be reinforced during policy learning due to the reward dependent objective, and there are no constraints to the action sequence taken. Furthermore, the stochasticity in the action sampling process is not controlled by the seed initialization. Despite these sources of stochasticity, CompleteP agents tend to demonstrate a higher tendency to converge to the same policy, which could be driven by increased feature consistency.

7 Improving Reward and Compute Efficiency



(a) PandaPickCube (b) HumanoidMaze (c) HumanoidMaze

Figure 7: **CompleteP improves speed of policy convergence and compute efficiency when scaling across different environments. Left:** With NTK width scaling, agents require more interactions and fail to reach peak rewards compared to CompleteP; CompleteP performance is stable at large widths. **Middle:** For a given reward, NTK agents need substantially more FLOPs than CompleteP. **Right:** Pareto frontiers diverge at larger FLOPs, letting CompleteP reach higher reward with more compute (reward efficiency) or the same reward with less compute (compute efficiency).

Lastly, we want to study how scaling reinforcement learning agents across width and depth using either the NTK or CompleteP parameterizations influence reward maximizing performance. As the agent's width is increased from $N = 4$ to $N = 64$, reward maximization performance increases monotonically for both NTK and CompleteP agents, with a faster increase in performance observed in the latter. However, as the width is further increased from $N = 64$ to $N = 2048$, NTK agents demonstrate the expected model collapse or slower learning performance with the need for additional training steps to achieve similar performance as smaller width (Bordelon & Pehlevan, 2022; Noci et al., 2024). Conversely, CompleteP agents demonstrate faster policy convergence as the width

is increased, and agents with widths between $N = 256$ and $N = 2048$ show consistent reward maximization behavior, replicating the learning consistency phenomenon seen in the supervised learning context. We observe the same behavior in the reward maximizing dynamics between the NTK and CompleteP agents across different depths and environments (Refer to Appendix N), with CompleteP agents achieving state-of-the-art performance reported prior (Zakka et al., 2025). Instead, the speed of policy convergence in NTK agents approaches the limiting behavior seen in supervised learning frameworks (Noci et al., 2024; Bordelon & Pehlevan, 2022). NTK agents achieve lower learning performance than CompleteP, even when using the optimal, width-specific learning rate that maximizes reward for each model size. Conversely, agents with Standard Parameterization can reach similar maximal performance to CompleteP, but only when the learning rate is carefully tuned for each width, requiring significant compute overheads for hyperparameter sweeps (Appendix U).

Compute refers to the number of model parameters multiplied by the number of gradient steps and environment steps. Since all the other hyperparameters such as the same number of parallel environments, mini-batch size, and trajectory rollout lengths were fixed (Refer to Appendix C), this is a sufficient metric to determine whether the CompleteP parameterization could improve compute efficiency for reinforcement learning agents. Figure 8 (middle) shows the average rewards agents of different widths between $N = 4$ to $N = 2048$ achieve based on the Humanoid Maze task with different compute budgets. The Humanoid Maze task cannot be easily solved by a network with a small number of parameters (Bortkiewicz et al., 2024), as the agent needs to learn to balance, move and reach a target. Hence, scaling parameters matters in this task, and we see that increasing the widths of CompleteP agents improves the maximum rewards attained while NTK agents demonstrate a slower onset of learning with the same compute budget. Refer to Fig. 46 for learning performance against wallclock runtime.

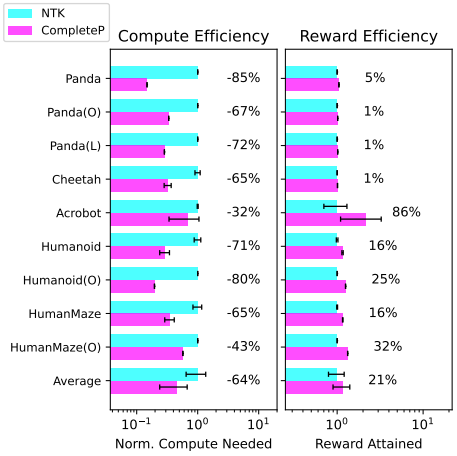


Figure 8: **Compute and reward efficiency summary.** Error bars are standard error across different depths, except for average which is across environments.

We visualize the pareto optimal compute frontier for both CompleteP (green) and NTK agents (Fig. 7, right). We then fit isotonic regression lines to the pareto frontier points. We define compute efficiency as the amount of compute flops needed to achieve 95% of the maximum reward achieved either by the NTK or CompleteP agent, which translates to a reward of approximately 1450 in Humanoid Maze. We also define reward efficiency as the amount of rewards attained based on 95% of compute used to train the models. In this task, we observe CompleteP agents requiring a significantly lower amount of compute budget ($\Delta C = -64\%$) to achieve the same amount of reward achieved by NTK agents, and a significantly higher amount of reward ($\Delta R = 19\%$) for the same compute budget, demonstrating the benefits of feature learning in reinforcement learning. Refer to Appendix N for compute frontiers for other environments.

Figure 8 and Appendix O shows a summary of the compute and reward efficiencies over five environments and their variants. Initializing network parameters using orthogonal initialization instead of Gaussian distribution improved NTK agent's learning performance for some environments (Appendix N.2). Additionally, layer normalization has been shown to improve reward learning performance (Lee et al., 2025). We noticed a mixture of improvements in NTK agents in terms of maximal reward that can be attained at lower compute budgets (Appendix N.3). Despite these techniques, CompleteP agents with orthogonal or layer normalization techniques and consistently demonstrated improved reward and compute efficiencies, suggesting that CompleteP parameterization can work synergistically with other methods to improve reinforcement learning performance. Additionally, CompleteP agents demonstrate significantly higher efficiencies in environments with sparse rewards while NTK agents failed to learn the task (Appendix N). Hence, CompleteP parameterization improves compute and reward efficiency as seen in supervised learning contexts (Yang & Hu, 2020; Vyas et al., 2023; Noci et al., 2024). Refer to Appendix O for details on compute needed and reward attained for each environment and variant.

8 DISCUSSION

Our work demonstrates that the reinforcement learning (RL) context introduces significant inconsistencies in reward maximization, feature learning, and policy alignment across different model widths and depths—a phenomenon not rigorously studied before. We studied the influence of the Neural Tangent Kernel (NTK) and CompleteP parameterization on how feature learning influences learning policies for continuous control tasks. Despite the non-stationary data distribution and loss of plasticity phenomena specific to the reinforcement learning context, CompleteP better mitigates these inconsistencies compared to the Neural Tangent Kernel (NTK) parameterization by maintaining feature learning and aligning action distributions with scale. Additionally, the CompleteP parameterization enables learning rate transfer across widths and depths, reducing hyperparameter tuning costs to train large scale RL agents. Importantly, CompleteP enhances compute and reward efficiency, particularly in complex tasks like humanoid locomotion where NTK agents fail to achieve comparable performance with the same compute budget. These findings align with prior work in supervised learning (Yang & Hu, 2020; Vyas et al., 2023) but extend them to the non-stationary RL setting, where policy evolution and environmental interactions exacerbate divergence in agent solutions (Menache et al., 2005). We clarify that the observed instability of NTK parameterization in RL arises from reduced feature learning capacity, rather than non-stationary data per se. While continuous-control tasks do exhibit evolving data distributions (see Fig. 12), our key finding is that enabling feature learning becomes increasingly critical as task complexity grows. Although CompleteP consistently improves feature evolution and kernel alignment (Fig. 4 and Appendix Figs. 15–24), the degree of improvement is moderate. We do not claim super-consistency (Noci et al., 2024); rather, feature learning parameterizations partially mitigate, but do not eliminate, the sources of inconsistency unique to RL.

Several important directions remain for future work. First, while we demonstrate CompleteP's benefits in MLPs, studying architectures combining both convolutional and feedforward modalities (e.g., for vision-based control) could reveal new scaling challenges and further inconsistencies (Hilton et al., 2023; Lin et al., 2025). Second, the interaction between CompleteP and RL-specific hyperparameters such as parallel environments or rollout length, requires systematic analysis, as these may affect the consistency benefits we observe. Third, evaluating CompleteP using off-policy algorithms (e.g., Soft Actor-Critic (Zakka et al., 2025)) or value-based methods would definitively demonstrate test its generality across existing RL approaches. To strengthen theoretical foundations, we propose: (1) Using intrinsic performance metrics to convert sigmoid-shaped learning curves into linear relationships for power-law fitting (Hilton et al., 2023), enabling direct comparison of scaling exponents across tasks; (2) Expanding the task suite from single-agent environments to multi-agent environments with continually changing dynamics (Chiappa et al., 2022; Dohare et al., 2024) to identify CompleteP's limits; and (3) Analyzing representations learned using CompleteP and biological neural networks to make predictions if similar feature learning regimes exist in natural systems (Kumar et al., 2024). Additionally, developing theoretical connections between CompleteP and distributional value functions (Dabney et al., 2018) may yield further improvements in stability for reinforcement learning. These steps would strengthen CompleteP's role in bridging theoretical parameterization and practical large-scale reinforcement learning approaches.

ETHICS AND REPRODUCIBILITY STATEMENT

We have used Large Language Models (LLMs) for assistance in proofreading, writing, and coding.

REFERENCES

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Alexander Atanasov, Alexandru Meterez, James B. Simon, and Cengiz Pehlevan. The optimization landscape of sgd across the feature learning strength, 2025. URL https://arxiv.org/abs/2410.04642.

Blake Bordelon and Cengiz Pehlevan. Self-consistent dynamical field theory of kernel evolution in wide neural networks. *Advances in Neural Information Processing Systems*, 35:32240–32256, 2022.

Blake Bordelon, Paul Masset, Henry Kuo, and Cengiz Pehlevan. Loss dynamics of temporal difference reinforcement learning. *Advances in Neural Information Processing Systems*, 36:14469–14496, 2023a.

Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. *arXiv preprint arXiv:2309.16620*, 2023b.

Blake Bordelon, Hamza Chaudhry, and Cengiz Pehlevan. Infinite limits of multi-head transformer dynamics. *Advances in Neural Information Processing Systems*, 37:35824–35878, 2024.

Michał Bortkiewicz, Władysław Pałucki, Vivek Myers, Tadeusz Dziarmaga, Tomasz Arczewski, Łukasz Kuciński, and Benjamin Eysenbach. Accelerating goal-conditioned rl algorithms and research. *arXiv preprint arXiv:2408.11052*, 2024.

Alberto Silvio Chiappa, Alessandro Marin Vargas, and Alexander Mathis. Dmap: a distributed morphological attention policy for learning to locomote with a changing body. *Advances in Neural Information Processing Systems*, 35:37214–37227, 2022.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

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.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244, 2022.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

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.

Jacopo Graldi, Alessandro Breccia, Giulia Lanzillotta, Thomas Hofmann, and Lorenzo Noci. The importance of being lazy: Scaling limits of continual learning. *arXiv preprint arXiv:2506.16884*, 2025.

Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):5721, 2021.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory F. Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017. URL `http://arxiv.org/abs/1712.00409`.

Jacob Hilton, Jie Tang, and John Schulman. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*, 2023.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017. URL https://arxiv.org/abs/1706.02515.

M Ganesh Kumar, Cheston Tan, Camilo Libedinsky, Shih-Cheng Yen, and Andrew YY Tan. A nonlinear hidden layer enables actor–critic agents to learn multiple paired association navigation. *Cerebral Cortex*, 32(18):3917–3936, 2022.

M Ganesh Kumar, Blake Bordelon, Jacob A Zavatone-Veth, and Cengiz Pehlevan. A model of place field reorganization during reward maximization. *bioRxiv*, pp. 2024–12, 2024. doi: 10.1101/2024.12.12.627755.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. *arXiv preprint arXiv:2410.09754*, 2024.

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.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Zijun Lin, Haidi Azaman, M Ganesh Kumar, and Cheston Tan. Compositional learning of visually-grounded concepts using reinforcement. *arXiv preprint arXiv:2309.04504*, 2023.

Zijun Lin, M Ganesh Kumar, and Cheston Tan. Human-like compositional learning of visually-grounded concepts using synthetic data. In *Will Synthetic Data Finally Solve the Data Access Problem?*, 2025.

Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

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, 2024.

Lorenzo Noci, Alexandru Meterez, Thomas Hofmann, and Antonio Orvieto. Super consistency of neural network landscapes and learning rate transfer. *Advances in Neural Information Processing Systems*, 37:102696–102743, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Oleh Rybkin, Michal Nauman, Preston Fu, Charlie Snell, Pieter Abbeel, Sergey Levine, and Aviral Kumar. Value-based deep rl scales predictably. *arXiv preprint arXiv:2502.04327*, 2025.

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

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

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.

Nikhil Vyas, Alexander Atanasov, Blake Bordelon, Depen Morwani, Sabarish Sainathan, and Cengiz Pehlevan. Feature-learning networks are consistent across widths at realistic scales. *Advances in Neural Information Processing Systems*, 36:1036–1060, 2023.

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

Kakei Yamamoto, Kazusato Oko, Zhuoran Yang, and Taiji Suzuki. Mean field langevin actor-critic: Faster convergence and global optimality beyond lazy learning. In *Forty-first International Conference on Machine Learning*, 2024.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

Greg Yang and Edward J Hu. Feature learning in infinite-width neural networks. *arXiv preprint arXiv:2011.14522*, 2020.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL https://arxiv.org/abs/2203.03466.

Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning, 2024a. URL https://arxiv.org/abs/2310.17813.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=17pVDnpwwl.

Anthony Zador, Sean Escola, Blake Richards, Bence Ölveczky, Yoshua Bengio, Kwabena Boahen, Matthew Botvinick, Dmitri Chklovskii, Anne Churchland, Claudia Clopath, et al. Catalyzing next-generation artificial intelligence through neuroai. *Nature communications*, 14(1):1597, 2023.

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.

A Details of NTK and μP Parameterization

A.1 Primer on Parameterizations (SP, NTK, μP) and Large width limits for Supervised Learning

Before describing how we apply μP for RL training, we first describe how it is implemented in the more classic setting of supervised learning. We will first illustrate the width scaling in multilayer perceptron (MLPs) following arguments similar to those in Bordelon & Pehlevan (2022); Yang & Hu (2020); Vyas et al. (2023).

Width Scaling Consider a depth L multi-layer perceptron $f(x) : \mathbb{R}^D \to \mathbb{R}$ maps inputs $x \in \mathbb{R}^D$ to output predictions f is defined recursively as

$$f(x) = \frac{1}{N^{a_L}} \sum_{i=1}^{N} w_i^L \phi(h_i^L) \ , \ h_i^{\ell+1} = \frac{1}{N^{a_\ell}} \sum_{j=1}^{N} W_{ij}^\ell \phi(h_j^\ell) \ , \ h^1 = \frac{1}{N^{a_0}} \sum_{j=1}^{D} W_{ij}^0 x_j \tag{2}$$

The parameters $\theta = \{w_i^L, \{W_{ij}^\ell\}_{\ell=1}^{L-1}, W^0\}$ are randomly initialized with variances

$$w_i^L \sim \mathcal{N}(0, N^{-2b_L}) \tag{3}$$

$$W_{ij}^\ell \sim \mathcal{N}(0, N^{-2b_\ell}) \ , \ \ell \in \{1, ..., L-1\} \tag{4}$$

$$W_{ij}^0 \sim \mathcal{N}(0, N^{-2b_0}) \tag{5}$$

For a given optimizer (SGD, Adam, etc), we must also (potentially) specify a rule for how to scale the learning rate for each hidden layer with N. The parameters θ are updated with a stochastic gradient of the loss $\mathcal{L}(\theta) = \mathbb{E}_x \ell(\theta, x)$ each step on a minibatch $\mathfrak{B}_t$ drawn from the data distribution. The function $\ell(\theta, x)$ represents the loss associated with data point x. The updates take a general form

$$\theta(t+1) = \theta(t) - \mathsf{OPT}\left(\{\eta^\ell\}, \left\{ \frac{\partial}{\partial \theta} \ell(\theta(t), x) \right\}_{x \in \mathfrak{B}_t} \right) \tag{6}$$

where OPT is a transformation of the gradients $\frac{\partial}{\partial \theta}\ell(\theta, x)$ that is specific to the optimizer (SGD, Adam, etc) and depends on the set of learning rates $\{\eta^\ell\}$. For example, the SGD update is simply

$$W_{ij}^\ell(t+1) = W_{ij}^\ell(t) - \eta_\ell \mathbb{E}_{x \in \mathfrak{B}_t} \frac{\partial}{\partial W_{ij}^\ell} \ell(\theta(t), x). \tag{7}$$

The last scaling criterion is choosing how the learning rates should be updated

$$\eta_\ell = \bar{\eta} N^{c_\ell}. \tag{8}$$

A parameterization is a collection of choices for the scaling exponents for the parameters and learning rates $\{(a_\ell, b_\ell, c_\ell)\}$ that enable the limit of width $N \to \infty$ with the input dimension D, fixed batch size, and fixed number of training steps under the optimizer of choice.

To derive a stable *feature-learning* parameterization, we desire the following properties (consistent with the desiderata of Yang & Hu (2020)) as the width N goes to infinity

Desiderata: For reliable and efficient scaling, we require:

1. **Stable signal:** Pre-activation variances $\rho(h^\ell)$ remain $\Theta(1)$ regardless of width N and depth L.
2. **Stable output learning:** The magnitude of output changes during learning $(\boldsymbol{a}(t) - \boldsymbol{a}(0))$ is $\Theta(1)$, ensuring functional changes are well-behaved as the network scales.
3. **Stable feature learning:** The change in intermediate representations $(\boldsymbol{h}(t) - \boldsymbol{h}(0))$ is separated from the choice of width and depth, and can be controlled from the hyperparameter Ω_0.

