APPENDIX

A EXTENDED BACKGROUND

A.1 RL ALGORITHMS DESCRIPTIONS

Continuing from section 2.1, we provide an overview of standard RL definitions and the deep RL algorithms we use in this work.

Two important functions in RL are the value function and the action-value function (also called the Q function). These quantify, for policy π, the expected sum of discounted future rewards given any initial fixed state or state-action pair, respectively:

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \cdots \sim \pi_\theta, P}\left[\sum_{t'=0}^{\infty} \gamma^{t'} r(s_{t+t'}, a_{t+t'})\right], \\ Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P}\left[V^{\pi}(s_{t+1})\right]. \tag{7}$$

Relatedly, the advantage function $A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ quantifies the expected improvement from executing any given action a_t from s_t rather than following the policy. These functions summarize the future evolution of an MDP and are often parameterized and learned auxiliary to or even in-place of the policy model.

On-policy methods. Modern on-policy RL algorithms collect a new set of trajectories at each iteration with the current policy, discarding old data. They use these trajectories to learn the current policy's value function and recover a corresponding advantage function from the observed Monte-Carlo returns, using techniques such as the popular Generalized Advantage Estimator (GAE) (Schulman et al., 2015). The estimated advantages A^{GAE} are then used to compute the *policy gradient* and update the policy, maximizing the probability of performing the best-observed actions (Sutton & Barto, 2018). Since the values of A^{GAE} are based on a limited set of trajectories, on-policy methods generally suffer from high-variance targets and gradients (Pendrith et al., 1997; Mannor et al., 2007; Wu et al., 2018). Proximal Policy Optimization (PPO) (Schulman et al., 2017) is one of the most established on-policy algorithms that attenuates these issues by taking conservative updates, restricting the policy update from making larger than ϵ changes to the probability of executing any individual action. PPO considers the ratio between the updated and old policy probabilities $R_{\pi}(a_t|s_t) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ to optimize a pessimistic clipped objective of the form:

$$\min\{R_{\pi}(a_t|s_t)A^{GAE}(s_t, a_t), \text{clip}(R_{\pi}(a_t|s_t), 1-\epsilon, 1+\epsilon)A^{GAE}(s_t, a_t)\}. \tag{8}$$

As mentioned in the main text, PPO also includes a small entropy bonus to incentivize exploration and improve data diversity. This term can be differentiated and optimized without any estimator since we have full access to the policy model and its output logits, independently of the collected data.

Off-policy methods. In contrast, off-policy algorithms generally follow a significantly different optimization approach. They store many different trajectories collected with a mixture of old policies in a large *replay buffer*, B. They use this data to directly learn the Q function for the optimal greedy policy with a squared loss based on the Bellman backup (Bellman, 1957):

$$E_{(s_t, a_t, s_{t+1}, r_t) \in B}\left[Q(s_t, a_t) - \left(r_t + \max_{a'} Q(s_{t+1}, a')\right)\right]^2 \tag{9}$$

(Bellman, 1957). Agent behavior is then implicitly defined by the *epsilon-greedy* policy based on the actions with the highest estimated Q values. We refer to the deep Q-networks paper (Mnih et al., 2013) for a detailed description of the seminal DQN algorithm. Rainbow DQN (Hessel et al., 2018) is a modern popular extension that introduces several auxiliary practices from proposed orthogonal improvements, which they show provide compatible benefits. In particular, they use n-step returns (Sutton & Barto, 2018), prioritized experience replay (Schaul et al., 2016), double Q-learning (Hasselt, 2010), distributional RL (Bellemare et al., 2017), noisy layers (Fortunato et al., 2018), and a dueling network architecture (Wang et al., 2016).

A.2 δ HYPERBOLICITY

δ-hyperbolicity was introduced by Gromov (1987) as a criterion to quantify how hyperbolic a metric space (X, d) is. We can express δ-hyperbolicity in terms of the *Gromov product*, defined for $x, y \in X$ at some base point $r \in X$ as measuring the defect from the triangle inequality:

$$(x|y)_r = \frac{1}{2}(d(x, r) + d(r, y) - d(x, y)). \tag{10}$$

Then, X is δ-*hyperbolic* if for all base points $r \in X$ and for any three points $x, y, z \in X$ the Gromov product between x and y is lower than the minimum Gromov product of the other pairs by at most some slack variable δ:

$$(x|y)_r \geq \min((x|y)_r, (x|y)_r) - \delta. \tag{11}$$

In our case (a complete finite-dimensional path-connected Riemannian manifold, which is a geodesic metric space), δ-hyperbolicity means that for every point on one of the sides of a geodesic triangle $\triangle xyz$, there exists some other point on one of the other sides whose distance is at most δ, or in other words, geodesic triangles are δ-slim. In trees, the three sides of a triangle must all intersect at some midpoint (Figure 3). Thus, *every point belongs to at least two of its sides* yielding $\delta = 0$. Thus the δ-hyperbolicity can be interpreted as measuring the deviation of a given metric from an exact tree metric.

A.3 GENERATIVE ADVERSARIAL NETWORKS AND SPECTRAL NORMALIZATION

GANs. In GAN training, the goal is to obtain a generator network to output samples resembling some 'true' target distribution. To achieve this, Goodfellow et al. (2014) proposed to alternate training of the generator with training an additional discriminator network, tasked to distinguish between the generated and true samples. The generator's objective is then to maximize the probability of its own samples according to the current discriminator, backpropagating directly through the discriminator's network. Since both the generated data and discriminator's network parameters constantly change from this alternating optimization, the loss landscape of GANs is also highly non-stationary, resembling, to some degree, the RL setting. As analyzed by several works, the adversarial nature of the optimization makes it very brittle to exploding and vanishing gradients instabilities (Arjovsky & Bottou, 2017; Brock et al., 2018) which often result in common failure modes from severe divergence or stalled learning (Lin et al., 2021). Consequently, numerous practices in the GAN literature have been proposed to stabilize training (Radford et al., 2015; Arjovsky et al., 2017; Gulrajani et al., 2017). Inspired by recent work, in this work, we focus specifically on spectral normalization (Miyato et al., 2018), one such practice whose recent success made in ubiquitous in modern GAN implementations.

