2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

HyperPPO: A scalable method for finding small policies for robotic control

Shashank Hegde, Zhehui Huang and Gaurav S. Sukhatme
University of Southern California

***Abstract*— Models with fewer parameters are necessary for the neural control of memory-limited, performant robots. Finding these smaller neural network architectures can be time-consuming. We propose HyperPPO, an on-policy reinforcement learning algorithm that utilizes graph hypernetworks to estimate the weights of multiple neural architectures simultaneously. Our method estimates weights for networks that are much smaller than those in common-use networks yet encode highly performant policies. We obtain multiple trained policies at the same time while maintaining sample efficiency and provide the user the choice of picking a network architecture that satisfies their computational constraints. We show that our method scales well - more training resources produce faster convergence to higher-performing architectures. We demonstrate that the neural policies estimated by HyperPPO are capable of decentralized control of a Crazyflie2.1 quadrotor. Website: https://sites.google.com/usc.edu/hyperppo**

I. Introduction

A common practice in robot learning (particularly deep reinforcement learning) is to fix a network size and architecture and train it to approximate the near-optimum policy for a given task. For locomotion tasks with only proprioceptive sensing, networks of $\sim$ 256 neurons and $\sim$ 3 layers are commonly employed [1], while for exteroceptive sensing, the configuration of the network varies with the data modality [2]. For tasks that require the neural network controller to be deployed onto a real robot, especially one with memory and computational constraints such as the Crazyflie2.1, with which we experiment here (192Kb of onboard RAM) [3], the choice of network size and architecture is of paramount importance.

There has been significant recent progress in neural architecture search (NAS) [4]. However, this has not focused on applications to neural robotic control. The problem of finding small yet performant neural networks for robot control is further exacerbated by the fact that performance and size of neural networks are not directly correlated [5]. Here, we build on the approach in [5] and present a method (Figure 1) that trains thousands of architecturally unique neural control policies simultaneously. We give the user the ability to choose an architecture that fits within their computation constraints and meets their performance requirements. We note that post-training, the weights for any chosen architecture can be estimated in one forward pass of our trained model.

khegde|zhehuihu|gaurav@usc.edu
GSS holds concurrent appointments as a Professor at USC and as an Amazon Scholar. This paper describes work performed at USC and is not associated with Amazon.

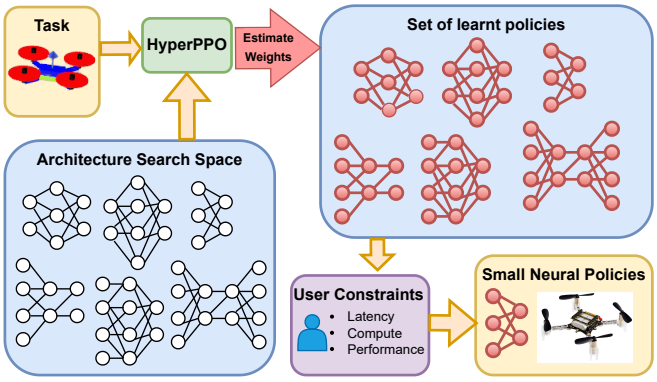


Fig. 1: For a given task and a large architecture search space, HyperPPO learns to estimate weights for multiple architectures simultaneously. The user can choose an architecture based on their performance requirements and computational constraints from the set of learned policies.

Contributions: The method proposed in [5] is off-policy. Such methods tend to be sample-efficient yet time-inefficient in training (when one measures wall-clock training time). Here we present an on-policy method (HyperPPO) that simultaneously produces thousands of policies, each with a unique architecture. HyperPPO has sample efficiency similar to one training run of regular proximal policy optimization (PPO) and results in unique performant policies for each architecture. We propose two versions of HyperPPO: with vectorized standard deviations (HyperPPO-VSD), suitable for the setting when training data are abundant and a fast simulator is available, and with common standard deviation (HyperPPO-CSD), suitable in the setting when gathering data is harder. We analyze and ablate the trade-offs of each version. We benchmark HyperPPO-VSD on GPU accelerated environments and HyperPPO-CSD on the quadrotor simulator, QuadSwarm [6]. We show that small networks estimated by HyperPPO-VSD are capable of outperforming the same networks obtained by training with regular PPO. We also show that the weights estimated by HyperPPO-CSD for a tiny neural network (just one hidden layer with 4 neurons) can be successfully deployed on a Crazyflie2.1 for autonomous flight control.