We compare various parameterizations and their properties under SGD in Table 2. We repeat this exercise for Adam in Table 3.

Params Feature Learning	(a_L, b_L, c_L)	(a_ℓ, b_ℓ, c_ℓ)	(a_0, b_0, c_0)	Stable Signal	Function Learning
Standard No	$(0, \frac{1}{2}, 0)$	$(0, \frac{1}{2}, 0)$	$(0, 0, 0)$	Yes	No
NTK No	$(\frac{1}{2}, 0, 0)$	$(\frac{1}{2}, 0, 0)$	$(0, 0, 0)$	Yes	Yes
MF Yes	$(1, 0, 1)$	$(\frac{1}{2}, 0, 1)$	$(0, 0, 1)$	Yes	Yes
μP Yes	$(\frac{1}{2}, \frac{1}{2}, 0)$	$(0, 0, 1)$	$(0, 0, 1)$	Yes	Yes

Table 2: The rules for scaling the model under the **SGD optimizer** , $\mathsf{OPT} = \mathsf{SGD}$. The function learning and feature learning columns indicate that the function (and resp. feature) updates are stable and $\Theta(1)$ as $N \to \infty$. The mean field (MF) and μP parameterizations are equivalent in their training dynamics.

Param.	(a_L, b_L, c_L)	(a_ℓ, b_ℓ, c_ℓ)	Stable Signal	Function Learning	Feature Learning
Standard	$(0, \frac{1}{2}, 0)$	$(0, \frac{1}{2}, 0)$	Yes	No	No
NTK	$(\frac{1}{2}, 0, \frac{1}{2})$	$(\frac{1}{2}, 0, \frac{1}{2})$	Yes	Yes	No
MF/μP	$(1, 0, 0)$	$(1, -\frac{1}{2}, 0)$	Yes	Yes	Yes

Table 3: The rules for scaling the model under the **Adam optimizer** , $\mathsf{OPT} = \mathsf{Adam}$. The function learning and feature learning columns indicate that the function (and resp. feature) updates are stable and $\Theta(1)$ as $N \to \infty$. The mean field (MF) and μP parameterizations are equivalent in their training dynamics. Note that a_0, b_0, c_0 is similar to a_L, b_L, c_L.

We note that the main change in our derivation for accomplishing this with the Adam optimizer comes from modeling the update as a simplified version of Adam without the exponential moving average, SignSGD (Dey et al., 2025):

$$W_{ij}^\ell(t+1) = W_{ij}^\ell(t) + \eta^\ell \frac{1}{Z_{ij}^\ell} g_i^{\ell+1} \phi(h_j^\ell),\ Z_{ij}^\ell = \sqrt{(g_i^{\ell+1})^2 \phi(h_j^\ell)^2},\ \mathbf{g}^\ell \equiv \frac{\partial \mathcal{L}^\ell}{\partial \mathbf{h}^\ell} \tag{9}$$

A.2 Depth Scaling

Following Bordelon et al. (2023b); Yang et al. (2024b), we study depth scaling of deep residual networks

$$h_i^{\ell+1} = h_i^\ell + L^{-\alpha} N^{-a_\ell} \sum_{j=1}^N W_{ij}^\ell \phi(h_j^\ell) \tag{10}$$

Now we can ask which multipliers α and learning rate scalings η_ℓ allow for stable signal propagation, stable function updates, and stable feature updates. A summary of popular depth scalings is provided in Table 4.

Parameterization	(a_L, b_L, c_L)	(a_ℓ, b_ℓ, c_ℓ)	Stable Signal	Function Learning	Feature Learning
Standard ResNet ($\alpha = 0$)	$(0, \frac{1}{2}, 0)$	$(0, \frac{1}{2}, 0)$	Yes	No	No
Scaled-ResNet ($\alpha = \frac{1}{2}$)	$(\frac{1}{2}, 0, \frac{1}{2})$	$(\frac{1}{2}, 0, \frac{1}{2})$	Yes	Yes	No
CompleteP ($\alpha = 1$)	$(1, 0, 0)$	$(1, -\frac{1}{2}, 0)$	Yes	Yes	Yes

Table 4: The rules for scaling the depth under SGD

A.3 Training with Bias Parameters

Here, we consider the effect of trainable bias parameters

$$h_i^{\ell+1} = h_i^\ell + \frac{1}{L^\alpha}\left[\frac{1}{N^{a_\ell}}\sum_{j=1}^{N} W_{ij}^\ell \phi(h_j^\ell) + b_i^\ell\right] \tag{11}$$

The bias parameters are initialized with

$$\text{Var}(b_i^\ell) = \Theta_{N,L}(1). \tag{12}$$

The biases need to move by $\Delta b_i^\ell = \Theta_{N,L}(L^{-1+\alpha})$. We summarize the learning rates needed to achieve this change in Table 5.

Param.	Bias Learning Rate SGD	Bias Learning Rate Adam
NTK	$\Theta(L^{-1+2\alpha})$	$\Theta(L^{-1+\alpha})$
MF	$\Theta(L^{-1+2\alpha}N)$	$\Theta(L^{-1+\alpha})$

Table 5: How bias parameters are scaled for SGD and Adam.

A.4 Standard Parameterization, LeCun Scalings

Following standard practice (Klambauer et al., 2017) and removing activation scaling by width and depth, we get the following architecture:

$$\begin{aligned} h_\mu^{(0)} &= W^{(\text{in})} x_\mu + b^{(\text{in})} & h_\mu^{(\ell+1)} &= h_\mu^{(\ell)} + W^{(\ell)}\,\phi\big(h_\mu^{(\ell)}\big) + b^{(\ell)} \\ h_\mu^{(\text{out})} &= W^{(\text{out})}\,\phi\big(h_\mu^{(L)}\big) + b^{(\text{out})} & f_\mu &= h_\mu^{(\text{out})} \end{aligned} \tag{13}$$

With LeCun normal scalings, the variance of the weights are drawn based on their "fan in" or input dimensionality. In our architecture that translates to the following:

$$W_{ij}^{\text{in}} \sim \mathcal{N}\big(0,\, \tfrac{1}{D}\sigma_W^2\big)\,, \qquad W_{ij}^\ell,\, W_{ij}^{\text{out}} \sim \mathcal{N}\big(0,\, \tfrac{1}{N}\sigma_W^2\big)\,, \qquad b_i^{\text{in}},\, b_i^\ell,\, b_i^{\text{out}} = 0 \tag{14}$$

With LeCun uniform, the only change is that the distribution sampled from is uniform. The bounds are setup such that the variance still corresponds to $\frac{1}{\text{fan\_in}}\sigma_W^2$. Biases are still initialized from zero.

$$W_{ij}^{\text{in}} \sim \mathcal{U}\left(-\sigma_W\sqrt{\tfrac{3}{D}},\, \sigma_W\sqrt{\tfrac{3}{D}}\right), \qquad W_{ij}^\ell,\, W_{ij}^{\text{out}} \sim \mathcal{U}\left(-\sigma_W\sqrt{\tfrac{3}{N}},\, \sigma_W\sqrt{\tfrac{3}{N}}\right) \tag{15}$$

B Synthetic Observation Dataset for Feature Analysis

For feature and policy analysis, we constructed a synthetic evaluation dataset by varying each observation dimension individually across a consistent range while fixing all other dimensions at zero. Specifically, for an environment with D observation dimensions, we select S linearly spaced values between -3 and 3 for each feature. For each dimension, S samples are generated where only that dimension is varied and all others are set to zero, producing a total of SD samples.

This approach matches the input range encountered by agents during training, as we utilize running observation normalization (Freeman et al., 2021). The synthetic dataset provides a controlled basis for evaluating feature and kernel responses along single input axes, and supports consistent comparisons across network width, depth, and seeds.

Example: Suppose the observation has $D = 2$ dimensions and we choose $S = 7$ samples per dimension, the varied values are -3, -2, -1, 0, 1, 2, and 3. The resulting dataset contains 14 samples in total. Each sample is a column in the table below corresponding to one synthetic input vector.

Table 6: Synthetic evaluation dataset for two features ($D = 2$), with each feature dimension varied over 7 values ($S = 7$); other features set to zero. **Each column represents one sample (input vector).**

	Vary Feature 1							Vary Feature 2						
Sample #	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Feature 1	-3	-2	-1	0	1	2	3	0	0	0	0	0	0	0
Feature 2	0	0	0	0	0	0	0	-3	-2	-1	0	1	2	3

The full dataset is created by concatenating these samples across all D dimensions. This enables direct and interpretable comparison of how each network responds to changes along individual input axes across all experiments.

C CONTINUOUS CONTROL ENVIRONMENT DETAILS

We provide additional details of the control environments used. For environment configurations we use the ones provided by Zakka et al. (2025). We also utilize domain randomization for G1 Humanoid, and access to a privileged state.

Environment Name	Action Dim	Policy Obs Dim	Value Obs Dim
CartpoleSwingup	1	5	5
CartpoleSwingupSparse	1	5	5
PandaPickCube	8	66	66
G1JoystickRoughTerrain	29	103	216
CheetahRun	6	17	17
SwimmerSwimmer6	5	25	25
AcrobotSwingup	1	6	6
AcrobotSwingupSparse	1	6	6
humanoid	17	271	271
humanoid_u_maze	17	271	271

Table 7: Dimensionalities for actions and observations across environments.

C.1 HYPERPARAMETER SELECTION

Table 8: Base Hyperparameters for key Environments

Parameter	CartpoleSwingup	PandaPickCube	G1JoystickRoughTerrain
env_name	`CartpoleSwingup`	`PandaPickCube`	`G1JoystickRoughTerrain`
policy_obs_key	`state`	`state`	`state`
value_obs_key	`state`	`state`	`privileged_state`
Ω	1.0	1.0	1.0
activation	`relu`	`relu`	`relu`
num timesteps	100,000,000	200,000,000	2,000,000,000
num evals	21	41	41
num envs	2048	2048	32,768
num minibatches	32	32	32
update epochs	16	8	5
unroll length	30	10	32
minibatch size	1024	512	1024

D FLOPS DERIVATION

We calculate our FLOPs based on the PPO implementation in Freeman et al. (2021) and following convention from (Hilton et al., 2023). We describe our quantities below.

For the resnet style architecture, we estimate FLOPs per forward pass based on the parameter count.

To count parameters we count the main quantity LN^2. Biases are neglible in size.

Definitions

`minibatch_size` Number of environment steps in each minibatch fed to the optimizer

`num_minibatches` How many minibatches are processed per batch collection.

`unroll_length` (Additional) trajectory length collected for the generalized advantage estimation

`num_timesteps` Total environment steps gathered over training

`num_evals` Number of checkpoints for evaluation (including the initial one).

`update_epochs` Times the same batch is reused inside a PPO update (e.g. policy, value, and entropy losses).

`flops_per_forward_pass` FLOPs required for one network forward pass.

Total model cost

$$\text{env\_steps\_per\_train\_step} = \text{minibatch\_size} \times \text{num\_minibatches} \times \text{unroll\_length} \tag{16}$$

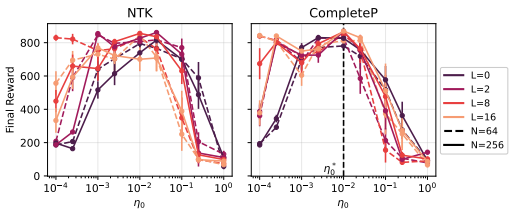
$$\text{num\_train\_steps\_per\_epoch} = \left\lceil \frac{\text{num\_timesteps}}{(\text{num\_evals} - 1) \times \text{env\_steps\_per\_train\_step}} \right\rceil \tag{17}$$

$$\text{unique\_env\_steps} = \text{num\_train\_steps\_per\_epoch} \times \text{env\_steps\_per\_train\_step} \tag{18}$$

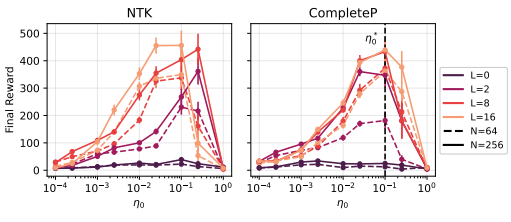
$$\text{total\_model\_flops} = 3 \times \text{flops\_per\_forward\_pass} \times \text{update\_epochs} \times \text{unique\_env\_steps} \tag{19}$$

Equation equation 19 gives the FLOP budget for one training epoch; summing over epochs yields the overall training cost.

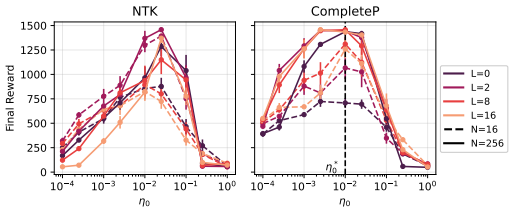
E LEARNING RATE HYPERPARAMETER TRANSFER



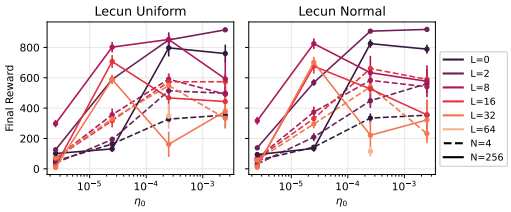
(a) CartpoleSwingup



(b) AcrobotSwingup



(c) PandaPickCube



(d) HalfCheetah, Standard Parameterization

Figure 9: **Learning rate transfers more consistently across width and depth with CompleteP.** Agents scaled using the NTK, CompleteP parameterizations were evaluated to determine if a single learning rate could transfer across agents with different depths (L) and widths (N). Increasing the depth required lowering the learning rate for NTK agents. Instead, CompleteP agents show a tighter consistency in learning rates across depth and width, with deeper networks showing improved final reward performance, suggesting that the learning rate hyperparameter could transfer across depth with the proposed parameterization. Additionally, we also observe improved stability across learning rates with CompleteP. We see a weaker consistency in learning rate transfer when we initialize our residual agents using standard parameterizations such as Lecun Uniform and Lecun Normal. Error bars indicate standard error across 10 seeds.

Scaling deep reinforcement learning—especially for large, high-dimensional embodied intelligence tasks (Gupta et al., 2021; Duan et al., 2022; Zador et al., 2023)—is often bottlenecked by the need to sweep learning rate hyperparameters for each new model width, depth, and environment. For NTK and Standard parameterizations, the optimal learning rate changes with scale, requiring repeated and expensive searches that multiply total compute cost.

In contrast, our results (Figure 3) show that CompleteP parameterization enables consistent transfer of a single optimally-tuned learning rate η_0^* from small to large models, across both simple and challenging tasks. This removes the need for repeated tuning and provides stable performance gains as models scale.

While NTK and Standard Parameterization agents can match CompleteP performance, they require exhaustive grid search for every width, depth and learning rate configuration, which is impractical for large-scale, generalist RL. CompleteP, by reducing this overhead, offers a practical and scalable solution in line with scaling laws that predict continued gains with increased capacity (Hilton et al., 2023; Rybkin et al., 2025; Fu et al., 2025).

In summary, the ability to reliably transfer learning rate hyperparameters across scales and environments is a distinct practical advantage of CompleteP, streamlining RL research and supporting the efficient training of ever-larger agents.

F OPTIMAL FEATURE LEARNING HYPERPARMATER FOR RL

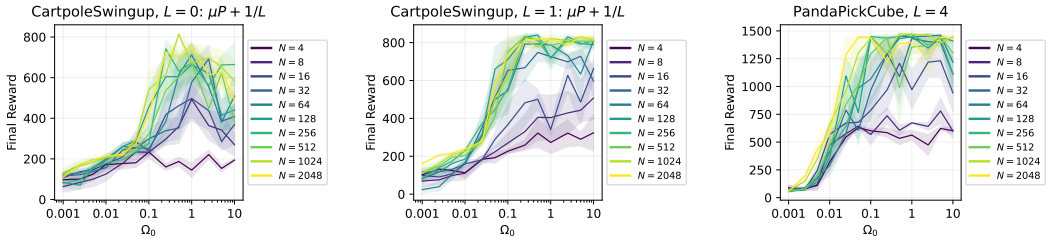


Figure 10: **RL performance as a function of the feature learning hyperparameter Ω_0.** (**Left**) For shallow networks ($L = 0$), large Ω_0 (> 5) leads to frequent model collapse and degraded performance, especially for large width. (**Middle**) Increasing the network depth to $L = 1$ markedly stabilizes learning for larger Ω_0, enabling agents to maintain high final reward. (**Right**) In PandaPickCube with deeper networks ($L = 4$), performance improves with increasing Ω_0 for wide agents with some signs of model collapse when $\Omega_0 > 5$. Together, these results suggest that while rich feature learning generally benefits RL, its stability depends on task, depth, and width.

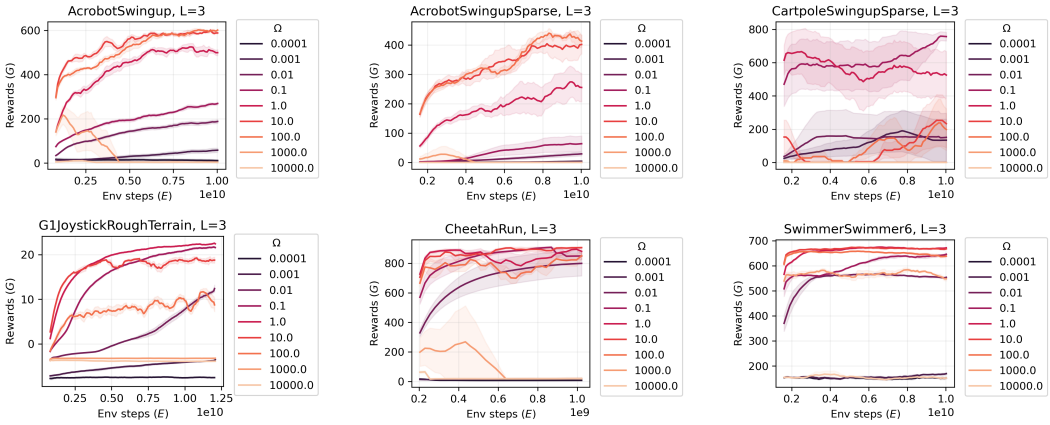


Figure 11: **RL performance across environment steps as a function of feature learning hyperparameter Ω_0.** Each panel shows the Ω_0 sweep for a different environment. Different environments vary in their range of "optimal" feature learning.