Spectral normalization. In the adversarial interplay characterizing GAN training, instabilities commonly derive from the gradients of the discriminator network, f_D (Salimans et al., 2016). Hence, Miyato et al. (2018) proposed to regularize the spectral norm of discriminator's layers, $l_j \in f_D$, i.e., the largest singular values of the weight matrices $\|W_j^{WN}\|_{sn} = \sigma(W_j^{SN})$, to be approximately one. Consequently, spectral normalization effectively bounds the Lipschitz constant of the whole discriminator network since, $\|f_D\|_{Lip} \leq \prod_{j=1}^{L} \|l_j\|_{Lip} \leq \prod_{j=1}^{L} \|W_j^{SN}\|_{sn} = 1$. In practice, the proposed implementation approximates the largest singular value of some original unconstrained weight matrices by running power iteration (Golub & Van der Vorst, 2000). Thus, it recovers the spectrally-normalized weights with a simple re-parameterization, dividing the unconstrained weights by their relative singular values $W_j^{SN} = \frac{W_j}{\sigma(W_j)}$. As mentioned in the main text, recent work (Lin et al., 2021) showed that one of the main reasons for the surprising effectiveness of spectral normalization in GAN training comes from *effectively regulating* both the magnitude of the activations and their respective gradients, very similarly to LeCun initialization (LeCun et al., 2012). Furthermore, when applied to the discriminator, spectral normalization's effects appear to persist *throughout training*, while initialization strategies tend to only affect the initial iterations. In fact, in Figure 2 of their paper, they also show that ablating spectral normalization empirically results in exploding gradients and degraded performance, closely resembling our same observed instabilities in Figure 6 (B).

Other recent works also studied the application of spectral normalization (Miyato et al., 2018) for reinforcement learning (Bjorck et al., 2021; Gogianu et al., 2021). In line with our results from Figure 7, they found that naively applying spectral normalization to traditional Euclidean architectures

leads to underwhelming results for RL. Yet, they also observed performance benefits when applying spectral normalization exclusively to particular layers. These empirical insights could inform future improvements for S-RYM to retain the stability benefits of SN with less restrictive regularization.

B STABILIZATION OF HYPERBOLIC REPRESENTATIONS

One of the main challenges of incorporating hyperbolic geometry with neural networks comes from end-to-end optimization of latent representations and parameters located in hyperbolic space. For instance, numerical issues and vanishing gradients occur as representations get too close to either the origin or the boundary of the Poincaré ball (Ganea et al., 2018). Moreover, training dynamics can tend to push representations towards the boundary, slowing down learning and make optimization problems of earlier layers ineffective (Guo et al., 2022). A number of methods have been used to help stabilize learning of hyperbolic representations including constraining the representations to have a low magnitude early in training, applying clipping and perturbations (Ganea et al., 2018; Khrulkov et al., 2020), actively masking invalid gradients (Mathieu et al., 2019), and designing initial 'burn-in' periods of training with lower learning rates (Nickel & Kiela, 2017; Bécigneul & Ganea, 2018). More recently Guo et al. (2022) also showed that very significant magnitude clipping of the latent representations can effectively attenuate these numerical and learning instabilities when training hyperbolic classifiers for popular image classification benchmarks.

B.1 MAGNITUDE CLIPPING

Guo et al. (2022) recently proposed to apply significant clipping of the magnitude of the latent representations when using hyperbolic representations within deep neural networks. As in our work, they also consider a *hybrid* architecture where they apply an exponential map before the final layer to obtain latent representations in hyperbolic space. They apply the proposed clipping to constrain the input vector of the exponential map to not exceed unit norm, producing hyperbolic representations via:

$$x_H = \exp_0^1\left(\min\left\{1, \frac{1}{||x_E||}\right\} \times x_E\right). \tag{12}$$

The main motivation for this practice is to constrain representation magnitudes, which the authors linked to a vanishing gradient phenomenon when training on standard image classification datasets (Krizhevsky et al., 2009; Deng et al., 2009). However, a side effect of this formulation is that the learning signal from the representations exceeding a magnitude of 1 will solely convey information about the representation's direction and not its magnitude. Since the authors do not share their implementation, we tested applying their technique as described in the paper. We found some benefits in additionally initializing the parameters of the last two linear layers (in Euclidean and hyperbolic space) to $100\times$ smaller values to facilitate learning initial angular layouts.

B.2 IMAGE CLASSIFICATION EXPERIMENTS

To empirically validate and analyze our clipping implementation we consider evaluating deep hyperbolic representations on image classification tasks, following the same training practices and datasets from Guo et al. (2022). In particular, we utilize a standard ResNet18 architecture (He et al., 2016b) and test our network on CIFAR10 and CIFAR100 (Krizhevsky et al., 2009). We optimize the Euclidean parameters of the classifier using stochastic gradient descent with momentum and the hyperbolic parameters using its Riemmanian analogue (Bonnabel, 2013). We train for 100 epochs with an initial learning rate of 0.1 and a cosine schedule (Loshchilov & Hutter, 2017), using a standard batch size of 128. We repeat each experiment 3 times, recording the final top-1 classification accuracy together with the latent representations in Euclidean space right before applying the exponential map at the final layer.

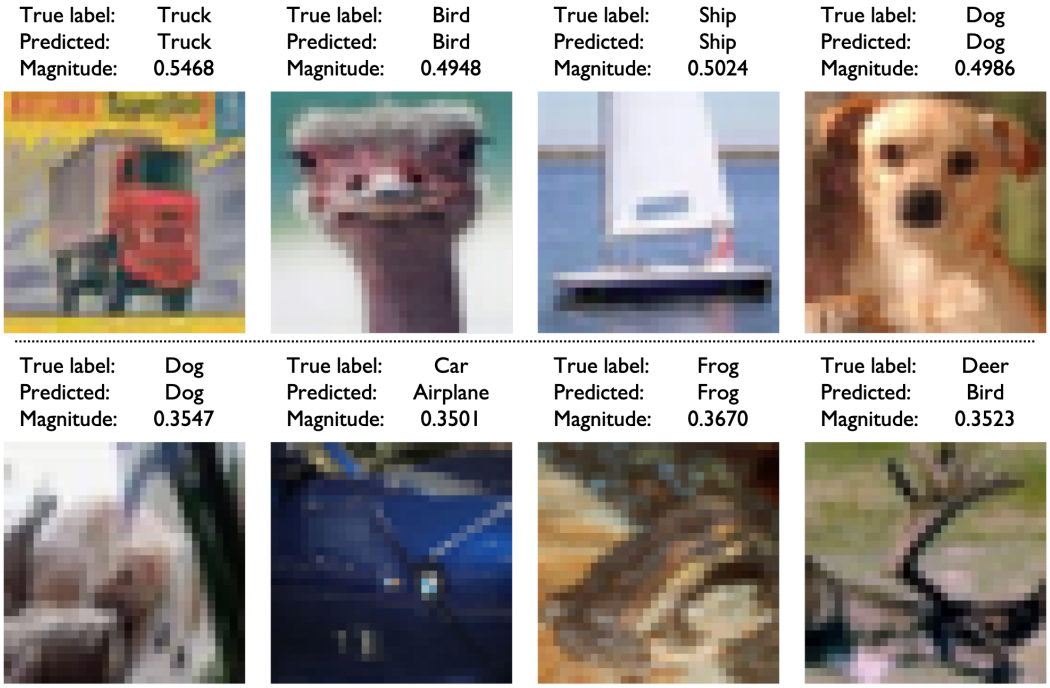

Figure 11: Visualization of test images from CIFAR10, with the corresponding final latent representations magnitudes from our hyperbolic ResNet18 classifier implemented with S-RYM. We sample datapoints with the 5% highest magnitudes (**Top**) and the 5% lowest magnitudes (**Bottom**).