II. Related Work

A. Proximal Policy Optimization (PPO)

PPO is a widely adopted on-policy learning algorithm [7]. As opposed to off-policy learning algorithms, PPO provides

separate loops for sample collection and training. This separation allows for massive parallelization, which provides trained policies more quickly. Further, PPO has been shown to have better stability. The governing equations of PPO are as follows.

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_k}\left(a_t \mid s_t\right)}$$

$$\hat{A}_t^{\pi_\theta} = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

$$\text{where } \delta_t = r_t + \gamma V^{\pi_\theta}\left(s_{t+1}\right) - V^{\pi_\theta}\left(s_t\right)$$

$$\mathcal{L}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\min\left(r_t(\theta)\hat{A}_t^{\pi_{\theta_k}}, \mathrm{clip}\left(r_t(\theta), 1\pm\epsilon\right)\hat{A}_t^{\pi_{\theta_k}}\right)\right]$$

$r_t(\theta)$ is the importance sampling ratio function between the policy that is used to collect data and the k'th version of the policy. $V^{\pi_\theta}\left(s_t\right)$ is the value function estimated by the critic for the policy π_θ at the state s_t.The generalized advantage estimate is given by $\hat{A}_t^{\pi_\theta}$. Finally, $\mathcal{L}_{\theta_k}(\theta)$ is the clipped loss objective.

Off-policy methods tend to be slower than on-policy methods, as the latter can be optimized easily. Further, on-policy methods have fewer hyperparameters and can have higher convergence stability if we have sufficient environment instances [8]. Optimizations needed to improve the performance of PPO are documented in [9]. A benefit of using PPO is the ability to scale with more computational resources. The availability of highly parallelized environments [10] and GPU-based physics engines [11], [12], have been shown to work well with PPO [13], [10]. For exploration, PPO generally samples its actions from a stochastic policy. The mean is obtained as the output from a parameterized state-conditioned network. The standard deviation is obtained either with another state-conditioned network or is simply characterized as a (non-state-conditioned) array whose values are directly modified during training. Here, we will consider the later version.

B. Neural Architecture Search

Neural architecture search [4] is the process of searching for an optimal neural architecture for a given task. While reinforcement learning has been used for NAS [14], the use of NAS for reinforcement learning-based policies is still an under-explored area. NAS has tremendous opportunities in robotic control as on-board compute size poses an architecture search constraint.

Differentiable Architecture Search (DARTS) [15] is a machine learning technique used to automate the process of finding optimal neural network architectures for tasks by introducing a continuous relaxation of the discrete architecture space, allowing gradient-based optimization methods to be used. In [16] DARTS was used for reinforcement learning policies. In [17] a differentiable approach was used for architecture search for robotic learning - the first to deploy a NAS-based neural controller on a robot. Efficient Neural Architecture Search (ENAS) [14] optimizes the architecture search process by sharing parameters across child models, reducing the computational overhead of evaluating multiple architectures. [18] and [19] utilize ENAS to find the best-performing architecture for RL tasks.

Another family of methods in NAS is one-Shot Model Architecture Search through Hypernetworks (SMASH) [20]. A primary network (hypernetwork [21]) is trained to estimate the optimal weights for a variable architecture secondary network. Once this hypernetwork is trained, the optimal weights for all architectures in a search space can be estimated, and the one with the best objective can be chosen. The idea of Graph Hypernetworks (GHN) was introduced in [22]. The computational graph of an architecture is provided as input, and common message-passing techniques akin to those found in GNNs are used to generate the weights of that architecture as its output. GHN benchmarking against other DARTS and ENAS methods shows that it only uses a fraction of the search cost associated with other NAS methods. Following this [23] introduces GHN2, which employs a gated graph network for better generalization of the hypernetwork. Hypernetworks have been studied for learning dynamics [24], continual learning [25], and online policy adaptation [26], but their application for variable policy architectures remains under-studied.