Prior work (Graldi et al., 2025) highlighted that rich feature learning, facilitated by large values of the richness hyperparameter Ω_0, may not always benefit continual learning; in fact, increased feature plasticity can accelerate catastrophic forgetting of useful structure. Motivated by these findings, we investigated whether promoting richer feature learning (by increasing Ω_0) improves or hinders reinforcement learning performance.

Figure 10 summarizes the relationship between Ω_0, network width, depth, and final agent reward on two tasks. When the network has minimal depth ($L = 0$), setting $\Omega_0 > 5$ leads to frequent model collapse and substantially reduced final performance, suggesting that unchecked feature plasticity can be harmful without sufficient network depth. Interestingly, even a single additional residual layer ($L = 1$) mitigates collapse, making training larger widths more robust to larger Ω_0 and leading to a performance plateau for $\Omega_0 \geq 1$. This behavior is consistent for the harder PandaPickCube task.

These observations lead to several key insights. First, richer feature learning is generally advantageous for RL agents on both simple and complex environments, provided the network has sufficient depth to absorb the increased feature evolution. Second, model collapse at shallow depths with large Ω_0 highlights the nuanced interaction between feature plasticity and representational stability; rigorous characterization of when and why this occurs is an important direction for future work. Third, the optimal degree of feature richness, as controlled by Ω_0, is highly dependent on both task and architecture: neither excessive rigidity nor excessive plasticity is universally optimal.

G REDUCING POLICY SHIFT CONSTRAINT FOR LEARNING

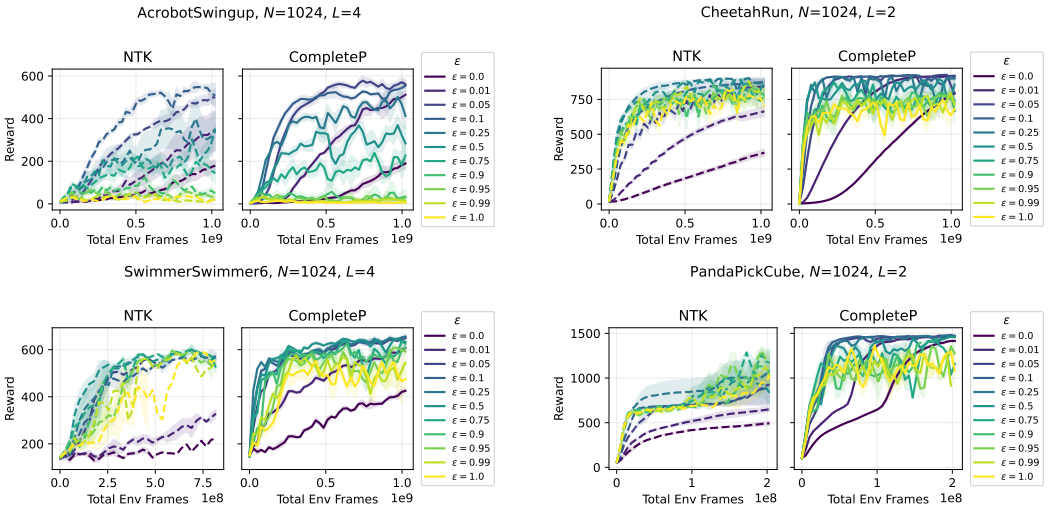


Figure 12: **RL performance as a function of the epsilon clipping hyperparameter ϵ.** Larger ϵ values allow a greater magnitude of change in model parameters, causing the policy to change to a greater extent after each gradient update.

Here we study the role that the clipping parameter in PPO plays in the process, and whether it changes our results regarding the presence of feature learning. One can view smaller values of clipping as a way to constrain the change in the policy between optimization steps. From observing 12 we argue that across both NTK and CompleteP agents the modulation of ε has roughly the same effect on both NTK and CompleteP performance.

H NON-STATIONARY DATA DISTRIBUTIONS IN REINFORCEMENT LEARNING

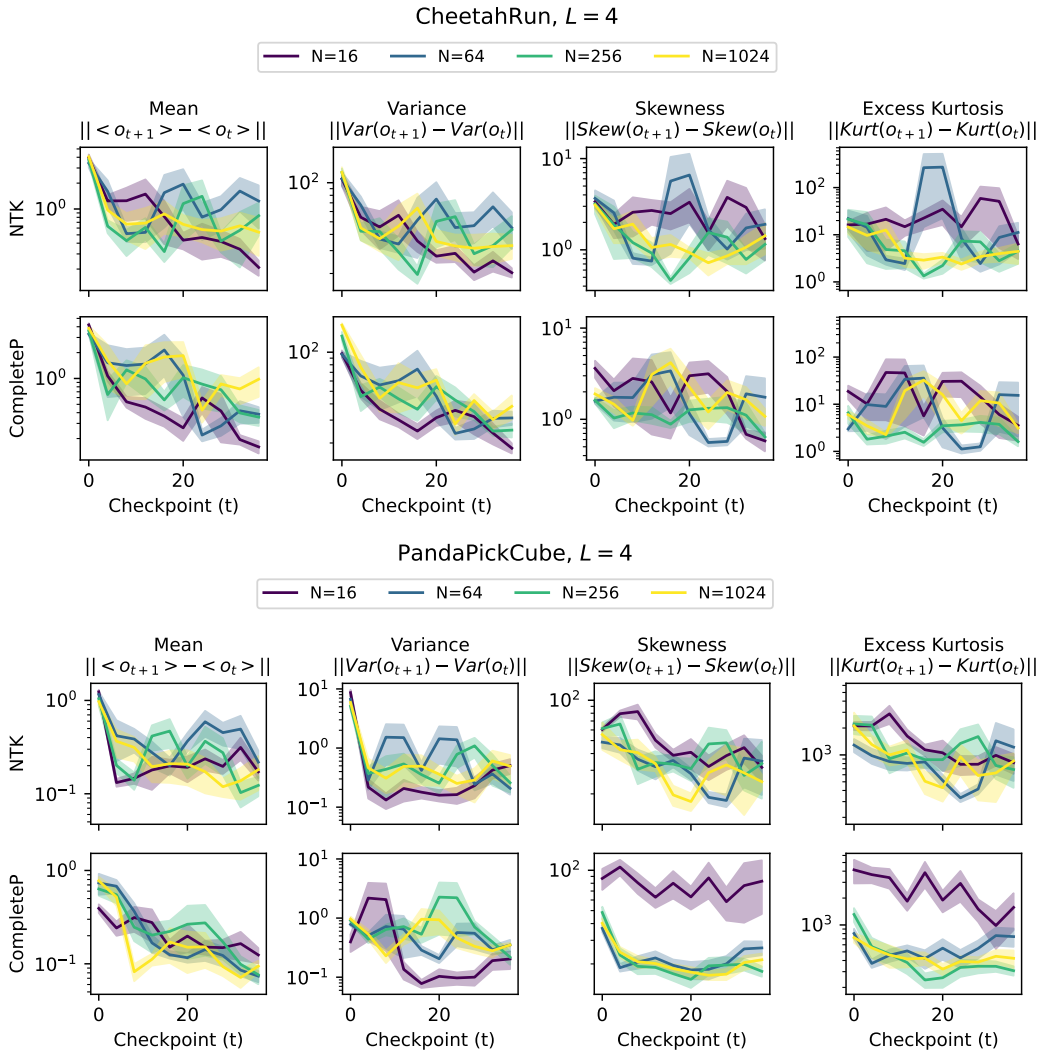


Figure 13: **Observation data distribution changes with policy evolution.** We plot the checkpoint-to-checkpoint difference in the first four moments (mean, variance, skewness, excess kurtosis) of the normalized observation data for two tasks and four model widths. Both NTK and CompleteP parameterizations show rapid changes early in training and stabilize as the policy converges. Larger models display slightly faster convergence toward a stable data distribution.

A central challenge unique to reinforcement learning, as compared to supervised learning, is the coupling between policy and environment. The policy not only determines the agent's actions, but directly influences what data trajectories are sampled from the environment. Every policy update therefore changes the future data distribution, which in turn affects future learning. This tight policy-environment loop leads to non-stationary—and initially unpredictable—data distributions throughout training, complicating both analysis and optimization.

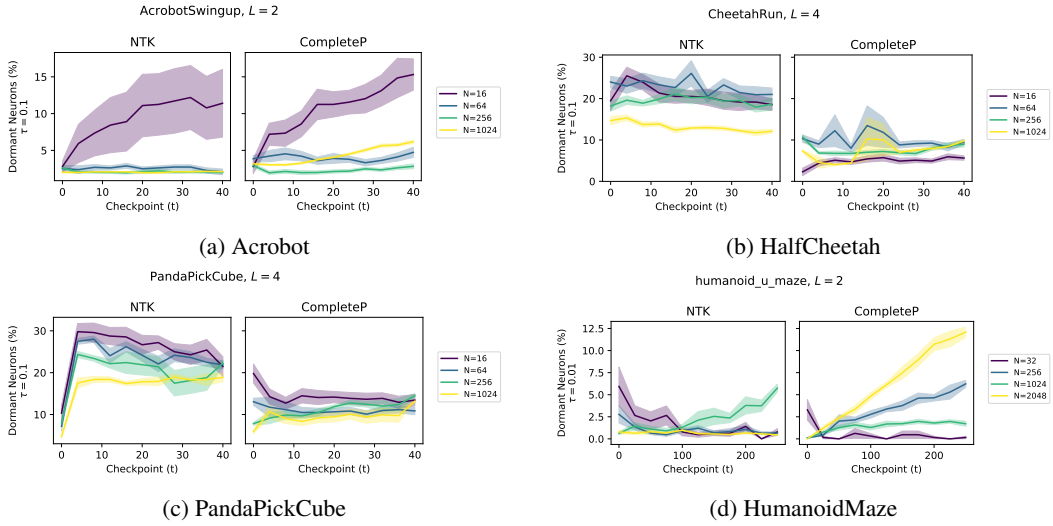
To characterize and visualize this effect, Figure 13 depicts the changes in the statistical moments of observed data distributions as training progresses. For each model checkpoint t, we roll out 30 trajectories and compute the first four moments (mean, variance, skewness, and excess kurtosis) of the observations collected. We then measure the change in each moment between consecutive checkpoints t and $t + 1$. This set of observations are not the synthetically generated dataset. This analysis allows us to quantify the rate and nature of data distribution shifts induced by policy updates.

The results show that, for both CheetahRun and PandaPickCube tasks, the rate of change in the mean and variance of the observation distribution is highest at the beginning of training, but gradually decreases over time. This trend reflects the learning process: initially, an untrained random policy generates highly diverse, failure-prone trajectories. As training proceeds and the policy improves, the distribution of sampled trajectories incorporates a growing fraction of successful behaviors. In later stages, as the agent approaches and maintains a near-optimal policy, the sampled data becomes more stereotyped—the mean and variance stabilize, indicating less dramatic distributional change.

The evolution of skewness and excess kurtosis provides additional insights. Large changes in these higher moments early in training suggest the distribution of observations is highly non-Gaussian with significant outliers, corresponding to rare events or failure states. As learning progresses, these statistics also generally decrease, indicating a reduction in the frequency and severity of such outliers and a move toward a more regular, lower-entropy observation distribution as the agent consistently succeeds.

Interestingly, this overall convergence trend is observed across all tested model widths, but appears somewhat faster for larger networks. This could indicate that wider models more rapidly converge on stable, successful policies, thus reducing the non-stationarity of the data distribution they induce at the later stages of learning.

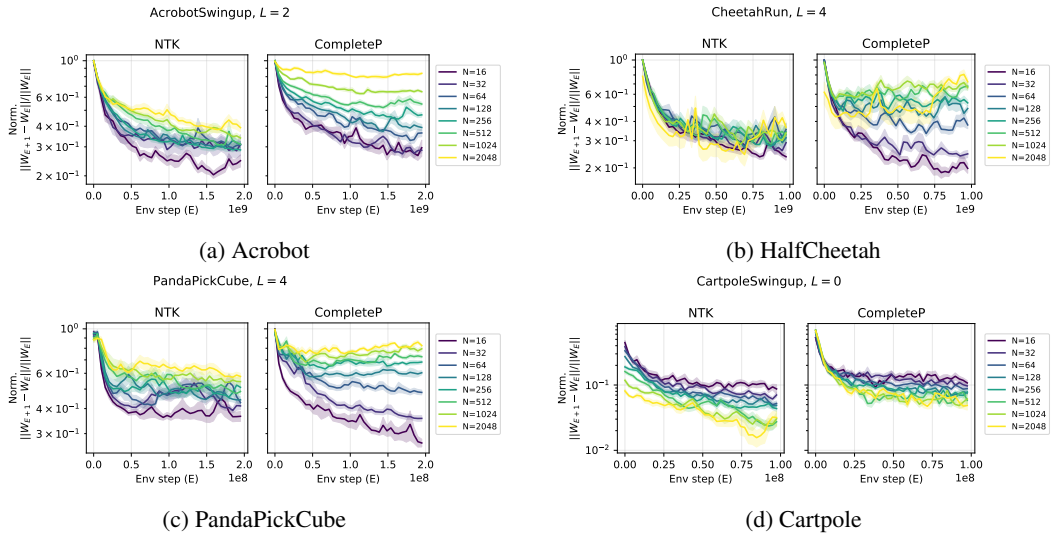
I Loss of Plasticity and Dormant Neurons in RL Agents



(a) Acrobot (b) HalfCheetah

(c) PandaPickCube (d) HumanoidMaze

Figure 14: **Dormant neurons increase in CompleteP.** Both NTK and CompleteP agents display fewer dormant neurons as width increases. However, note that larger width NTK agents have a reduced extent of feature learning and hence will be less prone to the dormant neurons phenomenon whereas feature learning is consistent in CompleteP agents.



(a) Acrobot (b) HalfCheetah

(c) PandaPickCube (d) Cartpole

Figure 15: **Weight change magnitude across learning.** In NTK, the weight change magnitude drops with width and across training, indicating loss of plasticity. By contrast, muP (CompleteP) yields stable weight dynamics for wider models, supporting ongoing adaptability.

A challenge in training neural networks, particularly in reinforcement and continual learning tasks, is the increase of activation sparsity over time. During learning, some neurons become ”dormant”: They fail to activate, with values shrinking toward zero and contributing neglible gradients during backpropagation (Sokar et al., 2023). This loss of plasticity can be particularly problematic in reinforcement learning, where continual adaptation to non-stationary data is required. When a significant proportion of neurons become dormant, the capacity of the model to adapt to new data distributions diminishes, often leading to learning collapse. Prior works have attempted to mitigate this effect through periodic reinitialization of dormant neurons (Dohare et al., 2024).

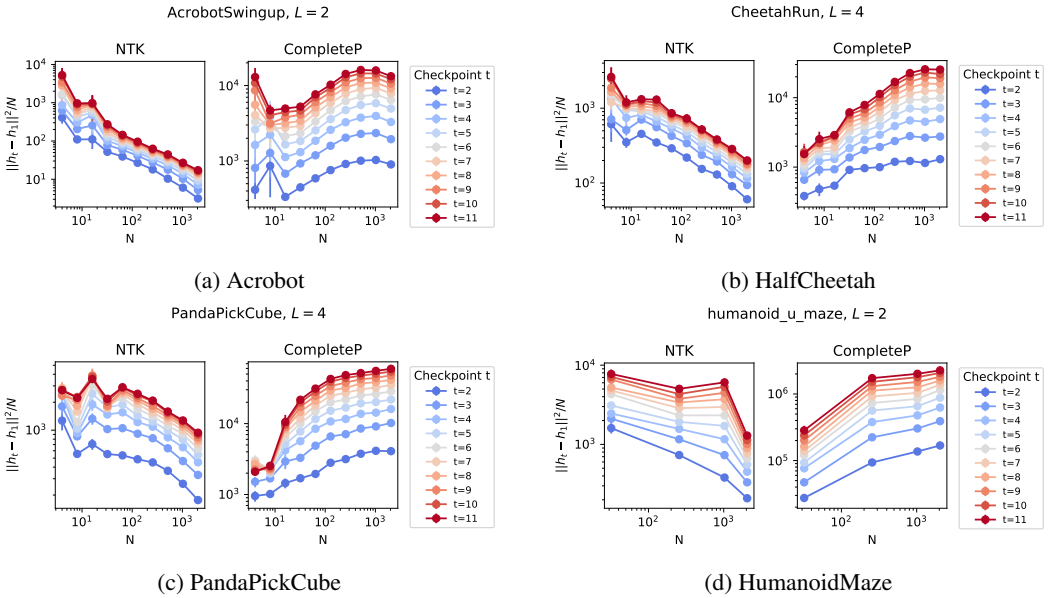
Here, we compare NTK and CompleteP parameterized agents in terms of the proportion of dormant neurons and the magnitude of weight changes throughout training. Figure 14 shows that CompleteP agents display a lower proportion of dormant neurons relative to NTK agents, particularly when the network width is sufficiently large ($N > 16$). Both parameterizations exhibit reduced dormancy as model width increases; however, in NTK, this arises because feature learning diminishes rapidly as width grows, effectively "freezing" representations and thus reducing exposure to the dormant neuron problem. By contrast, CompleteP agents preserve active, continually evolving features at $O(1)$ scale even for large widths. Refer to Appendix J for the rate of feature evolution for both NTK and CompleteP agents.