Table 3: Performance results on standard image classification benchmarks

CIFAR10 with ResNet18			
Metric\Architecture	Euclidean	Hyperbolic + Clipping	Hyperbolic + S-RYM
Top-1 accuracy	94.92 ± 0.19	94.81 ± 0.17	$\mathbf{95.12 \pm 0.09}$
L2 representations magnitudes	5.846	1.00	0.481
Magnitudes standard deviation	0.747	0.00	0.039
CIFAR100 with ResNet18			
Metric\Architecture	Euclidean ResNet18	Hyperbolic + Clipping	Hyperbolic + S-RYM
Top-1 accuracy	76.86 ± 0.23	76.75 ± 0.23	$\mathbf{77.49 \pm 0.35}$
L2 representations magnitudes	11.30	1.00	0.852
Magnitudes standard deviation	1.571	0.00	0.076

In Table 3, we report the different classifiers' performance together with the mean and standard deviation of the representations' magnitudes from the images in the test set. The performance of the clipped hyperbolic classifier is very close to the performance of the Euclidean classifier, matching Guo et al. (2022)'s results and validating our implementation. However, the learned representations' magnitudes soon overshoot the clipping threshold and get mapped to constant-magnitude vectors throughout training. Therefore, the model will effectively stop optimizing for the representations' magnitudes and only focus on their *unit direction*. As volume and distances on the Poincaré ball grow expoentially with radius, the magnitude component of the hyperbolic representations is precisely what facilitates encoding hierarchical information, providing its intuitive connection with tree structures. Hence, the resulting clipped 'hyperbolic' space spanned by the clipped latent representations will lose its *defining* degree of freedom and approximately resemble an $n-1$-dimensional Euclidean space with a rescaled metric, potentially explaining its performance similarity with standard Euclidean classifiers. Even though the focus of our work is not image classification, we find

S-RYM's performance remarkably recovers and even marginally exceeds the performance of both the Euclidean and the clipped hyperbolic classifiers on these saturated benchmarks. Furthermore, its representations do not explode and maintain magnitude diversity, enabling to more efficiently capture the relative hierarchical nature of image-classification benchmarks (Khrulkov et al., 2020). Overall, these results suggest that clipping simply treats the *symptoms* of the instabilities caused by end-to-end large scale training by essentially resorting back to Euclidean representations for image classification.

Analyzing the magnitude component of the latent representations for our hyperbolic classifier with S-RYM, we find it correlates with classification performance. For instance, on CIFAR10 the test performance on the images with representations's with the top 5% magnitudes is 97.17%, while for the bottom 5% is 79.64%. Furthermore, we display some samples from these two distinct groups in Figure 11. From these results and visualizations, it appears that the hyperbolic hierarchical structure serves to encode the degree of uncertainty to disambiguate between multiple image labels due to the blurriness and varying difficulty of the CIFAR10 datapoints. Hence, we believe the observed accuracy improvements of our hyperbolic classifier might be specifically due to more efficiently capturing this specific hierarchical property of the considered datasets.

C Implementation details

We provide a descriptions of the utilized benchmarks and implementations with the corresponding hyper-parameters for both our Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Rainbow DQN experiments (Hessel et al., 2018). We consider these two main baselines since they are two of the most studied algorithms in the recent RL literature onto which many other recent advances also build upon (e.g., (Cobbe et al., 2021; Laskin et al., 2020b; Mohanty et al., 2021; Raileanu et al., 2020; Raileanu & Fergus, 2021; Yarats et al., 2021a; Van Hasselt et al., 2019; Laskin et al., 2020a; Schwarzer et al., 2020)). Furthermore, PPO and Rainbow DQN are based on the main families of model-free RL algorithms, with very distinct properties as described in Appendix A.1. Hence, unlike most prior advances, we do not constrain our analysis to a single class of methods, empirically showing the generality of hyperbolic deep RL. Our implementations closely follow the reported details from recent research, and were not tuned to facilitate our integration of hyperbolic representations. The main reason for this choice is that we wanted to avoid introducing additional confounding factors from our evaluation of hyperbolic representations, as ad-hoc tuning frequently plays a significant role in RL performance (Islam et al., 2017). We would like to acknowledge the *Geoopt* optimization library (He et al., 2016a), which we used to efficiently train the network parameters located on Riemannian manifolds other than $\mathbb{R}^n$.

C.1 Benchmarks

Procgen. The Procgen generalization benchmark (Cobbe et al., 2020) consists of 16 game environments with procedurally-generated random levels. The state spaces of these environments consist of the RGB values from the 64x64 rescaled visual renderings. Following common practice and the recommended settings, we consider training agents using exclusively the first 200 levels of each environment and evaluate on the full distribution of levels to assess agent performance and generalization. Furthermore, we train for 25M total environment steps and record final training/test performance collected across the last 100K steps averaged over 100 evaluation rollouts.

Atari 100K. The Atari 100K benchmark (Kaiser et al., 2020) is based on the seminal problems from the Atari Learning Environment (Bellemare et al., 2013). In particular, this benchmark consists of 26 different environments and only 100K total environment steps for learning each, corresponding roughly to 2hrs of play time. The environments are modified with the specifications from (Machado et al., 2018), making the state spaces of these environments 84x84 rescaled visual renderings and introducing randomness through *sticky actions*. We note that this is a significantly different setting than Procgen, testing the bounds for the sample efficiency of RL agents.

C.2 PPO implementation

Our PPO implementation follows the original Procgen paper (Cobbe et al., 2020), which entails a residual convolutional network (Espeholt et al., 2018) and produces a final 256-dimensional latent

representation with a shared backbone for both the policy and value function. Many prior improvements over PPO for on-policy learning have been characterized by either introducing auxiliary domain-specific practices, increasing the total number of parameters, or performing additional optimization phases - leading to significant computational overheads (Cobbe et al., 2021; Raileanu & Fergus, 2021; Mohanty et al., 2021). Instead, our approach strives for an orthogonal direction by simply utilizing hyperbolic geometry to facilitate encoding hierarchically-structured features into the final latent representations. Thus, it can be interpreted as a new way to *modify the inductive bias of deep learning models for reinforcement learning*.