[5] introduced Graph Hyper Policies (GHP) that utilized a GHN to estimate the weights of robotic policies for manipulation and locomotion. This was done using off-policy reinforcement learning, specifically, Soft Actor critic [27] for locomotion and Hindsight Experience Replay[28] with Deep deterministic policy gradients [29] for manipulation. For a given architecture graph representation of a network g, this network, h_θ, can estimate the policy $\pi_\phi = h_\theta(g)$, where the estimated weights are ϕ. It was also shown in [5] that directly estimated weights of smaller policies were more performant than policies of the same architecture obtained by behavior cloning based distillation methods. Since these methods are off-policy, they are extremely sample efficient and can learn to estimate weights for multiple policies with the same number of samples as it would be to learn for a single architecture. A drawback for this method though is that it is not time efficient. As noted in the paper, this method had a $\sim$ 5x training time increase. This can amount to a large amount of time considering that off-policy methods are already time inefficient as compared to on-policy methods. Further, this method does not scale well with more compute resources as data collection is not a bottleneck for Q learning. From a constraint architecture search point of view, searching for architectures for robotic control, hypernetwork-based methods are an alluring option as having multiple options during deployment would reduce experimentation time drastically.

C. Deep Reinforcement Learning for Quadrotor Control

There is significant recent work in the control of quadrotors with direct rotor thrusts by using deep reinforcement learning (DRL). [30] investigates stabilizing a quadrotor with hash initialization, and a neural network policy with two hidden layers with 64 neurons in each layer. [31] can train control policies with minimal prior knowledge about a quadrotor's dynamics parameters and can transfer a single control policy to multiple quadrotor platforms with two

hidden layers with 64 neurons in each layer. [32] uses model-based DRL for the hover control of a quadrotor (up to 6 seconds with 3 minutes of training data with 2 hidden layers with 250 neurons in each layer). [33] proposes control policies that can achieve 60 km/h on a physical quadrotor by using 2 hidden layers with 128 neurons in each layer. [34] and [35] use DRL to design decentralized control policies that can fly quadrotor swarms in various scenarios with significant collision avoidance ability in the real world with two encoders, both consisting of 2 hidden layers, with only 16 and 8 neurons, respectively.

For agile tasks, it is desirable for neural network inference to have lower latency than sensing. This can become an issue when the sensing modality is complex (such as vision) or goal conditioning needs a larger encoder (such as language). For agile flight control of a quadrotor, [36] utilize a RealSense D435i camera for depth sensing, which runs at 30 Hz while their network inference on an onboard NVIDIA Jetson TX2 runs at 25 Hz.

III. Method

A. Multi Architecture Proximal Policy Optimization

The method proposed in [5] is off-policy. Such methods tend to be sample efficient, yet time-inefficient in training. To find an on-policy version of [5], as a first cut, we ran PPO where the policy is replaced with a graph hyper policy estimating policies for randomly sampled architectures, on the halfcheetah environment [37]. This setup is similar to [5] but with PPO instead of Soft Actor Critic [27]. As the model trained, we evaluated it on a fixed set of architectures. We observed that for all architectures, the policies estimated by the graph hyper policy reach the same reward and collapse to a single policy. This is because PPO, being an on-policy algorithm, cannot effectively use data obtained from one architecture to estimate weights for a different architecture. This becomes evident on inspecting the equations for PPO from II-A.