Weight change dynamics, depicted in Figure 15, further highlight differences. In NTK agents, the average magnitude of weight updates declines steadily during training and across width, indicating a sharp loss of plasticity with scale. For CompleteP, weight change magnitude decreases initially with width but plateaus for wide models, reflecting stable, consistent plasticity throughout learning. Larger CompleteP networks are thus better equipped to maintain adaptability, a property especially useful for continual learning and challenging RL environments (Graldi et al., 2025).

Together, these results suggest an underexplored benefit of wide, richly-parameterized RL agents: they may avoid catastrophic loss of plasticity, enabling both continual adaptation and retention of prior knowledge. This capacity could be crucial for scaling embodied control and lifelong learning systems far beyond current empirical limits.

J Coordinate Check for Feature and Logit Evolution

J.1 Feature coordinate checks



(a) Acrobot (b) HalfCheetah

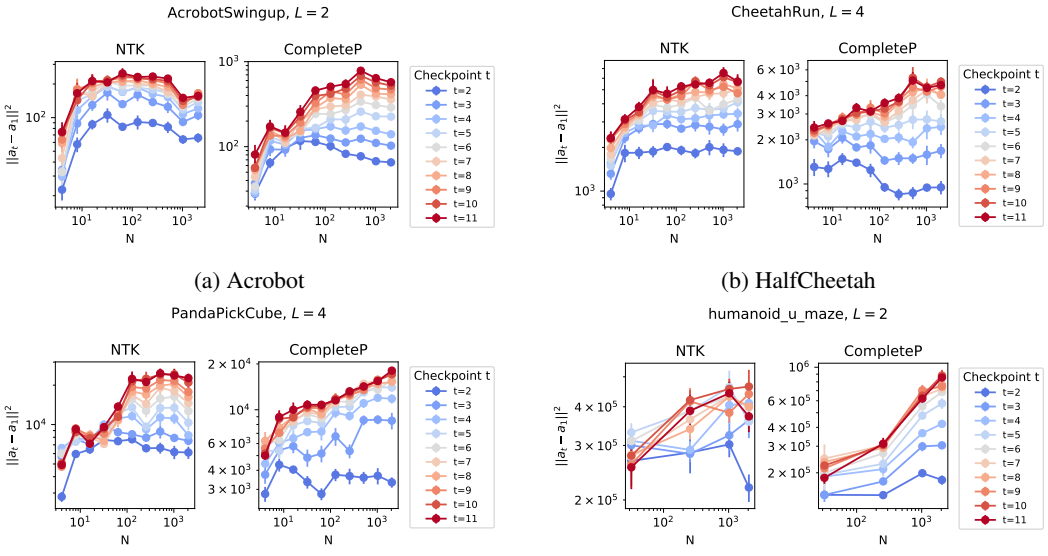
(c) PandaPickCube (d) HumanoidMaze

Figure 16: **Feature evolution.** NTK shows decreasing feature evolution with width scaling, confirming that feature learning diminishes in wide networks. CompleteP shows increasing feature evolution that plateaus for $N > 512$, supporting robust, width-invariant feature learning in large networks.

To empirically check that our NTK and CompleteP agents achieve the theoretical desiderata for network parameterization, especially concerning their feature and output function evolution across width scaling, we follow the methodology outlined in Yang et al. (2022); Dey et al. (2025). Feature evolution is quantified as the L2 norm $\|h_t - h_1\|$ with h measured on the synthetic dataset, where t denotes the training checkpoint and h_1 the feature vector after the first model checkpoint. This metric tracks how much the network's internal representations deviate from those at the start of training, and thus captures the degree of feature learning achieved (our desideratum 3).

As seen in Figure 16, NTK agents display a clear trend: feature evolution *decreases* as width increases, consistent with theory that NTK parameterization causes feature learning to diminish in wide networks. For CompleteP agents, we observe that feature evolution *increases* with width up to $N \approx 512$, after which it plateaus—suggesting $\mathcal{O}(1)$ feature change in the large width regime. This agrees with our desiderata for rich, width-invariant feature learning, but with one caveat: for smaller widths, the observed feature evolution is less consistent, possibly due to low reward signals associated with smaller network capacity agents that suppress effective gradient updates. As width grows, agents have the capacity to learn to maximize rewards and maintain evolving features, as predicted by theory and recent empirical work (Dey et al., 2025).

J.2 LOGIT COORDINATE CHECKS



(a) Acrobot (b) HalfCheetah

(c) PandaPickCube (d) humanoid_u_maze

Figure 17: **Logit evolution.** Both NTK and CompleteP agents achieve stable, width-invariant logit evolution for $N > 512$. Smaller widths show greater variability, likely due to reward-dependent gradient suppression.

Logit (function) evolution is measured as $\|a_t - a_1\|$, where a_t is the action (logit) output at checkpoint t, again evaluated on the synthetic dataset. Both NTK and CompleteP agents demonstrate *stable logit evolution* for sufficiently large width: as shown in Figure 17, function learning becomes width-invariant for $N > 512$, even as smaller widths display more variability.

Overall, we confirm that both NTK and CompleteP agents achieve desiderata 2 (stable function learning) and 3 (controllable feature learning) for sufficiently large widths. The instability in feature and function evolution at small widths likely arises from the reward-dependent nature of policy gradients, where low rewards limit effective weight (and thus feature) updates.

K FEATURE CONSISTENCY WITH WIDTH SCALING

CompleteP parameterization robustly improves learning performance and plasticity at scale. In this section, we analyze how feature learning and representational consistency evolve with network width, focusing on both the structure and alignment of learned feature kernels.

Figure 18 visualizes the feature kernels (the similarity matrix $HH^\top$ of penultimate-layer activations) at initialization and after training under NTK and CompleteP parameterizations, for varying network widths in AcrobotSwingup, CheetahRun, and PandaPickCube. At initialization, kernels are highly similar and preactivation densities $\rho(h^L)$ are consistent across widths, verifying stable signal propagation (desideratum 1). After training, NTK kernels increasingly resemble their initial state and grow more diffuse, while CompleteP retains consistent kernel structure and sharper, low-dimensional features for larger widths ($N > 256$).

Figure 19 shows the cumulative variance explained by the top five eigenvectors of the feature kernel, highlighting feature dimensionality. As width increases, initial feature spectra decays rapidly, indicating increasingly high-dimensional, disordered features at scale. NTK shows significant spectra decay in simpler control tasks i.e. Acrobot but the decay slows down as the task becomes progressively harder. In contrast, CompleteP exhibits stable, strongly concentrated spectra, indicating the emergence of low-dimensional, task-relevant representations even as width grows.

Figure 21 examines feature kernel alignment across seeds using CKA similarity. Kernel alignment between initial and NTK increases rapidly with width, suggesting that NTK kernels become more similar to random kernels as width grows. In comparison, CompleteP alignment increases more slowly, reflecting the preservation of task-relevant diversity and less convergence to randomness.

Figure 22 shows direct visualizations of feature kernel variability across seeds. Both NTK and muP show inconsistent features and action logits with different seeds, but CompleteP exhibits greater CKA similarity, supporting improved robustness and representation consistency, especially at width N 256.

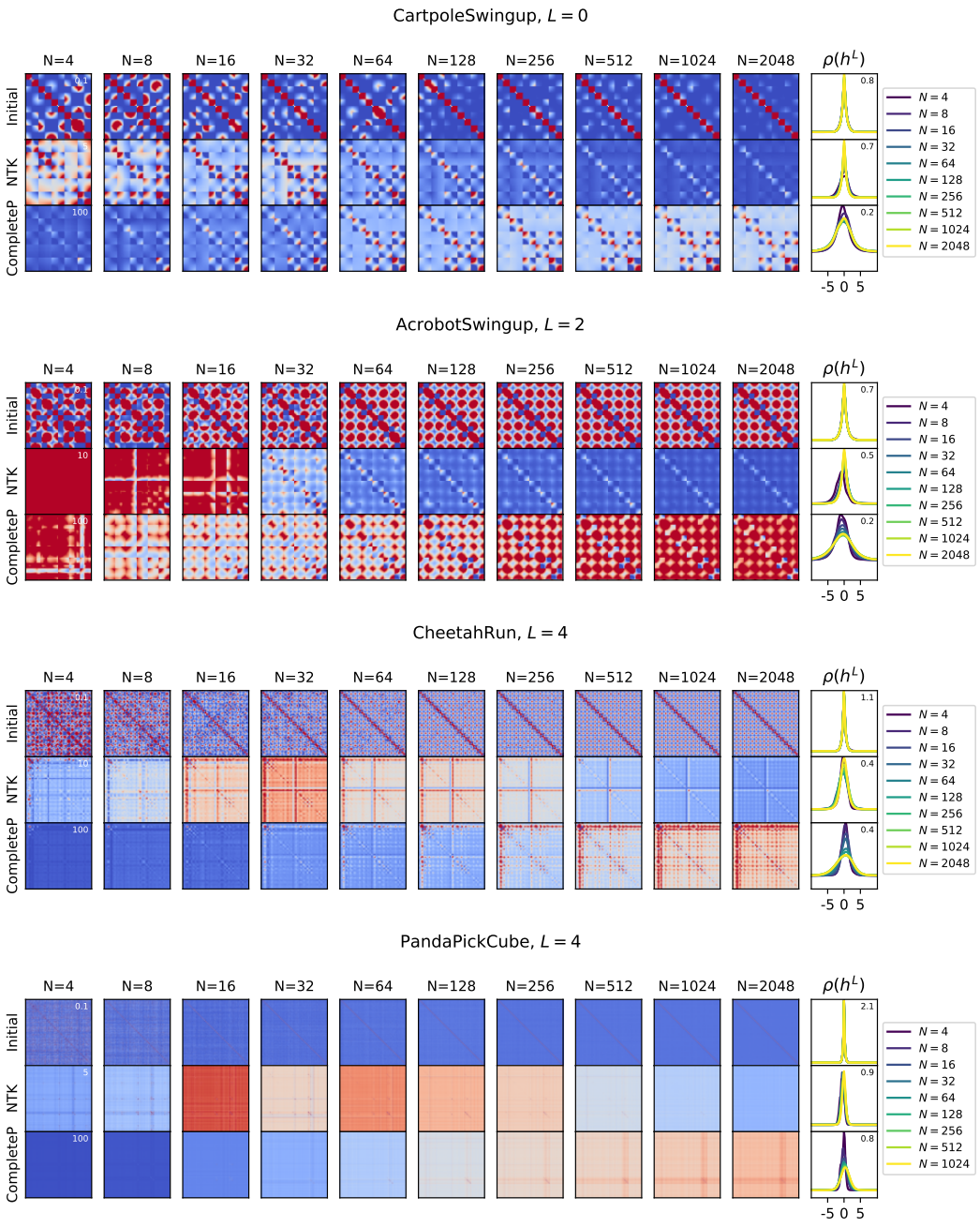
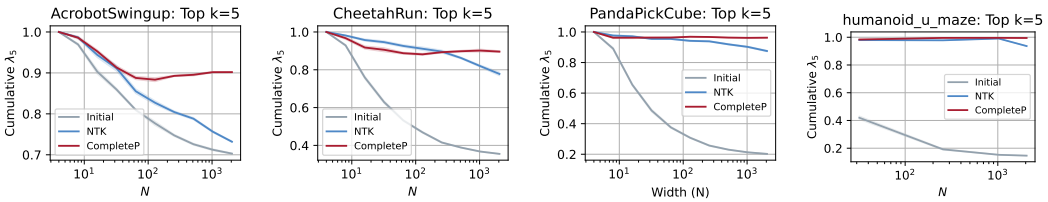
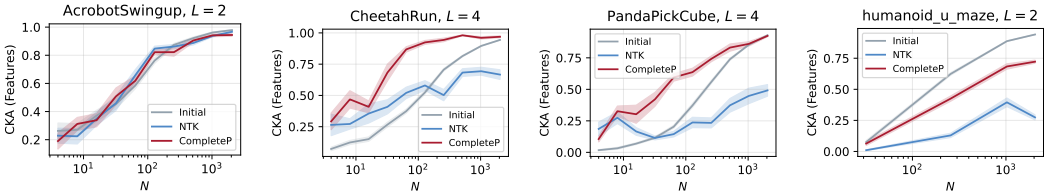


Figure 18: **Consistent features across width.** At initialization, both NTK and CompleteP show similar feature kernels and preactivation densities. After training, NTK feature kernels become increasingly diffuse and similar to the initial state, indicating reduced feature learning at large widths. CompleteP retains consistent and structured kernels with sufficiently large widths ($N > 256$), demonstrating $\Theta(1)$ feature learning.



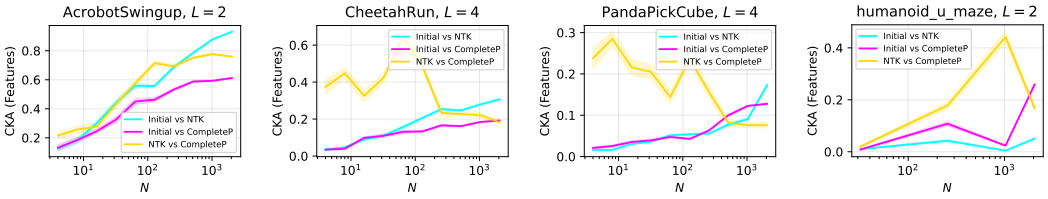
(a) Acrobot (b) HalfCheetah (c) PandaPickCube (d) HumanoidMaze

Figure 19: **CompleteP learns low-dimensional features with width scaling.** For random initializations and NTK features, the cumulative variance explained by the top 5 eigenvalues decays with width, indicating increasingly diffuse features. However, NTK's spectral decay seems to be task dependent, where the decay is slower for harder tasks. By contrast, CompleteP preserves concentrated, low-dimensional structure across widths, supporting robust representation learning.



(a) Acrobot (b) HalfCheetah (c) PandaPickCube (d) HumanoidMaze

Figure 20: **Seed specific kernel alignment increases more rapidly for CompleteP.** CKA similarity for CompleteP between different seed initializations rises quickly with width, indicating CompleteP kernels converge toward similar structured states.



(a) Acrobot (b) HalfCheetah (c) PandaPickCube (d) HumanoidMaze

Figure 21: **Kernel alignment increases more rapidly for NTK with initial features.** CKA similarity between initial and NTK kernels rises quickly with width, indicating NTK kernels converge toward the structureless random initial state. For CompleteP, alignment rises more slowly, indicating greater diversity and stability of learned features across seeds and widths. However, this rate of increase seems to be task dependent.

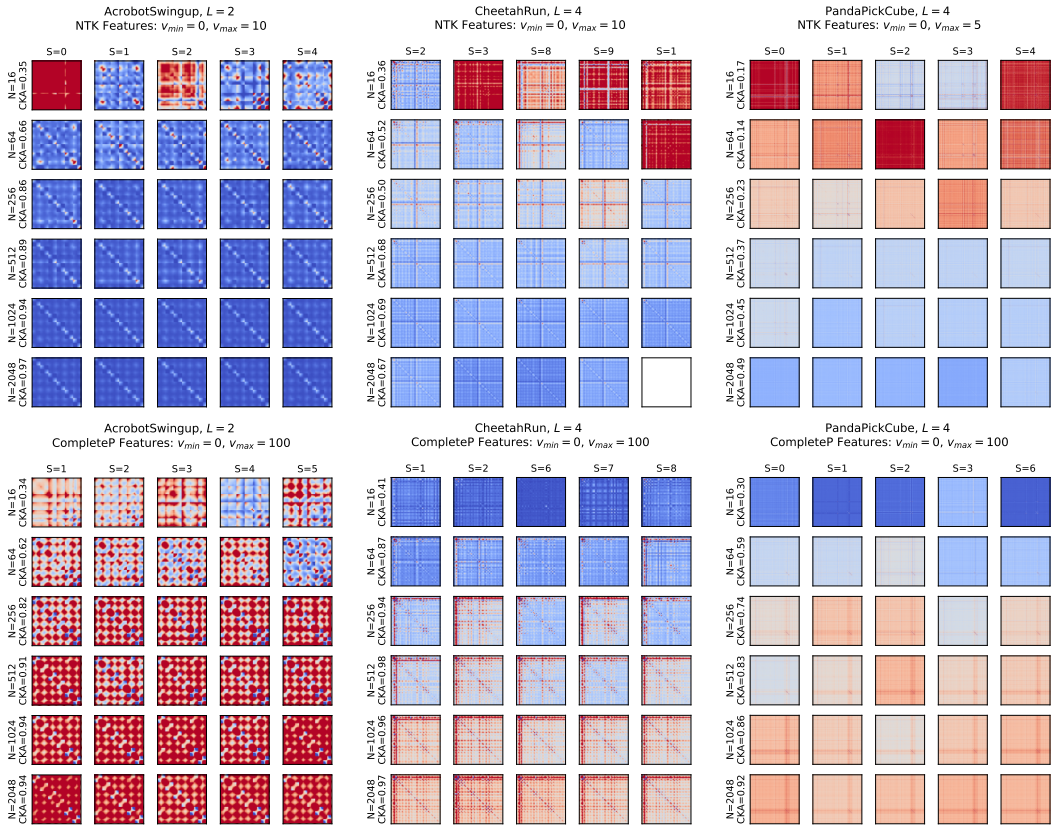


Figure 22: **Feature consistency improves across seeds with width scaling.** Both NTK and muP show inconsistent action logits and features across seeds, but muP exhibits slightly higher kernel alignment, indicating enhanced representation robustness across initializations.