Table 4: PPO hyper-parameters used for the Procgen generalization benchmark

PPO hyperparameters	
Parallel environments	64
Stacked input frames	1
Steps per rollout	16384
Training epochs per rollout	3
Batch size	2048
Normalize rewards	True
Discount γ	0.999
GAE λ (Schulman et al., 2015)	0.95
PPO clipping	0.2
Entropy coefficient	0.01
Value coefficient	0.5
Shared network	True
Impala stack filter sizes	16, 32, 32
Default latent representation size	256
Optimizer	*Adam* (Kingma & Ba, 2014)
Optimizer learning rate	5×10^-4
Optimizer stabilization constant (ϵ)	1×10^-5
Maximum gradient norm.	0.5

In Table 4 we provide further details of our PPO hyper-parameters, as also described by the original Procgen paper (Cobbe et al., 2020). When using hyperbolic latent representations, we optimize the hyperbolic weights of the final Gyroplane linear layer with the Riemannian Adam optimizer (Bécigneul & Ganea, 2018), keeping the same learning rate and other default parameters. As per common practice win on-policy methods, we initialize the parameters of the final layer with $100\times$ times lower magnitude. We implemented the naive hyperbolic reinforcement learning implementations introduced in Subsection 3.2 by initializing also the weights of the preceding layer with $100\times$ lower magnitudes to facilitate learning appropriate angular layouts in the early training iterations. We found our S-RYM stabilization procedure also enable to safely remove this practice with no effects on performance.

C.3 Rainbow implementations

Our implementation of Rainbow DQN uses the same residual network architecture as our PPO implementation (Espeholt et al., 2018) but employs a final latent dimensionality of 512, as again specified by Cobbe et al. (2020). Since Cobbe et al. (2020) do not open-source their Rainbow implementation and do not provide many of the relative details, we strive for a simple implementation removing unnecessary complexity and boosting overall efficiency. Following Castro et al. (2018), we only consider Rainbow DQN's three most significant advances over vanilla DQN (Mnih et al., 2013): distributional critics (Bellemare et al., 2017), prioritized experience replay (Schaul et al., 2016), and n-step returns (Sutton & Barto, 2018). While the methodology underlying off-policy algorithms is fundamentally different from their on-policy counterparts, we apply the same exact recipe of integrating hyperbolic representations in the final layer, and compare against the same variations and baselines.

Table 5: Rainbow DQN hyper-parameters used for the Procgen generalization benchmark

Rainbow DQN Procgen hyperparameters	
Parallel environments	64
Stacked input frames	1
Replay buffer size	1.28M
Batch size	512
Minimum data before training	32K steps
Update network every	256 env. steps
Update target network every	12800 env. steps
ϵ-greedy exploration schedule	1→0.01 in 512K steps
Discount γ	0.99
N-step	3
Use dueling (Wang et al., 2016)	False
Use noisy layers (Fortunato et al., 2018)	False
Use prioritized replay (Schaul et al., 2016)	True
Use distributional value (Bellemare et al., 2017)	True
Distributional bins	51
Maximum distributional value	10
Minimum distributional value	-10
Impala stack filter sizes	16, 32, 32
Default latent representation size	512
Optimizer	*Adam* (Kingma & Ba, 2014)
Optimizer learning rate	$5\times10^{-}4$
Optimizer stabilization constant (ϵ)	$1\times10^{-}5$
Maximum gradient norm.	0.5

In Table 5 we provide details of our Rainbow DQN hyper-parameters. We note that sampling of off-policy transitions with n-step returns requires retrieving the future n rewards and observations. To perform this efficiently while gathering multiple transitions from the parallelized environment, we implemented a parellalized version of a segment tree. In particular, this extends the original implementation proposed by Schaul et al. (2016), through updating a set of segment trees implemented as a unique data-structure with a single parallelized operation, allowing for computational efficiency without requiring any storage redundancy. We refer to the shared code for further details. As with our hyperbolic PPO extensions, we also optimize the final layer's hyperbolic weights with Riemannian Adam, keeping the same parameters as for the Adam optimizer used in the other Euclidean layers.

Table 6: Rainbow DQN hyper-parameters changes for the Atari 100K benchmark

Rainbow DQN Atari 100K training hyper-parameters	
Stacked input frames	4
Batch size	32
Minimum data before training	1600 steps
Network updates per step	2
Update target network every	1 env. steps
ϵ-greedy exploration schedule	1→0.01 in 20K steps

The characteristics of the Atari 100K benchmark are severely different from Procgen, given by the lack of parallelized environments and the $250\times$ reduction in total training data. Hence, we make a minimal set of changes to the training loop hyper-parameters of our Rainbow DQN implementation to ensure effective learning, as detailed in Table 6. These are based on standard practices employed by off-policy algorithm evaluating on the Atari 100K benchmark (Van Hasselt et al., 2019; Laskin et al., 2020a; Yarats et al., 2021a) and were not tuned for our specific implementation.

Table 7: Detailed performance comparison for the Rainbow DQN algorithm on the full Procgen benchmark. We *train* for a total of 25M steps on 200 training levels and *test* on the full distribution of levels. We report the mean returns, the standard deviation, and relative improvements from the original Rainbow DQN baseline over 5 random seeds.