Let us denote the entire search space of architectures by U. Let the sampled architectures from this space be $g \sim \mathrm{U}$. In order to use the PPO algorithm for multi-architecture training, we need to substitute $\pi_\theta \leftarrow h_\theta(g)$ in these equations, where h_θ is a graph hypernetwork parameterized by θ, which estimates the weights for architecture g. Doing so results in the following equations:

$$
\begin{aligned}
r_t(\theta, g) &= \frac{h_\theta\left(a_t \mid s_t, g\right)}{h_{\theta_k}\left(a_t \mid s_t, g\right)} \\
\hat{A}_t^{h_\theta(g)} &= \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\
\text{where } \delta_t &= r_t + \gamma V^{h_\theta}\left(s_{t+1}, g\right) - V^{h_\theta}\left(s_t, g\right) \\
\mathcal{L}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\substack{g\sim\mathrm{U} \\ \tau\sim h_\theta(g)}} &\left[\min\left(\begin{matrix} r_t(\theta, g)\hat{A}_t^{h_{\theta_k}(g)}, \\ \mathrm{clip}\left(r_t(\theta, g), 1 \pm \epsilon\right)\hat{A}_t^{h_{\theta_k}(g)}\end{matrix}\right)\right]
\end{aligned}
$$

We see that the importance sampling ratio, advantage estimate, and the value function, are all now conditioned on the current policy's architecture. Since the architecture remains g while estimating all the above values, no mixing of data between architectures must happen.

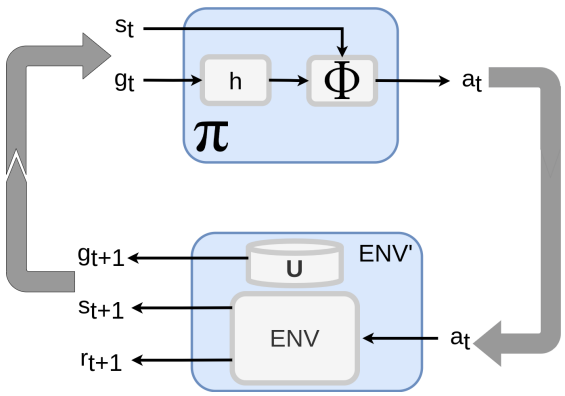


Fig. 2: Architecture and State concatenated Markov Decision Process. By augmenting the architecture into the MDP state space, we can train RL agents with varying architecture.

B. Intuition

Another way of visualizing the above formulation is by restructuring the underlying Markov Decision Process. We concatenate the randomly sampled architecture graph into the state variable. As shown in figure 2, this allows us to reformulate the policy as the actions sampled from the policy estimated by the hypernetwork for that given combination of graph and state variables. The concatenation of the state and architecture can be seen while estimating the GAE $\hat{A}_t^{h_\theta(g)}$, specifically while estimating the state value function $V^{h_\theta}\left(s_t, g\right)$. Practically, we condition the critic network of PPO with state and architecture and make sure we use the same architecture's data for the Bellman update.

C. Algorithm

Based on these changes we propose HyperPPO. As shown in Algorithm 1, for a given task we start with a predefined architecture space U. For every iteration of the algorithm, we sample architecture g_i from the search space. For this work, we restrict the search to the architecture space of Multi Layer Perceptrons (MLPs). Our architecture search space U consists of all possible MLPs with four or fewer layers. that can be constructed with the number of neurons in each layer being $\{4, 8, 16, 32, 64, 128, 256\}$. This gives us 2800 unique architectures. We use the same graph hyper policy model as in [5] and estimate policy π_{ϕ_i} for that architecture. We then collect data $\{\mathcal{D}_k\}_i$ using this policy. Using this data we estimate GAE $\hat{A}_t^{h_{\theta_k}(g_i)}$ and the ratio function $r_t(\theta, g_i)$. This process can be parallelized for a meta batch size of architectures for faster computation. Using these estimates, we then use SGD to optimize the objective $\mathcal{L}_{\theta_k}$ over the hypernetwork weights θ.

Just like regular PPO for continuous action spaces, actions are sampled from a Gaussian distribution. The mean of the distribution is obtained using the policies estimated by the graph hyper network. For standard deviations, we propose two approaches, which lead to two versions of our

method. HyperPPO-VSD (Vectorized Standard Deviations) constructs a vector of standard deviation arrays, one for each architecture. This enables independent exploration for all architectures. HyperPPO-CSD (Common Standard Deviation) uses a common standard deviation array for all architectures. This reduces computation and converges faster.