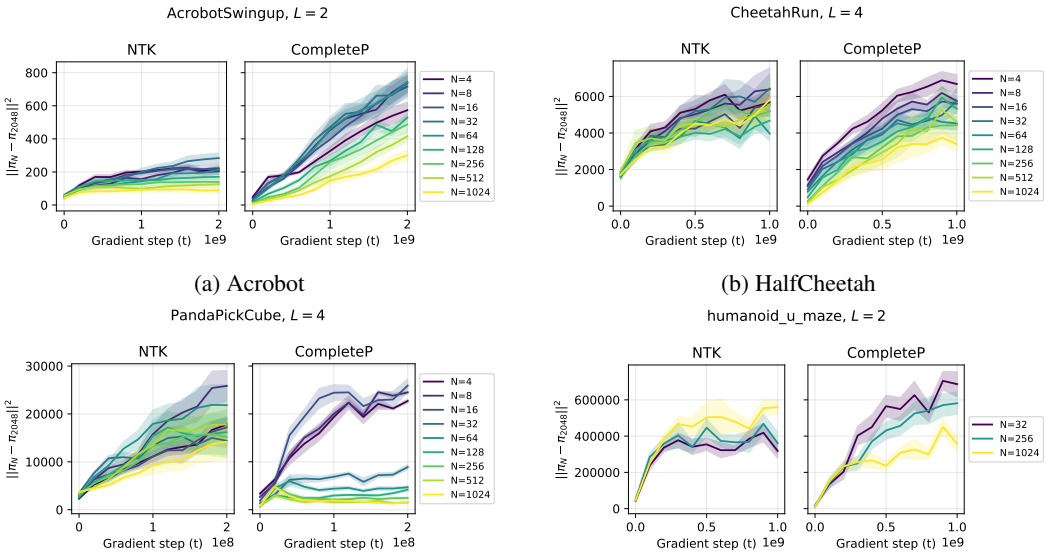
L Policy Consistency with Width Scaling

Network parameterization also affects the consistency and reproducibility of learned policies. Here, we analyze how the action logits and policy kernels evolve with width scaling and seed variation.

L.1 Consistency with Width Scaling

Figure 23 shows the comparison of action logits between models of different widths for NTK and CompleteP agents across multiple environments. For each setting, the largest width ($N = 2048$) is used as a reference; logit mean squared error is measured between each smaller width model and the largest over the course of learning.

Both NTK and CompleteP agents demonstrate increasing logit alignment across widths as models get wider, indicating consistency in policy learning, especially in the high-capacity regime. Averaged logit kernels are consistent across widths for both parameterizations after learning, which is expected since the NTK parameterization places no restriction on output evolution (see desideratum 2). In smaller networks, divergence is higher, aligning with their reduced performance. CompleteP models not only match NTK but often show even faster width-dependent convergence of logits, likely a result of their more robust feature learning.



(a) Acrobot (b) HalfCheetah

(c) PandaPickCube (d) HumanoidMaze

Figure 23: **Consistent logit evolution across width.** NTK and CompleteP agents both show increasing policy consistency across width, with logits converging toward those of the largest agent. Policy outputs for narrow networks are more variable due to limited capacity, while wide networks converge toward similar control strategies.

L.2 Inconsistency Across Seeds

Despite improved consistency with width scaling, stochasticity inherent to RL (e.g., exploration noise, environment transitions) can introduce variability in learned policies across seeds. **Figure 24** examines the kernel structure of action logits for NTK and muP agents trained with different random seeds.

While both parameterizations show some variation across seeds, CompleteP demonstrates notably higher CKA (centered kernel alignment) similarity compared to NTK, suggesting improved reproducibility of learned policies even in the presence of RL stochasticity. In contrast, NTK policies are more susceptible to seed-dependent variations, consistent with reduced feature plasticity at high width.

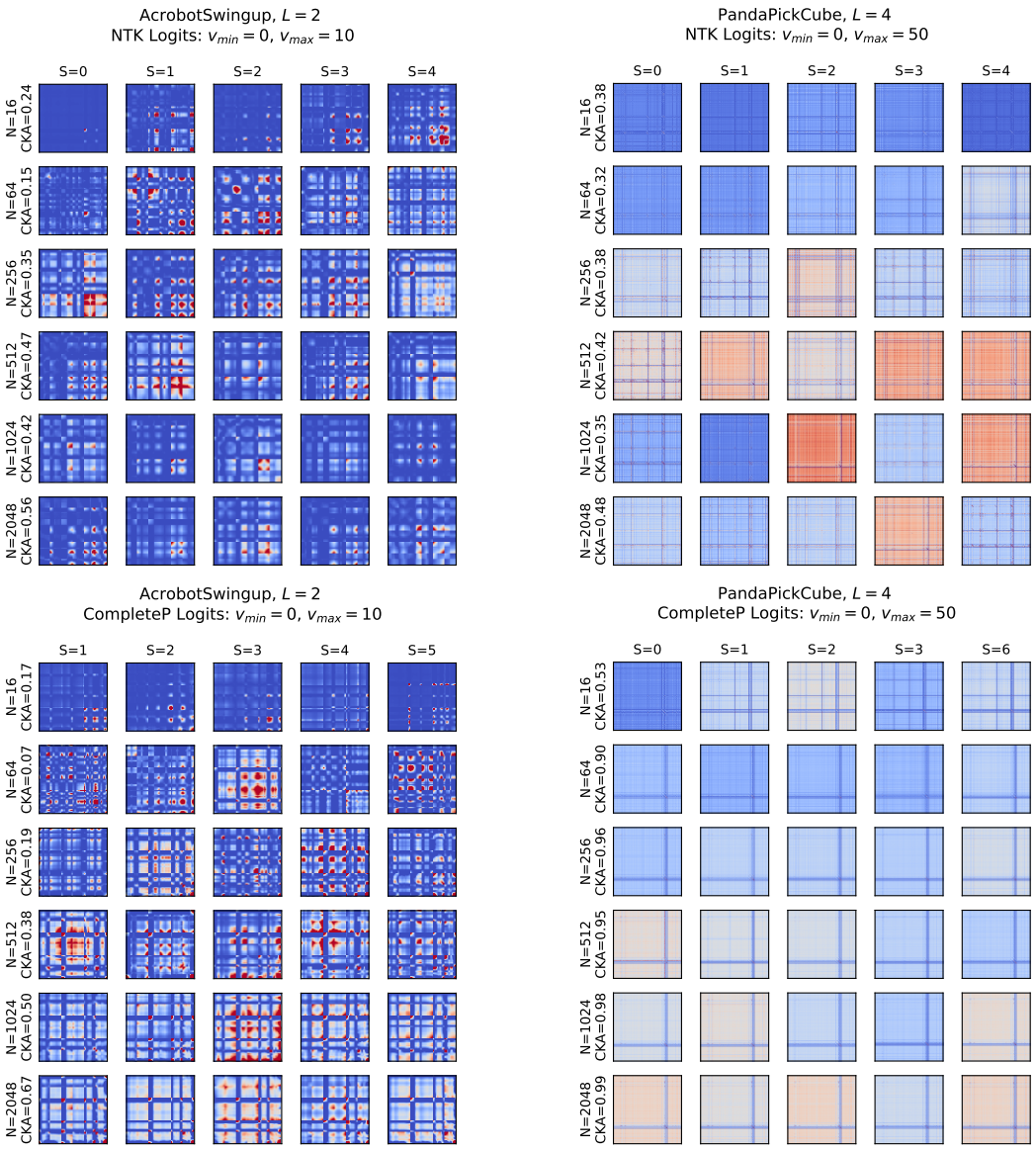
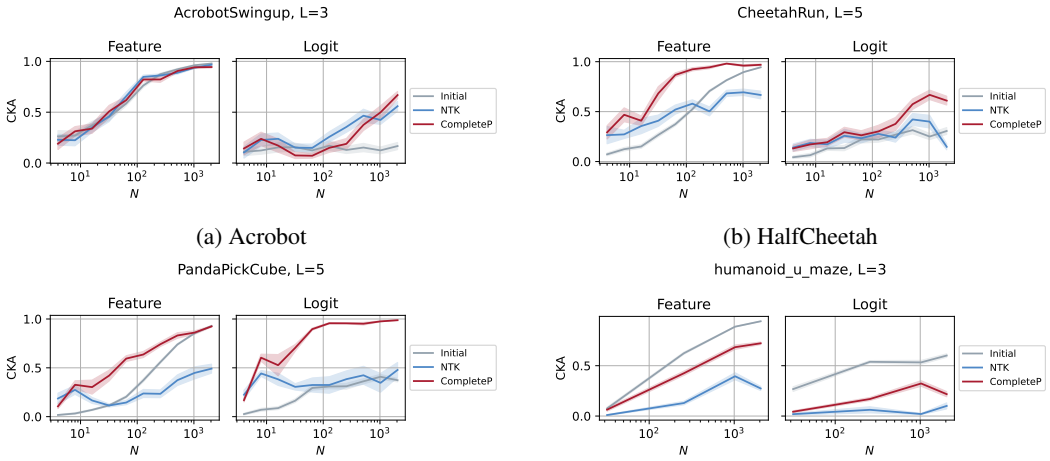


Figure 24: **CompleteP achieves higher logit consistency acros seeds with width scaling.** Both NTK and muP agents show some inconsistency in action logits across seeds, but muP exhibits notably higher kernel alignment (CKA), supporting improved reproducibility of the learned policy as width increases.

M FEATURES AND POLICIES CONVERGE AT DIFFERENT RATES



(a) Acrobot

(b) HalfCheetah

(c) HalfCheetah

(d) HumanoidMaze

Figure 25: **Logit kernels for individual seeds converge slower than feature kernels with width scaling suggesting possible divergence between feature and policy learning.**

N ADDITIONAL LEARNING CURVES

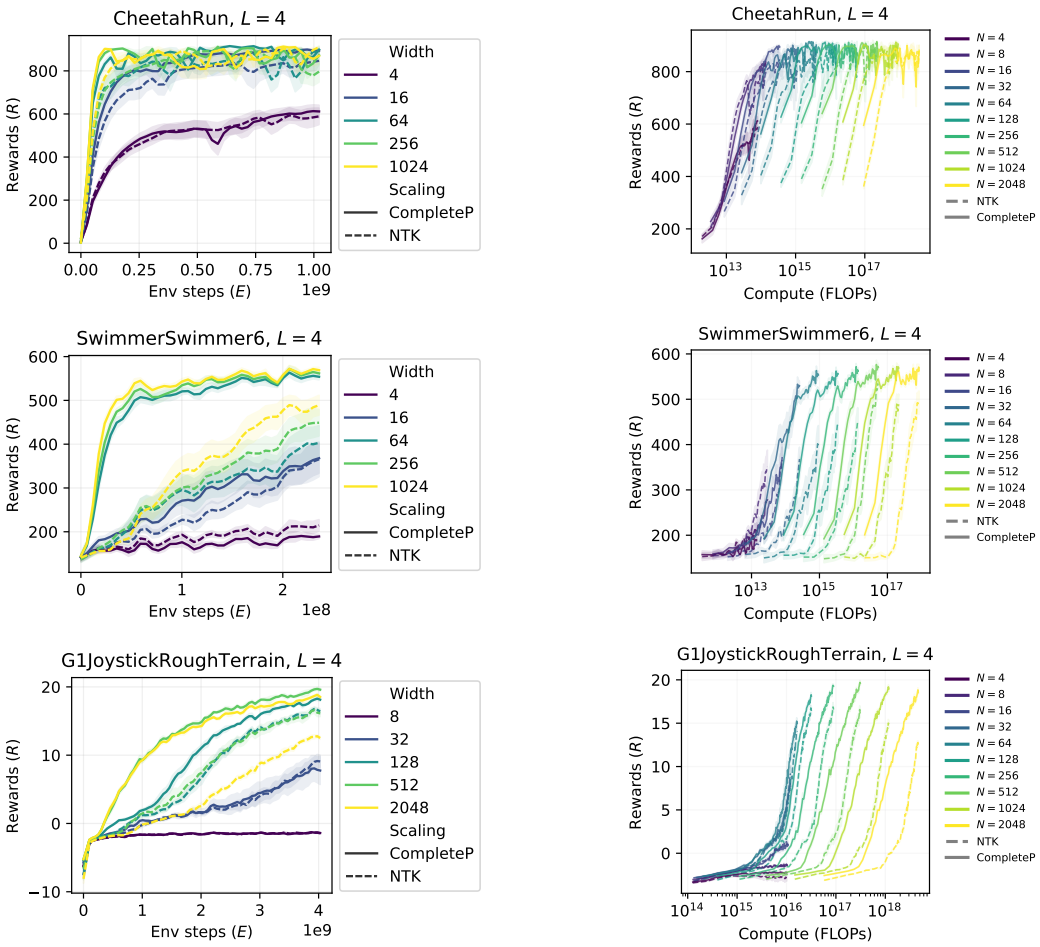


Figure 26: **Summary of reward learning curves, compute usage, and Pareto frontiers for three hard control tasks.**

N.1 DEPTH SCALING

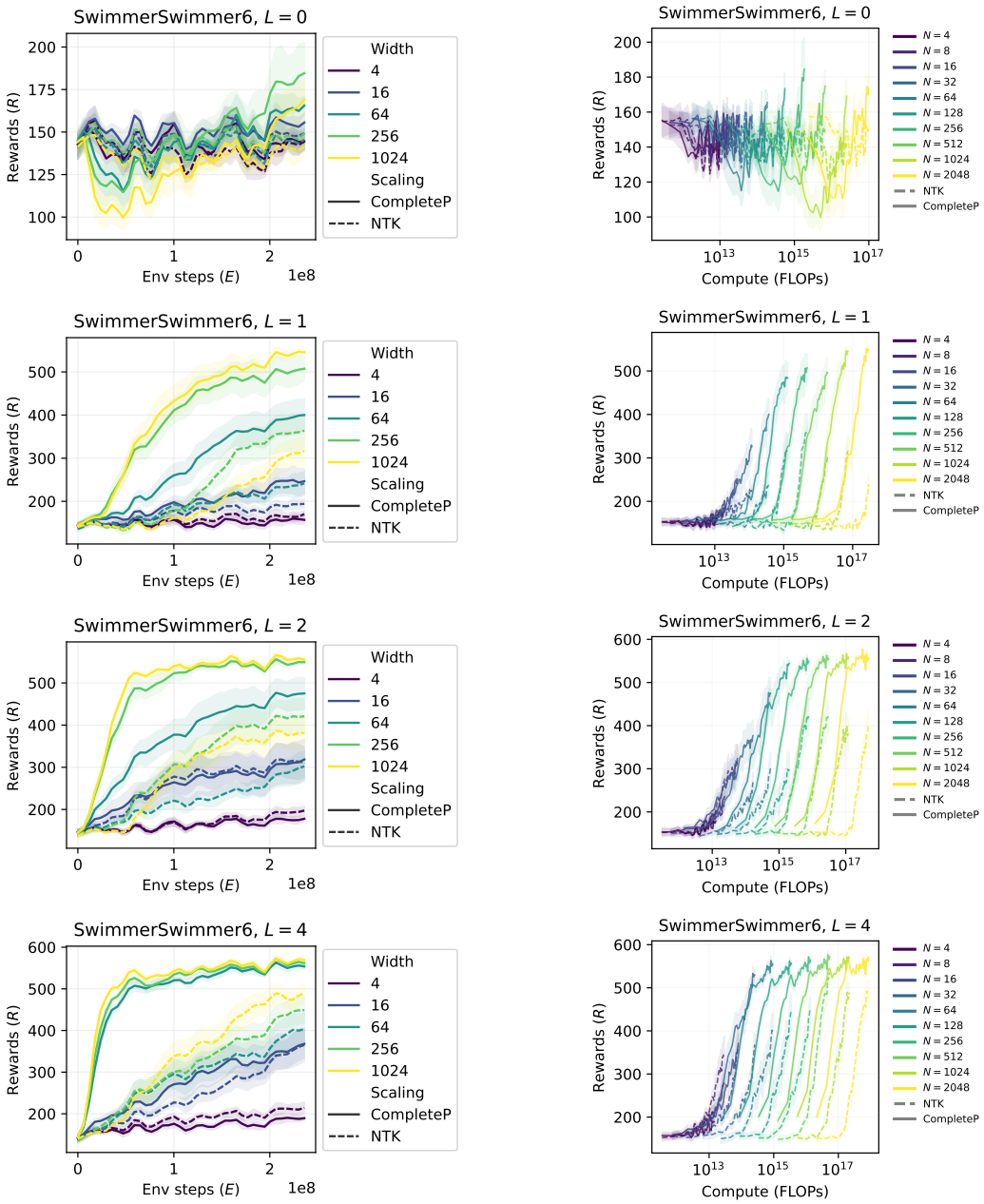


Figure 27: **Depth scaling improves learning performance in both CompleteP and NTK agents on Swimmer environment.**