Task\Algorithm	Rainbow DQN	Rainbow DQN + data aug.	Rainbow DQN + S-RYM	Rainbow DQN + S-RYM, 32 dim.
Levels distribution	*train/test*	*train/test*	*train/test*	*train/test*
bigfish	23.17±4 15.47±2	19.61±4 (-15%) 17.39±4 (+12%)	27.61±0 (+19%) **23.03±2 (+49%)**	30.85±2 (+33%) 22.37±2 (+45%)
bossfight	7.17±1 7.29±1	6.22±1 (-13%) 6.97±1 (-4%)	9.41±1 (+31%) 7.75±1 (+6%)	8.21±1 (+15%) **8.71±1 (+20%)**
caveflyer	7.00±1 3.92±1	7.59±0 (+8%) 5.36±1 (+37%)	6.39±1 (-9%) 3.11±1 (-21%)	6.45±1 (-8%) **5.46±1 (+39%)**
chaser	3.09±1 2.31±0	2.89±0 (-6%) 2.61±0 (+13%)	4.03±1 (+30%) **3.65±1 (+58%)**	3.78±0 (+23%) 3.29±0 (+43%)
climber	3.68±1 1.73±1	2.57±1 (-30%) 2.36±1 (+36%)	3.91±0 (+6%) 2.39±0 (+38%)	4.80±2 (+31%) **3.00±0 (+73%)**
coinrun	5.56±1 4.33±1	3.22±1 (-42%) 3.00±1 (-31%)	5.20±0 (-6%) 5.07±1 (+17%)	6.00±1 (+8%) **6.33±1 (+46%)**
dodgeball	7.42±1 4.67±1	8.91±1 (+20%) **6.96±1 (+49%)**	6.07±1 (-18%) 3.60±1 (-23%)	6.89±1 (-7%) 5.31±1 (+14%)
fruitbot	21.51±3 16.94±2	22.29±2 (+4%) 20.53±3 (+21%)	20.31±1 (-6%) 20.30±1 (+20%)	22.81±1 (+6%) **21.87±2 (+29%)**
heist	0.67±0 0.11±0	1.67±0 (+150%) **0.67±0 (+500%)**	1.27±0 (+90%) 0.40±0 (+260%)	0.93±1 (+40%) 0.47±0 (+320%)
jumper	5.33±1 3.11±0	4.22±0 (-21%) 2.78±1 (-11%)	4.78±1 (-10%) 2.44±1 (-21%)	5.53±1 (+4%) **3.47±1 (+11%)**
leaper	1.78±1 2.56±1	6.11±1 (+244%) **5.11±1 (+100%)**	2.00±1 (+13%) 1.00±0 (-61%)	0.80±0 (-55%) 0.53±0 (-79%)
miner	2.22±1 **2.33±0**	1.89±0 (-15%) 1.33±1 (-43%)	2.40±0 (+8%) 1.40±0 (-40%)	2.73±1 (+23%) 2.00±0 (-14%)
maze	2.01±0 0.67±0	2.07±0 (+3%) **1.58±1 (+137%)**	1.91±0 (-5%) 0.93±0 (+40%)	1.97±0 (-2%) 0.92±0 (+38%)
ninja	3.33±0 2.33±1	3.44±1 (+3%) 2.56±1 (+10%)	3.33±1 (+0%) 2.11±0 (-10%)	3.73±1 (+12%) **3.33±1 (+43%)**
plunder	8.69±0 **6.28±1**	6.06±1 (-30%) 5.30±1 (-16%)	7.33±1 (-16%) 5.93±1 (-5%)	7.11±1 (-18%) 5.71±1 (-9%)
starpilot	47.83±6 42.42±1	51.79±3 (+8%) 46.23±5 (+9%)	57.64±2 (+21%) **55.86±3 (+32%)**	59.94±1 (+25%) 54.77±3 (+29%)
Average norm. score	0.2679 0.1605	0.2698 (+1%) 0.2106 (+31%)	0.2774 (+4%) 0.1959 (+22%)	**0.3097 (+16%) 0.2432 (+51%)**
Median norm. score	0.1856 0.0328	0.1830 (-1%) 0.1010 (+208%)	0.2171 (+17%) 0.0250 (-24%)	**0.2634 (+42%) 0.1559 (+376%)**
# Env. improvements	0/16 0/16	8/16 11/16	8/16 9/16	**11/16 13/16**

D Extended results and comparisons

In this Section we provide detailed per-environment Rainbow DQN Procgen results that were omitted from the main text due to space constraints. For both Rainbow DQN and PPO, we also compare the performance improvements from the integration of our deep hyperbolic representations with the reported improvements from recent state-of-the-art (SotA) algorithms, employing one or several orthogonal domain-specific practices. In Appendix E.3, we provide examples empirically validating that hyperbolic representations provide mostly complementary benefits and are compatible with different domain-specific practices, potentially yielding even further performance gains.

D.1 Rainbow DQN Procgen results

As shown in Table 7, our hyperbolic Rainbow DQN with S-RYM appears to yield conspicuous performance gains on the majority of the environments. Once again, we find that reducing the dimensionality of the representations to 32 provides even further benefits, outperforming the Euclidean baseline in 13 out of 16 environments. This result not only highlights the efficiency of hyperbolic geometry to encode hierarchical features, but also appears to validate our intuition about the usefulness of regularizing the encoding of non-hierarchical and potentially spurious information. While still inferior to our best hyperbolic implementation, data augmentations seem to have a greater overall beneficial effect when applied to Rainbow DQN rather than PPO. We believe this result is linked with recent literature (Cetin et al., 2022) showing that data-augmentation also provides off-policy RL with an auxiliary regularization effect that stabilizes temporal-difference learning.

D.2 SotA comparison on Procgen

In Table 8 we compare our best hyperbolic PPO agent with the reported results for the current SotA Procgen algorithms from Raileanu & Fergus (2021). All these works propose domain-specific practices on top of PPO (Schulman et al., 2017), designed and tuned for the Procgen benchmark: Mixture Regularization (MixReg) (Wang et al., 2020), Prioritized Level Replay (PLR) (Jiang et al., 2021), Data-regularized Actor-Critic (DraC) (Raileanu et al., 2020), Phasic Policy Gradient (PPG) (Cobbe et al., 2021), and Invariant Decoupled Advantage Actor Critic (Raileanu & Fergus, 2021).Validating our implementation, we see that our Euclidean PPO results closely match the previously reported ones, lagging severely behind all other methods. In contrast, we see that introducing our deep hyperbolic representations framework makes PPO outperform all considered baselines but IDAAC, attaining overall similar scores to this algorithm employing several domain-specific practices. In

Table 8: Performance comparison on the test distribution of levels for our Euclidean and Hyperbolic PPO agents with the reported results of recent RL algorithms designed specifically for the Procgen benchmark.

Task\Algorithm	PPO (Reported)	Mixreg	PLR	UCB-DrAC	PPG	IDAAC	PPO (Ours)	Hyperbolic PPO + S-RYM (Ours)
bigfish	3.7	7.1	10.9	9.2	11.2	18.5	1.46±1	**16.57±2 (+1037%)**
bossfight	7.4	8.2	8.9	7.8	**10.3**	9.8	7.04±2	9.02±1 (+28%)
caveflyer	5.1	6.1	**6.3**	5	7	5	5.86±1	5.20±1 (-11%)
chaser	3.5	5.8	6.9	6.3	**9.8**	6.8	5.89±1	7.32±1 (+24%)
climber	5.6	6.9	6.3	6.3	2.8	**8.3**	5.11±1	7.28±1 (+43%)
coinrun	8.6	8.6	8.8	8.6	8.9	**9.4**	8.25±0	9.20±0 (+12%)
dodgeball	1.6	1.7	1.8	4.2	2.3	3.2	1.87±1	**7.14±1 (+281%)**
fruitbot	26.2	27.3	28	27.6	27.8	27.9	26.33±2	**29.51±1 (+12%)**
heist	2.5	2.6	2.9	3.5	2.8	3.5	2.92±1	**3.60±1 (+23%)**
jumper	5.9	6	5.8	6.2	5.9	**6.3**	6.14±1	6.10±1 (-1%)
leaper	4.9	5.3	6.8	4.8	**8.5**	7.7	4.36±2	7.00±1 (+61%)
maze	5.5	5.2	5.5	6.3	5.1	5.6	6.50±0	**7.10±1 (+9%)**
miner	8.4	9.4	9.6	9.2	7.4	9.5	9.28±1	**9.86±1 (+6%)**
ninja	5.9	6.8	**7.2**	6.6	6.6	6.8	6.50±1	5.60±1 (-14%)
plunder	5.2	5.9	8.7	8.3	14.3	**23.3**	6.06±3	6.68±0 (+10%)
starpilot	24.9	32.4	27.9	30	**47.2**	37	26.57±5	38.27±5 (+44%)
Average norm. score	0.3078	0.3712	0.4139	0.3931	0.4488	**0.5048**	0.3476	0.4730 (+36%)
Median norm. score	0.3055	0.4263	0.4093	0.4264	0.4456	**0.5343**	0.3457	0.4705 (+36%)