For our method, we utilize vectorized environments. These environments enable parallelization and allow us to sample data for different architectures simultaneously. The larger the number of environments we can run in parallel the better our estimates should be for our objective functions.

Algorithm 1 HyperPPO

input: Initial Hypernetwork parameters θ_0.
input: Clipping threshold ϵ.
input: Architecture Search space U, Meta-batch size M.
for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots M$ **do**
    Sample architecture $g_i \sim \mathrm{U}$
    Estimate Policies $\pi_{\phi_i} \leftarrow h_{\theta_k}(g_i)$
    Collect trajectories $\{\mathcal{D}_k\}_i$ using policy π_{ϕ_i}
    Estimate GAE $\hat{A}_t^{h_{\theta_k}(g_i)}$
    Estimate importance sampling ratio $r_t(\theta, g_i)$
  end for
  Compute policy update
    $\theta_{k+1} = argmax_\theta \mathcal{L}_{\theta_k}(\theta)$
  by taking K steps of minibatch SGD (via Adam)
end for

IV. Results and Discussion

To implement our method, we use the Sample Factory [38] package. Its efficient design enables us to parallelize data collection and train Graph Hyper Policies quickly. The experiments are carried out on standard locomotion tasks that have been implemented on Brax [11] and Mujoco [39]. We also train on the quadrotor simulator described in QuadSwarm [6]. All experiments were run 4 seeds at a time on an AWS g4dn.12xlarge instance with 48vCPU, 4 Telsa T4 GPUs and 192 GB RAM.

A. Ablations

For our ablations, we train on the Humanoid task in Brax for 1 billion steps for 8 seeds. We simulated 4096 environment instances in parallel and ran for approximately 200 minutes. Every few steps, we evaluate the performance of policies estimated by the GHP for every architecture in the search space. To estimate the quality of all architectures we find the average reward across all architectures.

1) Vectorized Standard deviations: First, we analyze the performance of HyperPPO with VSD and CSD. Figure 4 shows this for both CSD and VSD. We see that with CSD, the average reward grows faster initially. This is because the standard deviation converges faster with CSD. But with more training, we see that VSD eventually achieves a larger reward. As mentioned in IV-A.1, we believe this is because individual exploration for each architecture can eventually obtain better performance. Therefore we suggest using the VSD when massively parallel environments such as Brax or IsaacGym [12] are available.

2) Architecture Sampling: During experimentation, we first implemented the uniform architecture sampling as described in [5]. On further analysis, we found that the graph hyper policy has a learning bias toward deeper network architectures. We believe this is because there are fewer shallower architectures than deeper ones. To compensate for this effect, we sample architectures with their sampling probability inversely proportional to the number of layers. We shall call this biased sampling.