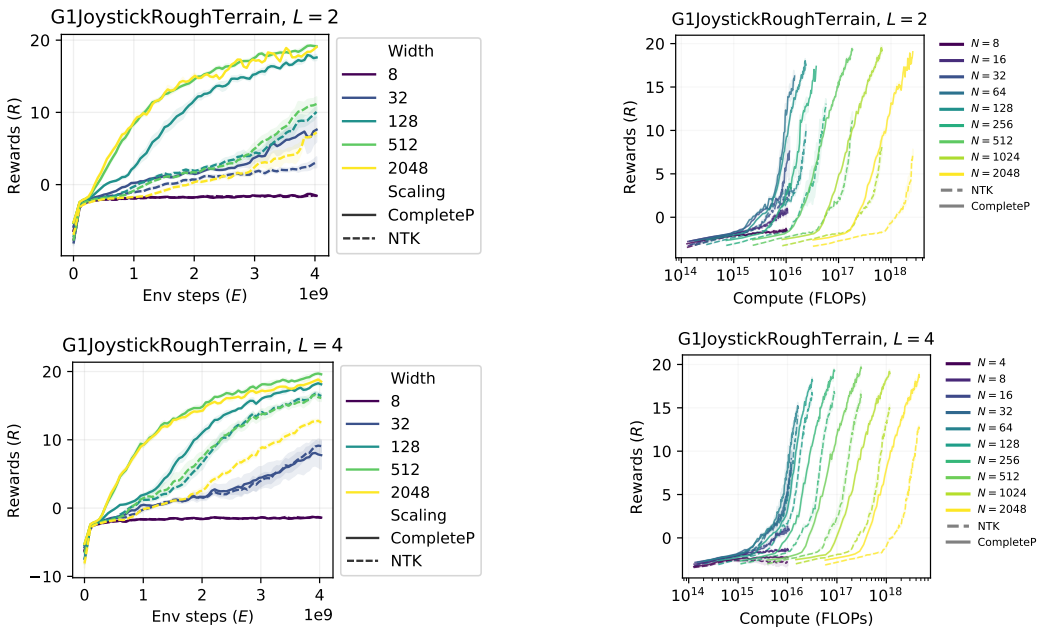
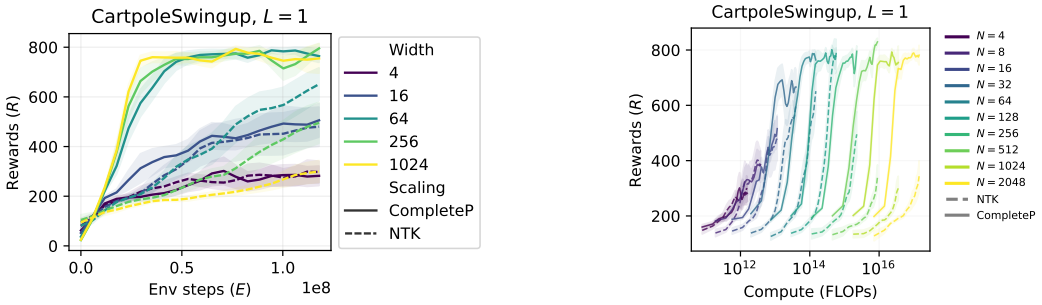
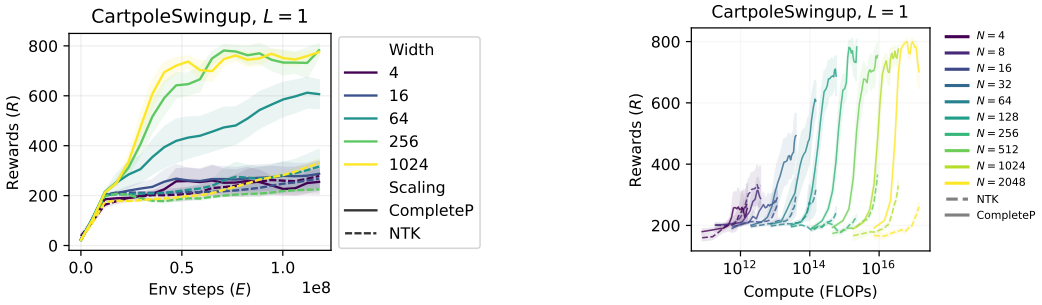


Figure 28: **Depth scaling improves learning performance in both CompleteP and NTK agents on G1Rough environment.**

N.2 ORTHOGONAL WEIGHT INITIALIZATION

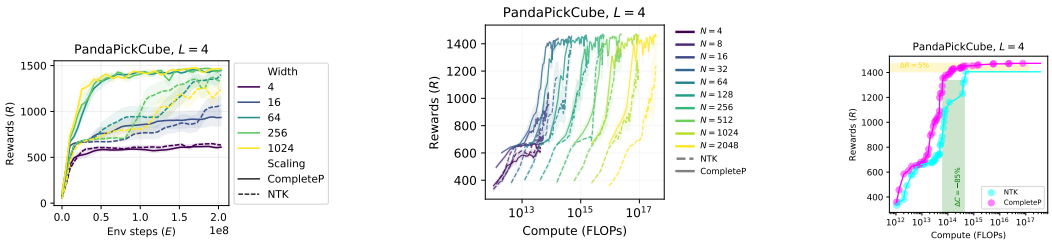


(a) Gaussian weight initialization

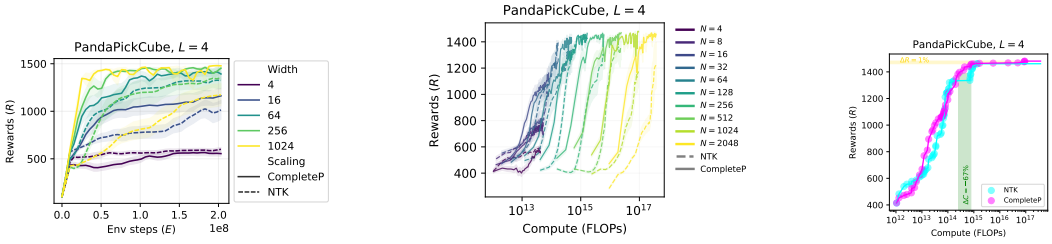


(b) Orthogonal weight initialization

Figure 29: **Cartpole and orthogonal initialization.** Each row shows reward curves, compute, and Pareto frontier for (top) Gaussian and (bottom) Orthogonal initialization. We note that

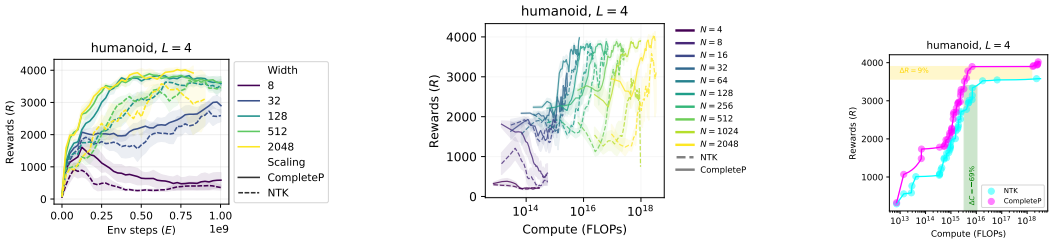


(a) Gaussian initialization

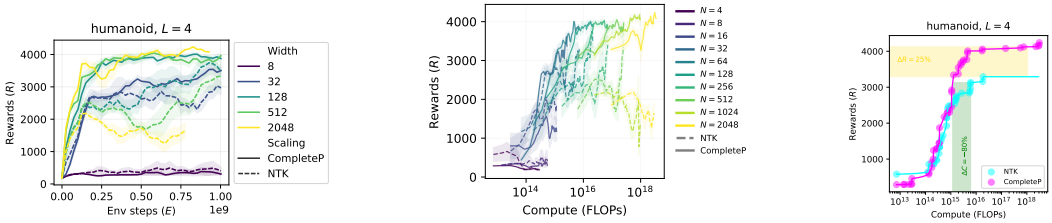


(b) Orthogonal initialization

Figure 30: **PandaPickCube and orthogonal initialization.** Each row: reward-vs-steps, reward-vs-FLOPs, Pareto frontier for (top) Gaussian and (bottom) Orthogonal weight initialization.

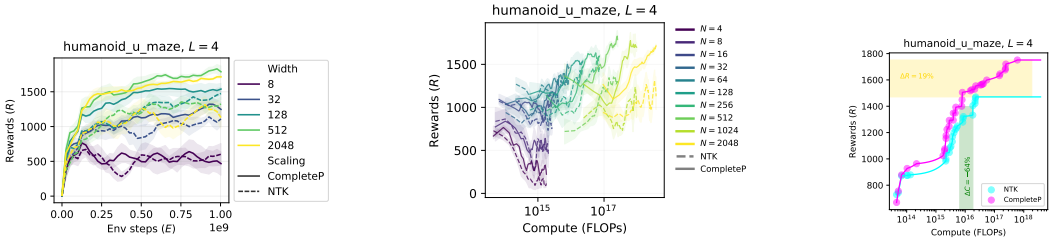


(a) Gaussian initialization

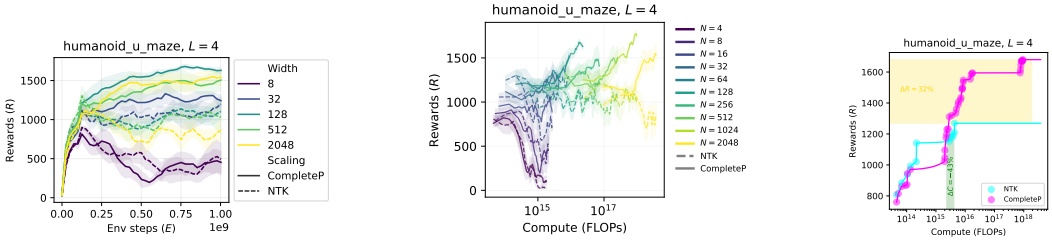


(b) Orthogonal initialization

Figure 31: **Humanoid and orthogonal initialization.** Each row: reward-vs-steps, reward-vs-FLOPs, Pareto frontier for (top) Gaussian and (bottom) Orthogonal weight initialization.



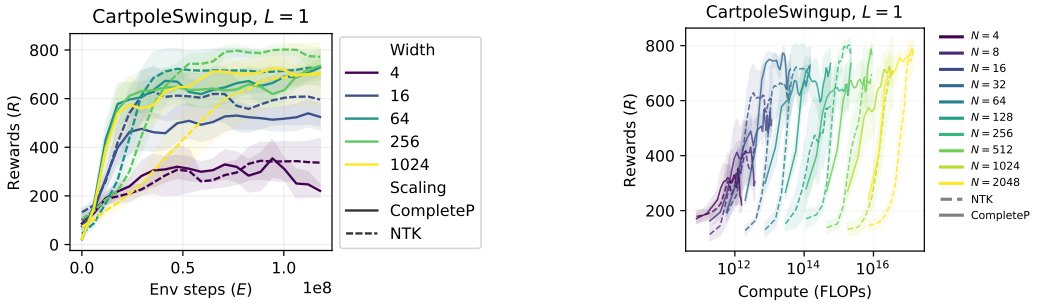
(a) Gaussian initialization



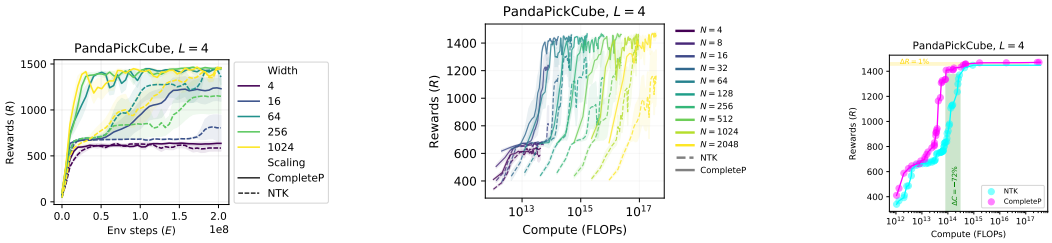
(b) Orthogonal initialization

Figure 32: **HumanoidMaze and Orthogonal initialization.** Each row: reward-vs-steps, reward-vs-FLOPs, Pareto frontier for (top) Gaussian and (bottom) Orthogonal weight initialization.

N.3 LAYER NORMALIZATION



(a) CartpoleSwingup with layer normalization



(b) PandaPickCube with layer normalization

Figure 33: **Adding Layer Normalization into each residual layer.** Each row: reward-vs-steps, reward-vs-FLOPs, Pareto frontier for (top) Cartpole and (bottom) PandaPickCube.

N.4 SPARSE REWARDS WITH SCALING

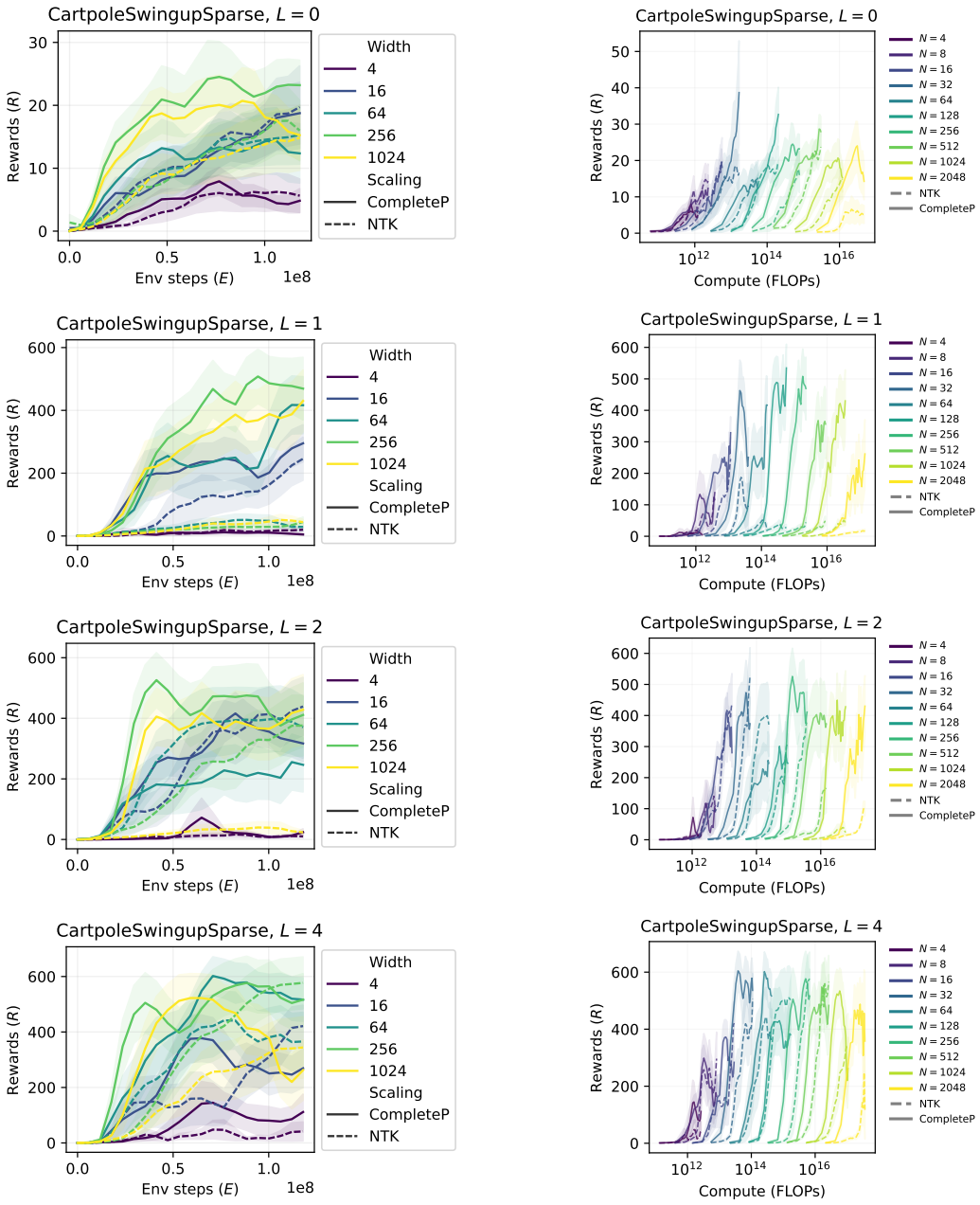
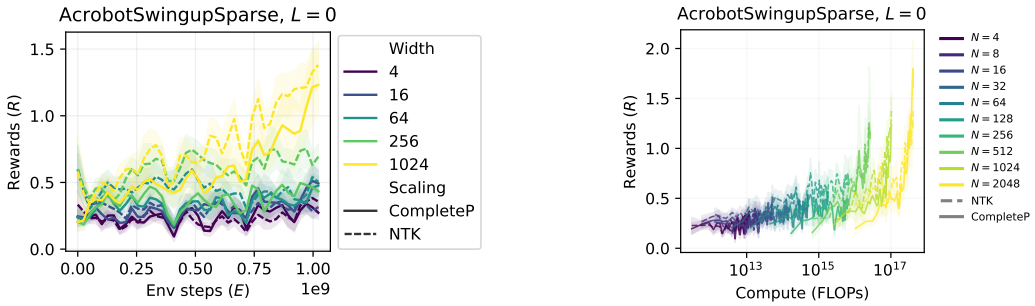
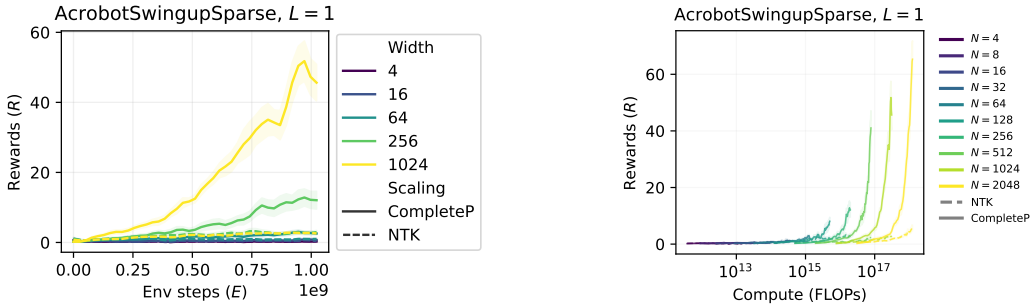


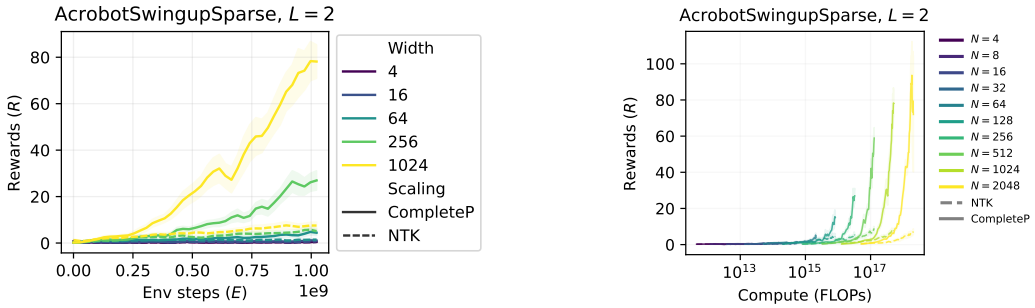
Figure 34: **CompleteP demonstrates improved learning with width and depth scaling in CartpoleSwingup with sparse rewards.** Each row indicates models with 0,1,2 or 4 residual layers.