Table 9: Performance comparison for our Euclidean and Hyperbolic Rainbow DQN agents with the reported results of recent RL algorithms designed specifically for the Atari 100K benchmark.

Task\Algorithm	Random	Human	DER	OTRainbow	CURL	DrQ	SPR	Rainbow DQN (Ours)	Rainbow DQN + S-RYM (Ours)
Alien	227.80	7127.70	739.9	**824.7**	558.2	771.2	801.5	548.33	679.20 (+41%)
Amidar	5.80	1719.50	**188.6**	82.8	142.1	102.8	176.3	132.55	118.62 (-11%)
Assault	222.40	742.00	431.2	351.9	600.6	452.4	571	539.87	**706.26 (+52%)**
Asterix	210.00	8503.30	470.8	628.5	734.5	603.5	**977.8**	448.33	535.00 (+36%)
Bank Heist	14.20	753.10	51	182.1	131.6	168.9	**380.9**	187.5	255.00 (+39%)
Battle Zone	2360.00	37187.50	10124.6	4060.6	14870	12954	16651	12466.7	**25800.00 (+132%)**
Boxing	0.10	12.10	0.2	2.5	1.2	6	**35.8**	2.92	9.28 (+226%)
Breakout	1.70	30.50	1.9	9.8	4.9	16.1	17.1	13.72	**58.18 (+370%)**
Chopper Command	811.00	7387.80	861.8	1033.3	**1058.5**	780.3	974.8	791.67	888.00 (+498%)
Crazy Climber	10780.50	35829.40	16185.3	21327.8	12146.5	20516.5	**42923.6**	20496.7	22226.00 (+18%)
Demon Attack	152.10	1971.00	508	711.8	817.6	1113.4	545.2	1204.75	**4031.60 (+269%)**
Freeway	0.00	29.60	27.9	25	26.7	9.8	24.4	**30.5**	29.50 (-3%)
Frostbite	65.20	4334.70	866.8	231.6	1181.3	331.1	**1821.5**	318.17	1112.20 (+314%)
Gopher	257.60	2412.50	349.5	778	669.3	636.3	715.2	343.67	**1132.80 (+917%)**
Hero	1027.00	30826.40	6857	6458.8	6279.3	3736.3	7019.2	**9453.25**	7654.40 (-21%)
Jamesbond	29.00	302.80	301.6	112.3	471	236	365.4	190.83	**380.00 (+117%)**
Kangaroo	52.00	3035.00	779.3	605.4	872.5	940.6	**3276.4**	1200	1020.00 (-16%)
Krull	1598.00	2665.50	2851.5	3277.9	**4229.6**	4018.1	3688.9	3445.02	3885.02 (+24%)
Kung Fu Master	258.50	22736.30	**14346.1**	5722.2	14307.8	9111	13192.7	7145	10604.00 (+50%)
Ms Pacman	307.30	6951.60	1204.1	941.9	**1465.5**	960.5	1313.2	1044.17	1135.60 (+12%)
Pong	-20.70	14.60	-19.3	1.3	-16.5	-8.5	-5.9	3.85	**11.98 (+33%)**
Private Eye	24.90	69571.30	97.8	100	**218.4**	-13.6	124	72.28	106.06 (+71%)
Qbert	163.90	13455.00	1152.9	509.3	1042.4	854.4	669.1	860.83	**2702.00 (+264%)**
Road Runner	11.50	7845.00	9600	2696.7	5661	8895.1	14220.5	6090	**22256.00 (+266%)**
Seaquest	68.40	42054.70	354.1	286.9	384.5	301.2	**583.1**	259.33	476.80 (+114%)
Up N Down	533.40	11693.20	2877.4	2847.6	2955.2	3180.8	**28138.5**	2935.67	3255.00 (+13%)
Human Norm. Mean	0.000	1.000	0.285	0.264	0.381	0.357	**0.704**	0.353	0.686 (+94%)
Human Norm. Median	0.000	1.000	0.161	0.204	0.175	0.268	**0.415**	0.259	0.366 (+41%)
# Super	N/A	N/A	2	1	2	2	**7**	2	5

particular, IDAAC not only makes use of a very specialized architecture, but also introduces an auxiliary objective to minimize the correlation between the policy representations and the number of steps until task-completion. Raileanu & Fergus (2021) found this measure to be an effective heuristic correlating with the occurrence of overfitting in many Procgen environments. Moreover, we see that our hyperbolic PPO attains the best performance on 7 different environments, more than any other method. Furthermore, in these environment the other Euclidean algorithms specifically struggle, again indicating the orthogonal effects of our approach as compared to traditional RL advances.

D.3 SOTA COMPARISON ON ATARI 100K

In Table 9 we provide detailed raw results for our hyperbolic Rainbow DQN agent, comparing with the results for recent off-policy algorithms for the Atari 100K benchmark, as reported by

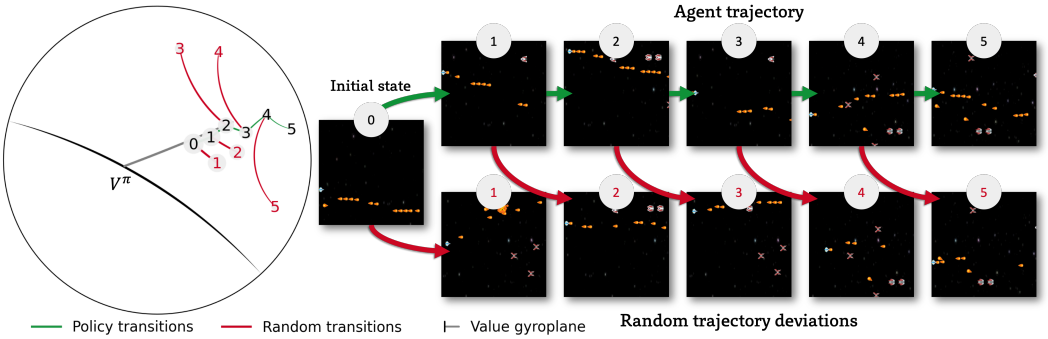

Figure 12: Visualization of 2-dimensional hyperbolic embeddings in the *starpilot* Procgen environment. We sub-sample states from recorded agent trajectories every 15 timesteps. We show the evolution of the hyperbolic latent representations following the recorded policy transitions as compared to random transitions collected by resetting the environments from each state and executing a random policy for the same 15 timesteps.

Schwarzer et al. (2020). All the considered algorithms build on top of the original Rainbow algorithm (Hessel et al., 2018). We consider Data Efficient Rainbow (DER) (Van Hasselt et al., 2019) and Overtrained Rainbow (OTRainbow) (Kielak, 2019) which simply improve the model architectures and other training-loop hyper-parameters, for instance, increasing the number of update steps per collected environment step. We also compare with other more recent baselines that incorporate several additional auxiliary practices and data-augmentation such as Data-regularized Q (DrQ) (Yarats et al., 2021a), Contrastive Unsupervised Representations (CURL) (Laskin et al., 2020a), and Self-Predictive Representations (SPR) (Schwarzer et al., 2020). While our Euclidean Rainbow implementation attains only mediocre scores, once again we see that introducing our deep hyperbolic representations makes our approach competitive with the state-of-the-art and highly-tuned SPR algorithm. In particular, SPR makes use of several architectural advancements, data-augmentation strategies from prior work, and a model-based contrastive auxiliary learning phase. Also on this benchmark, our hyperbolic agent attains the best performance on 8 different environments, more than any other considered algorithm.

D.4 2-DIMENSIONAL REPRESENTATIONS PERFORMANCE AND INTERPRETATION

To visualize and allow us to interpret the structure of the learned representations, we analyze our hyperbolic PPO agents using only *two dimensions* to model the final latent representations. As mentioned in Section 4 and shown in Table 10, we find even this extreme implementation to provide performance benefits on the test levels over Euclidean PPO. Furthermore, the generalization gap with the training performance is almost null in three out of the four considered environments. As the 2-dimensional representation size greatly constraints the amount of encoded information, this results provides further validation for the affinity of hyperbolic geometry to effectively prioritize features useful for RL. We then observe how these 2-dimensional hyperbolic latent representations evolve within trajectories, mapping them on the Poincaré disk and visualizing the corresponding input states. As summarized in Section 4, we observe a recurring *cyclical* behavior, where the magnitude of the representations monotonically increases within subsets of the trajectory as more obstacles and/or enemies appear. Together with Figure 10 (on the *bigfish* environment), we provide another example visualization of this phenomenon in Figure 12 (on the *starpilot* environment). These plots compare the representations of on-policy states sampled at constant intervals within a trajectory, every 15 timesteps, and deviations from executing 15 timesteps of random behavior after resetting the environment to the previous on-policy

Table 10: Performance of 2-dimensional hyperbolic PPO as compared to the original PPO algorithm.

Task\Algorithm	PPO + S-RYM, 2 dim.	
Levels distribution	*train*	*/test*
bigfish	5.65±4 (+52%)	2.34±3 (+60%)
dodgeball	2.62±0 (-48%)	2.36±1 (+26%)
fruitbot	27.18±4 (-10%)	25.75±1 (-2%)
starpilot	30.27±3 (-1%)	29.72±6 (+12%)

state. We observe the state representations form tree-like branching structures, somewhat reflecting the tree-like nature of MDPs. Within these sub-trajectory structures, we find that the magnitudes in the on-policy trajectory tends to grow in the direction of the Value function's *gyroplane*'s normal. Intuitively, this indicates that as new elements appear (e.g., new enemies in *Starpilot*), the agent recognizes a larger opportunity for rewards (e.g., from defeating them) but requiring a much finer level of control. This is because as magnitude increases, the signed distances with the policy gyroplanes will also grow exponentially, and so will the value of the different action logits, decreasing entropy. In contrast, the magnitudes of the state representations following the random deviations grow in directions with considerably larger orthogonal components to the Value gyroplane's normal. This still reflects the higher precision required for optimal decision-making, as magnitudes still increase, but also the higher uncertainty to obtain future rewards from these less optimal states.

E Further experiments and ablation studies

In this section, we further analyze the properties of our hyperbolic RL framework and its implementation, through additional experiments and ablations. We focus on our hyperbolic PPO algorithm and four representative tasks from the Procgen benchmark.

E.1 S-RYM's components contribution

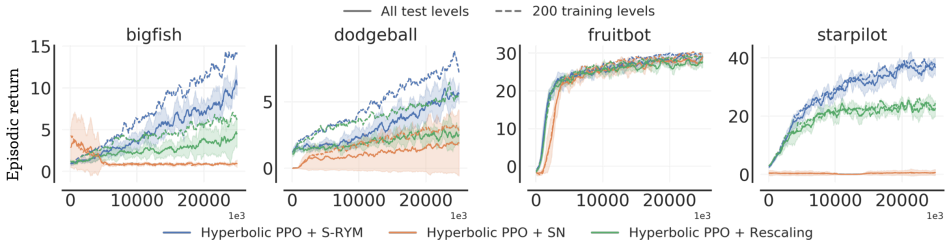

Figure 13: Performance ablating either spectral normalization or rescaling from our Hyperbolic PPO agent stabilized with S-RYM.

Our proposed spectrally-regularized hyperbolic mappings (S-RYM) relies on two main distinct components: spectral normalization and rescaling. As described in Section 3, we design our deep RL models to produce a representation by applying traditional neural network layers in Euclidean space $x_E = f_E(s)$. Before the final linear layer f_H, we then use an exponential map from the origin of the Poincaré to yield a final representation in hyperbolic space $x_H = \exp_0^1(x_E)$. As shown by Lin et al. (2021), applying spectral normalization to the layers of f_E regulates both the values and gradients similarly to LeCun initialization (LeCun et al., 2012). Hence, we make the regularization approximately dimensionality-invariant by rescaling $x_E \in \mathbb{R}^n$, simply dividing its value by $\sqrt{n}$. In Figure 13, we show the results from ablating either component from S-RYM. From our results, both components seem crucial for performance. As removing spectral normalization simply recovers the unregularized hyperbolic PPO implementation with some extra rescaling in the activations, its performance is expectedly close to the underwhelming performance of our naive implementations in Figure 6. Removing our dimensionality-based rescaling appears to have an even larger effect, with almost no agent improvements in 3 out of 4 environments. The necessity of appropriate scaling comes from the influence the representations magnitudes have on optimization. When applying spectral normalization, the dimensionality of the representations directly affects its expected magnitude. Thus, high-dimensional latents will result in high-magnitude representations, making it challenging to optimize for appropriate angular layouts in hyperbolic space (Nickel & Kiela, 2017; Ganea et al., 2018) and making the gradients of the Euclidean network parameters stagnate (Guo et al., 2022). These issues cannot even be alleviate with appropriate network initialization, since the magnitudes of all weights will be rescaled by the intruduced spectral normalization.