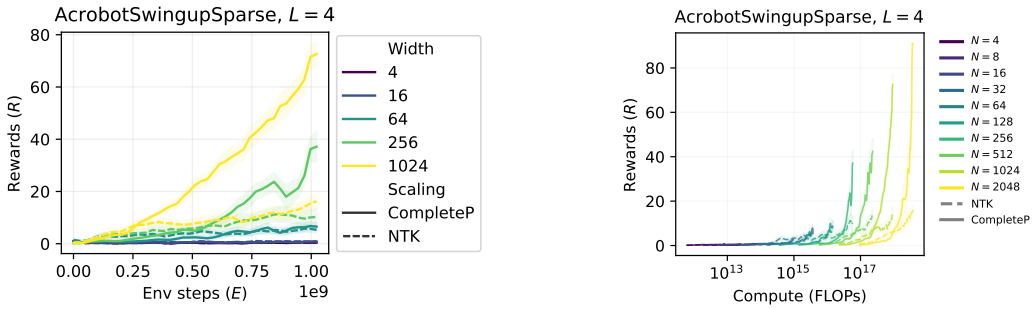
(a) Acrobot with no residual layers



(b) Acrobot with 1 residual layer



(c) Acrobot with 2 residual layers



(d) Acrobot with 4 residual layers

Figure 35: **CompleteP demonstrates improved learning with width and depth scaling in AcrobotSwingup with sparse rewards.** Each row indicates models with 0,1,2 or 4 residual layers.

O REWARD AND COMPUTE EFFICIENCY TABLES

Environment	Variant	Depth (L)	Reference Reward	NTK (FLOPs)	μP (FLOPs)	$\Delta\%$
CartpoleSwingup	-	0,1	626	2.09e14	3.04e13	-61.5
CartpoleSwingup	Ortho	0,1	445	3.54e14	5.81e13	-49.3
CartpoleSwingup	LN	0,1	622	1.00e15	5.19e12	-93.5
CartpoleSwingupSparse	Sparse	0,1,2,4	337	1.26e15	2.01e13	-58.0
PandaPickCube	-	4	1364	3.80e14	6.29e13	-83.5
PandaPickCube	Ortho	4	1380	5.18e14	2.43e14	-53.0
PandaPickCube	LN	4	1385	3.89e14	5.11e13	-86.9
CheetahRun	-	0,1,2,4	863	2.71e14	1.27e14	-52.4
SwimmerSwimmer6	-	0,1,2,4	395	1.14e16	3.02e14	-88.8
AcrobotSwingup	-	0,1,2,4	142	4.04e17	2.48e16	-74.9
AcrobotSwingupSparse	Sparse	0,1,2,4	30	7.30e17	1.47e17	-68.0
HumanoidGoToTarget	-	1,4	3704	5.67e15	2.94e15	-45.2
HumanoidGoToTarget	Ortho	4	4186	1.75e18	1.02e16	-99.4
HumanoidUMaze	-	1,4	1564	1.73e16	7.73e15	-60.7
HumanoidUMaze	Ortho	4	1397	7.20e15	2.51e15	-65.2
G1JoystickRoughTerrain	-	4	17	2.93e16	2.72e16	-7.1
Average			1154	1.85e17	1.39e16	-59.3

Table 9: Compute needed for various control environments and parameterizations. Ortho: Orthogonal weight init. LN: LayerNorm residual. Sparse: sparse reward.

Environment	Variant	Depth (L)	Reference FLOPs	NTK (Rewards)	μP (Rewards)	$\Delta\%$
CartpoleSwingup	-	0,1	3.36e16	659	786	22.8
CartpoleSwingup	Ortho	0,1	3.36e16	477	577	15.9
CartpoleSwingup	LN	0,1	3.36e16	649	778	23.4
CartpoleSwingupSparse	Sparse	0,1,2,4	6.45e16	354	442	33.1
PandaPickCube	-	4	1.30e17	1436	1477	2.8
PandaPickCube	Ortho	4	1.30e17	1453	1482	2.0
PandaPickCube	LN	4	1.30e17	1458	1474	1.1
CheetahRun	-	0,1,2,4	6.54e17	909	917	0.9
SwimmerSwimmer6	-	0,1,2,4	1.51e17	416	473	13.7
AcrobotSwingup	-	0,1,2,4	6.52e17	148	208	37.2
AcrobotSwingupSparse	Sparse	0,1,2,4	6.52e17	29	54	63.3
HumanoidGoToTarget	-	1,4	1.09e18	3899	4122	5.8
HumanoidGoToTarget	Ortho	4	1.61e18	4166	4261	2.3
HumanoidUMaze	-	1,4	1.27e18	1646	1874	13.8
HumanoidUMaze	Ortho	4	1.88e18	1470	1800	22.5
G1JoystickRoughTerrain	-	4	2.32e18	18	20	8.4
Average			6.78e17	1199	1297	16.8

Table 10: Reward attained for various control environments and parameterizations. Depth shows the L values used. Reference column shows reference compute required (FLOPs). Ortho: Orthogonal weight init. LN: LayerNorm residual. Sparse: sparse reward.

P FEATURE AND LOGIT DYNAMICS

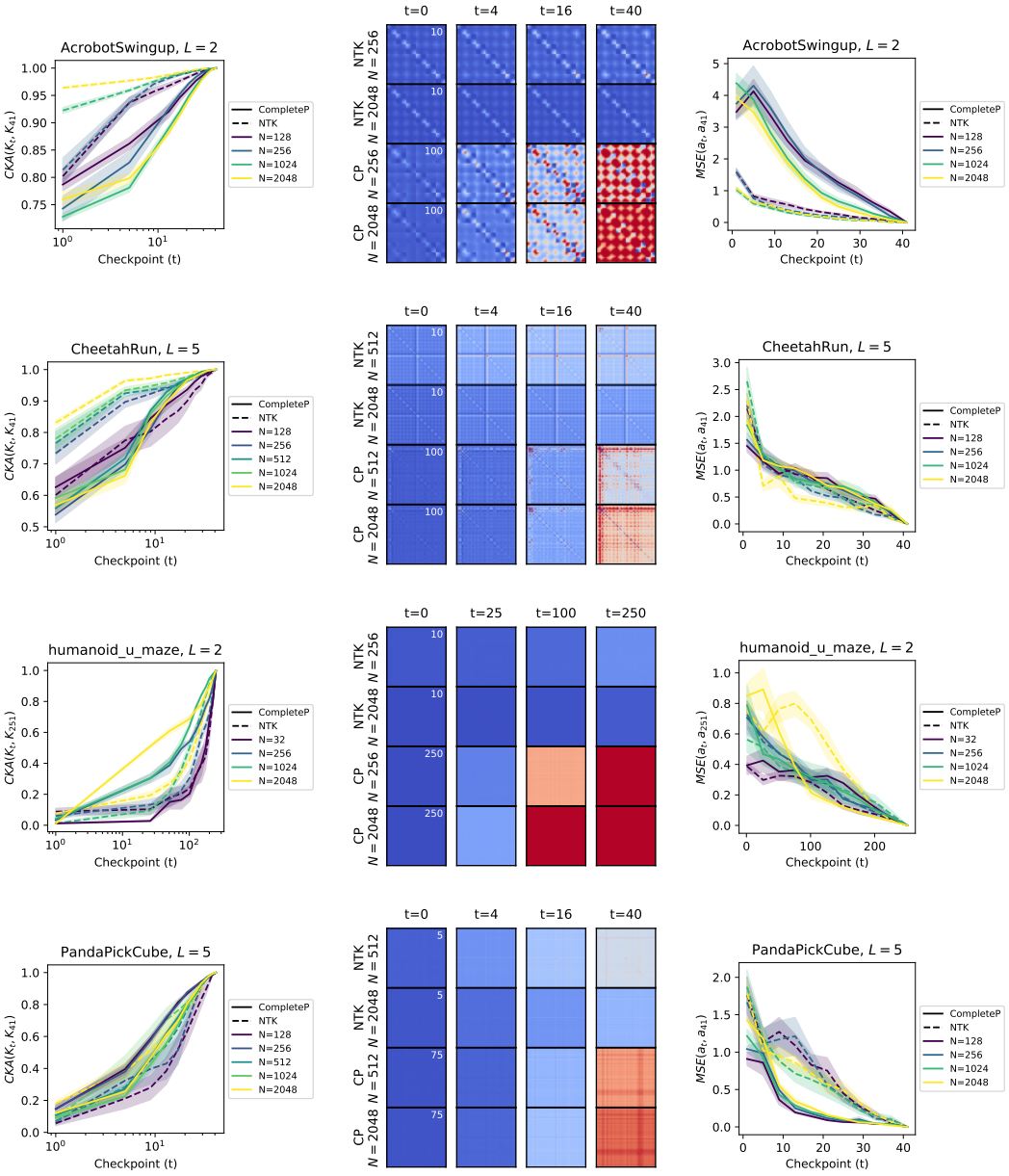


Figure 36: Temporal feature and logit evolution dynamics across environments. Each row corresponds to a task (AcrobotSwingup, CheetahRun, humanoid_u_maze, PandaPickCube); columns show feature CKA, kernel similarity, and logit MSE, respectively.

Q WIDTH SCALING IN ENVIRONMENTS WITH LONGER TRAINING TIME

We provide more plots comparing the performances of CompleteP and NTK agents across different widths. In some of these plots we train agents around 5-10x times longer than before in order to observe convergence in reward. CompleteP tends to perform better than NTK agents, and does not suffer from reward collapse at larger widths.

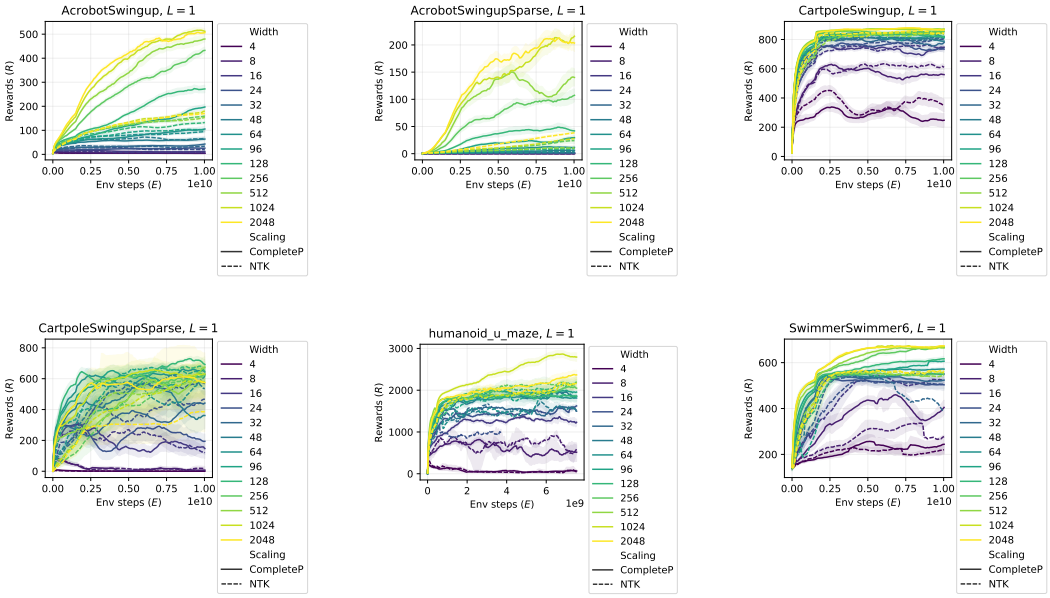


Figure 37: **Environment reward vs timesteps for $L = 1$ across tasks.**

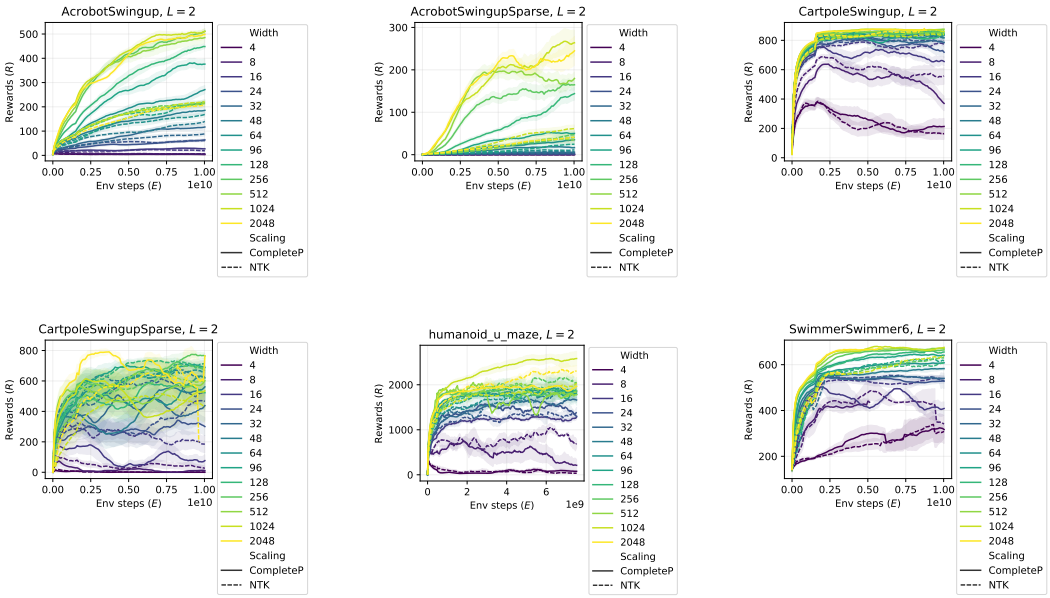


Figure 38: **Environment reward vs timesteps for $L = 2$ across tasks.**

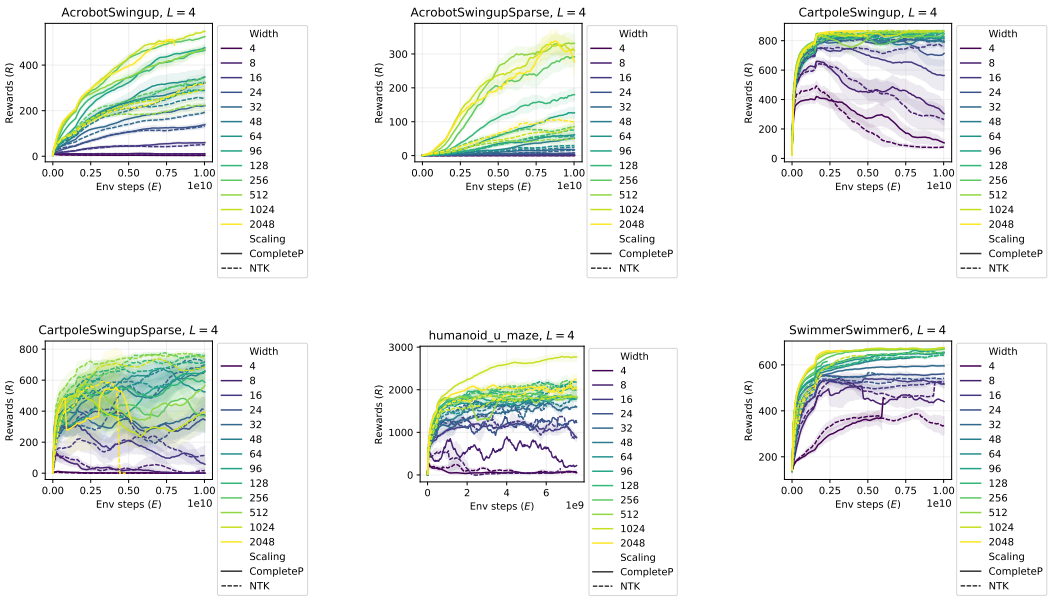


Figure 39: **Environment reward vs timesteps for $L = 4$ across tasks.**

R FLOP CURVES FROM ENVIRONMENTS WITH LONGER TRAINING TIME

What follows are plots describing the relationship of compute usage to reward in the same set of environments as Appendix Q. This corroborates our findings from before: CompleteP agents are better or at parity with NTK agents at smaller widths, and are much more efficient than NTK agents at larger widths.

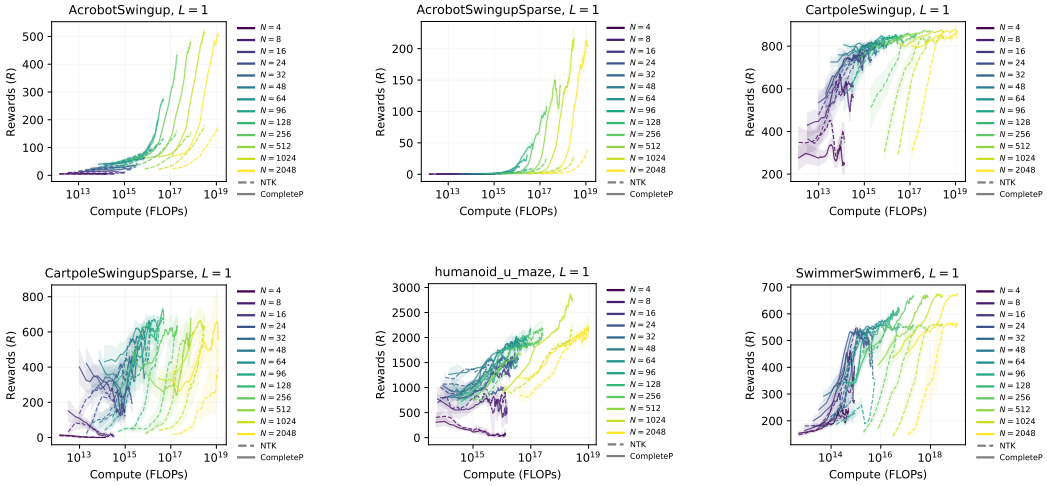


Figure 40: **Reward–compute tradeoffs for $L = 1$ as performance v.s. FLOPs.**

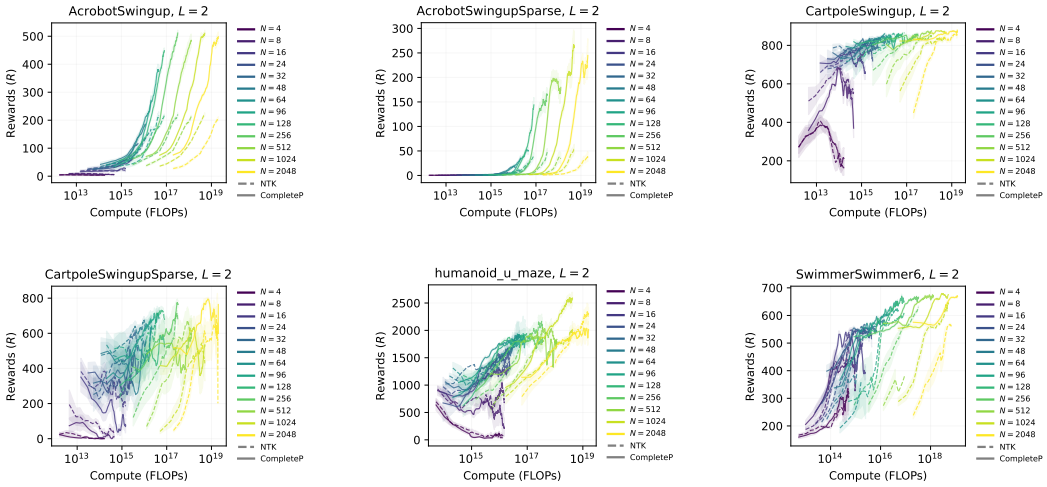


Figure 41: **Reward–compute tradeoffs for $L = 2$ as performance v.s. FLOPs.**

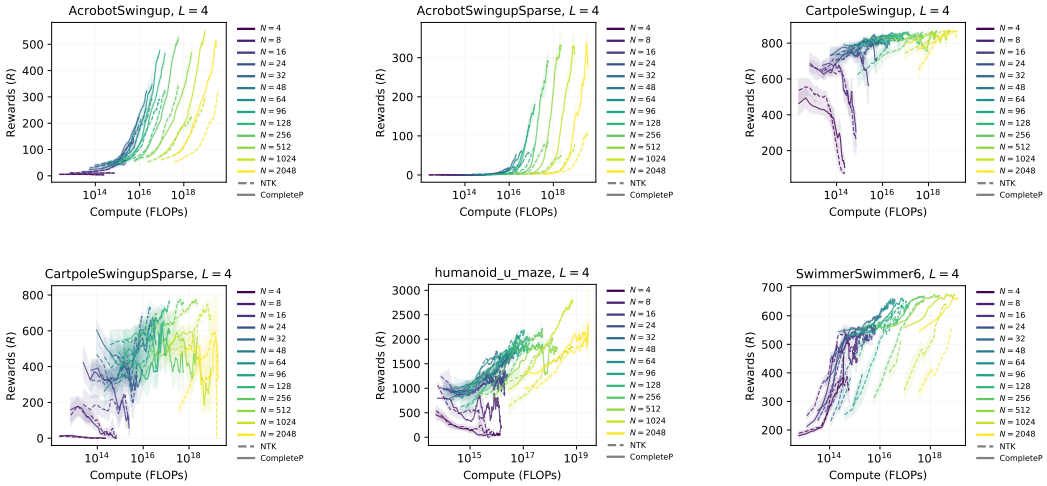


Figure 42: **Reward–compute tradeoffs for $L = 4$ as performance v.s. FLOPs.**

S PARETO CURVES FROM ENVIRONMENTS WITH LONGER TRAINING TIME

What follows are plots describing isotonic regression fits on the relationship of compute usage to reward from the data in Appendix R in the same set of environments as Appendix Q. We find that CompleteP agents perform better or at parity than NTK agents in terms of their pareto frontier. We note that in more complex environments, where more parameters and hence larger widths are required to perform well, CompleteP agents improve upon the pareto frontier. However, in a simpler task such as Cartpole, both CompleteP and NTK agents share roughly the same pareto frontier.

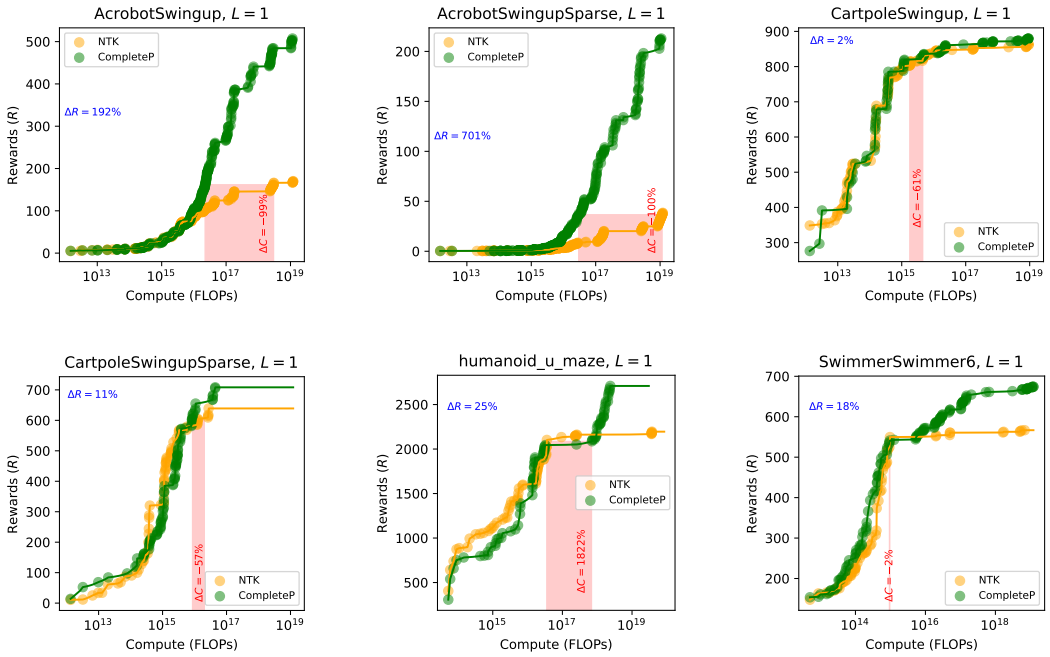


Figure 43: **Pareto frontiers for reward vs compute at $L = 1$.**

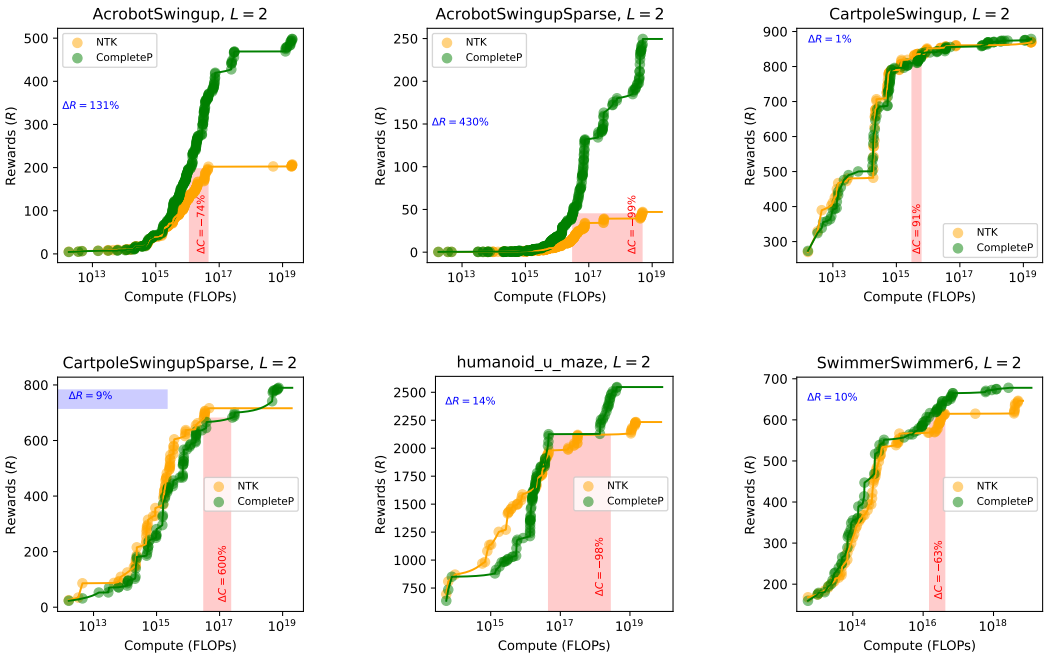


Figure 44: **Pareto frontiers for reward vs compute at $L = 2$.**

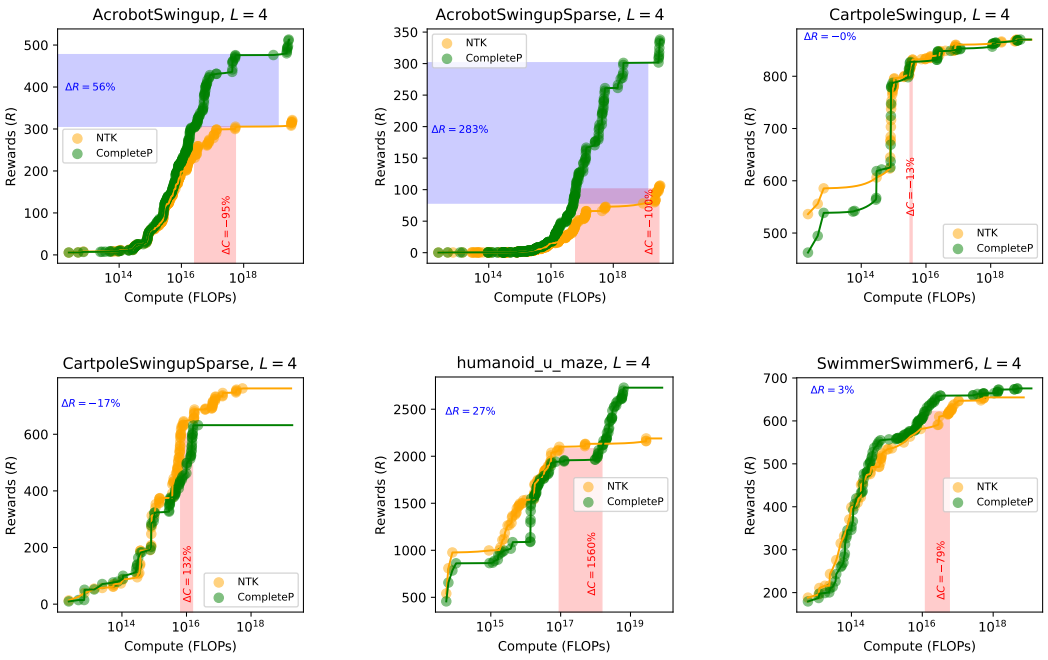


Figure 45: **Pareto frontiers for reward vs compute at $L = 4$.**

T WALLTIME DEPENDENT LEARNING PERFORMANCE

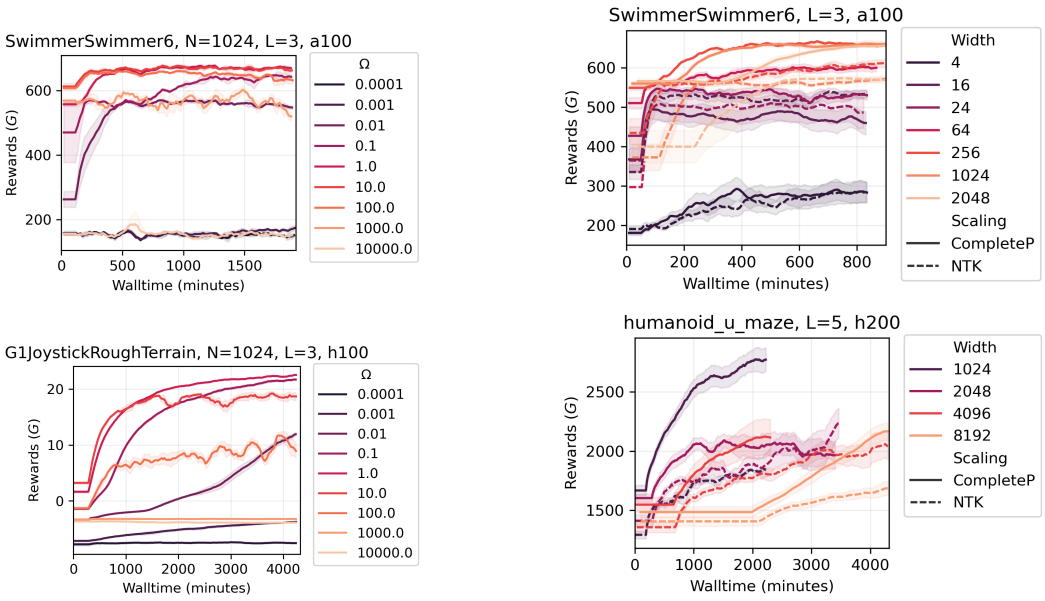


Figure 46: Reward versus wall clock. CompleteP demonstrates improved learning performance over NTK parameterization over different widths and sweeps on Ω. These environments were trained with a single GPU type (A100, H100, or H200) for direct comparison. Note that different widths vary in runtime i.e. larger widths require longer walltime.

U LEARNING RATE TUNED WIDTH SCALING CURVES

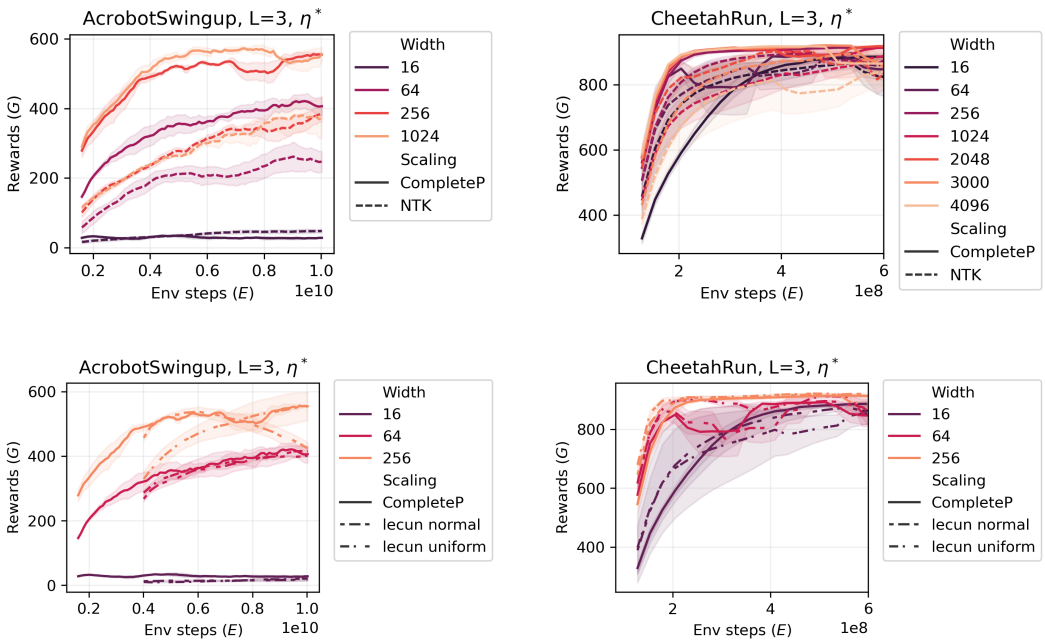


Figure 47: **CompleteP versus NTK or Standard Parameterization with width-dependent learning rates for latter.** (Top) Although the best learning rate for each width is chosen for NTK, these agents struggle to learn as well as CompleteP agents.

Table 11: **Optimal learning rates for Acrobot and CheetahRun across network widths and initializations.** While both NTK and Standard Parameterization (SP) require a reduction in learning rates to achieve maximum performance with width scaling, CompleteP allows learning rate transfer across scales to achieve similar performance in Fig. 47, minimizing compute costs involved in hyperparameter sweeps.

Initialization	N=16	N=64	N=256	N=1024
AcrobotSwingup				
CompleteP (CP)	0.025	0.025	0.025	0.025
NTK	0.001	0.025	0.025	0.025
SP normal	0.0025	0.0025	0.00025	0.0001
SP uniform	0.0025	0.0025	0.0001	0.0001
CheetahRun				
CompleteP (CP)	0.01	0.01	0.0025	0.0025
NTK	0.025	0.025	0.01	0.01
SP normal	0.001	0.001	0.00025	0.0001
SP uniform	0.001	0.001	0.00025	0.0001