E.2 Representation size

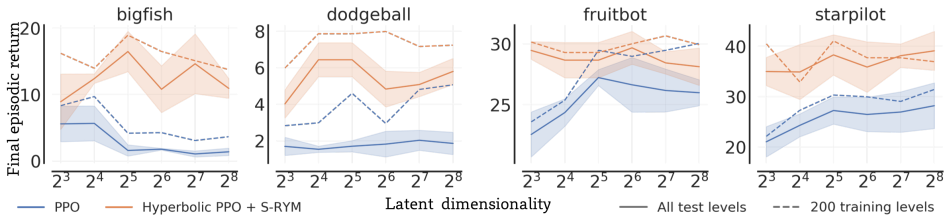

Figure 14: Final performance comparison between PPO agents with Euclidean and hyperbolic representations with different dimensionalities.

In Figure 14, we show the final train and test performance attained by our Euclidean and hyperbolic PPO agents with different dimensionalities for their final latent representations. We collect results on a log scale 2^n with $n \in \{3, 4, 5, 6, 7, 8\}$, i.e., ranging from $2^3 = 8$ to $2^8 = 256$ latent dimensions. Integrating our hyperbolic representations framework with PPO boosts performance across all dimensionalities. Moreover, in 3/4 environments we see both train and test performance of the Euclidean PPO agent considerably dropping as we decrease the latent dimensions. In contrast, the performance of hyperbolic PPO is much more robust, even attaining some test performance gains from more compact representations. As described in Section 2, Euclidean representations require high dimensionalities to encode hierarchical features with low distortion (Matoušek, 1990; Gupta, 1999), which might explain their diminishing performance. Instead, as hyperbolic representations do not have such limitation, lowering the dimensionality should mostly affect their ability of encoding non-hierarchical information, which we believe to counteract the agent's tendency of overfitting to the limited distribution of training levels and observed states.

E.3 Compatibility with orthogonal practices

Introducing hyperbolic geometry to model the representations of RL agents is fundamentally orthogonal to most recent prior advances. Thus, we validate the compatibility of our approach with different methods also aimed at improving the performance and generalization of PPO.

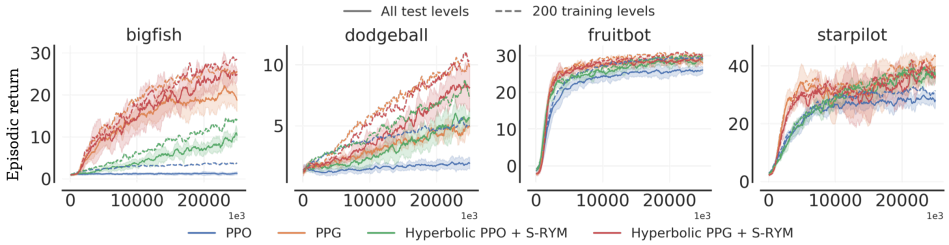

Figure 15: Performance comparison from integrating the advances from the PPG algorithm our hyperbolic reinforcement learning framework.

Phasic Policy Gradient (PPG). We re-implement this recent PPO extension designed by Cobbe et al. (2021) specifically for the Procgen benchmark. PPG adds non-trivial algorithmic and computational complexity, by performing two separate optimization phases. In the first phase, it optimizes the same policy and value optimization objective as in PPO, utilizing the latest on-policy data. In the second phase, it utilizes a much larger buffer of past experience to learn better representations in its policy model via an auxiliary objective, while avoiding forgetting with an additional behavior cloning weighted term. The two phases are alternated infrequently after several training epochs. Once again, we incorporate our hyperbolic representation framework on top of PPG without any additional tuning. In Figure 15, we show the results from adding our deep hyperbolic representation framework to PPG. Even though PPG's performance already far exceeds PPO, hyperbolic representations appear to have similar effects on the two algorithms, with performance on the 200 training

levels largely invaried, and especially notable test performance gains on the bigfish and dodgeball environments. Hence, in both PPO and PPG, the new prior induced by the hyperbolic representations appears to largely reduce overfitting to the observed data and achieve better generalization to unseen conditions. Our approach affects RL in an orthogonal direction to most other algorithmic advances, and our results appear to confirm the general compatibility of its benefits.

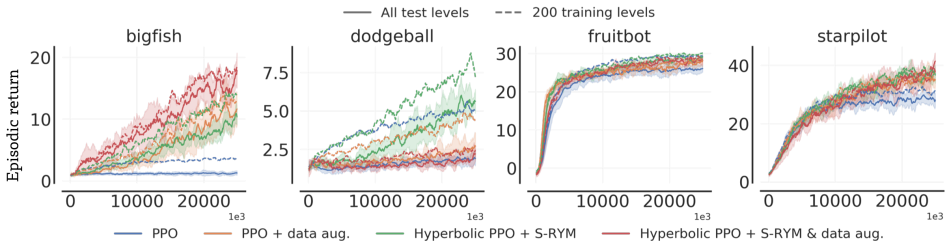

Figure 16: Performance comparison from integrating data augmentation with the Euclidean and hyperbolic PPO agents.

Data augmentation. Finally, we also test introducing data augmentation to our Hyperbolic PPO implementation. We consider the same popular random shifts from Yarats et al. (2021a), evaluated in Section 4. We note that the problem diversity characterizing procgen makes it challenging for individual hand-designed augmentations to have a generally beneficial effect, with different strategies working best in different environments (Raileanu et al., 2020). In fact, applying random shifts to PPO appears to even hurt performance on a considerable subset of environments (see Table 1), likely due to the agents losing information about the exact position and presence of key objects at the borders of the environment scene. This inconsistency is reflected onto the hyperbolic PPO agent. In particular, while the addition of random shifts further provides benefits on the bigfish environment, it appears to hurt performance on dodgeball. Overall, integrating our hyperbolic framework still appears considerably beneficial even for the test performance of the data-augmented agent, further showing the generality of our method.