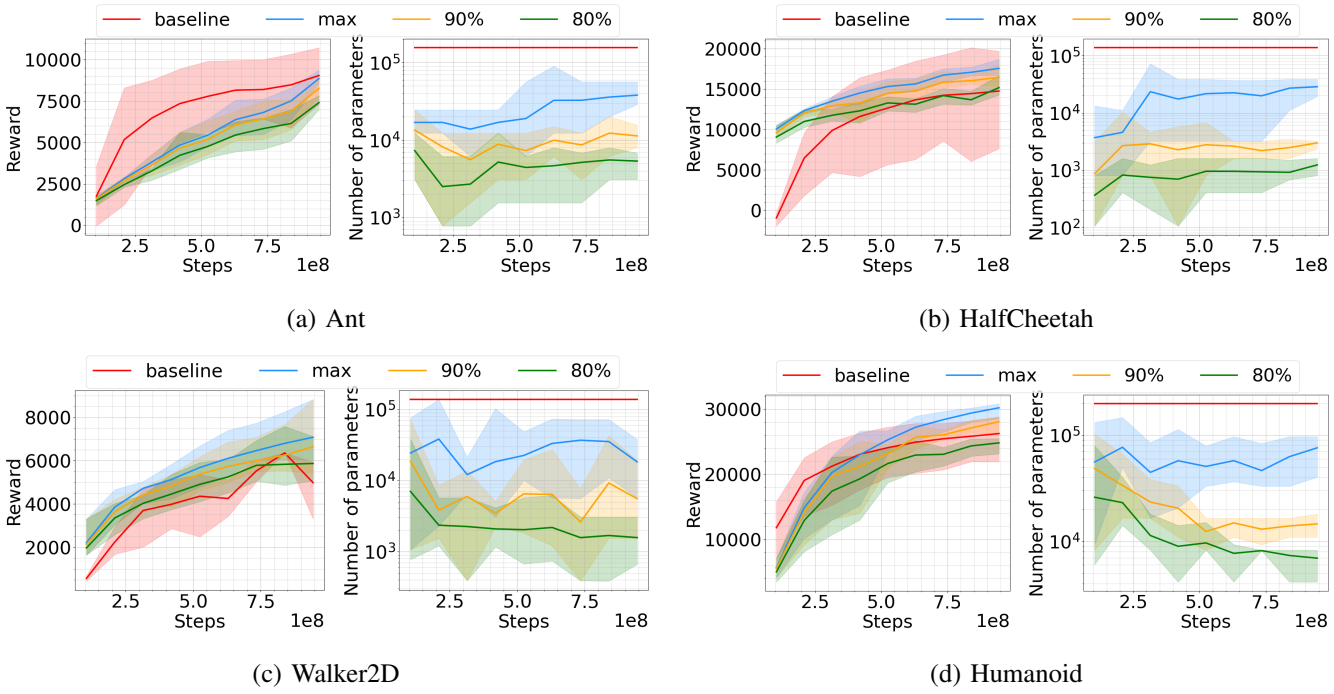
We run HyperPPO-VSD with both modes of architecture sampling. From figure 4, we can see that with biased sampling, we obtain better performance. Further, smaller networks gained a bigger performance boost with biased sampling, since more of these were considered during training. We see similar performance differences between these ablations in the Brax HalfCheetah task as well. Therefore, for all other experiments in this paper, we set the architecture sampling mode to biased sampling.

B. Scaling HyperPPO

Here, we show that HyperPPO can scale up to provide better results with more computation. We train HyperPPO-CSD on the Mujoco halfcheetah task for 5 hours while varying the number of environment instances from which data are sampled. We run this experiment over 5 seeds, and at the end of the experiment, we evaluate every architecture in the search space. Figure 5 shows us the distribution of performance over all unique architecture policies estimated by GHP. This plot is similar to those used to evaluate policy data sets in [40], [41]. The x-axis is the policy's accumulated reward, while the y-axis represents the number of policies with reward greater than x. N represents the number of environment instances from which data are sampled. We can see that scaling up the algorithm with more parallel environments in HyperPPO with more computation can provide a better collection of policies over the same time.

C. Brax benchmarks

Having shown that our method scales with performance, we benchmark HyperPPO-VSD on GPU-accelerated Brax environments. We use 4 locomotion tasks, namely, humanoid, ant, halfcheetah, and walker2d. On each task, we train for 1 billion state transition steps and show results across 8 seeds. During training, every few steps, we evaluate the GHP on every architecture in the search space. From this evaluation, we identify architectures that provided the highest reward, the smallest architectures that provided 90% of the highest reward, and the smallest architectures that provided 80% of the highest reward. We call these max, 90%, and 80% architectures respectively. As a baseline, we train regular PPO also implemented on Sample Factory with the same hyperparameters, with 3 hidden layers with 256 neurons each. This is a common choice of model architecture for these locomotion tasks. Figure 3 shows the



(a) Ant

(b) HalfCheetah

(c) Walker2D

(d) Humanoid

Fig. 3: **Learning smaller networks.** All architectures are evaluated as training progresses. For each pair, **left**: (max performance, 90% of max performance, 80% of max performance, baseline performance) vs training samples collected; **right**: the minimum number of parameters needed to achieve these levels of performance vs training samples collected.

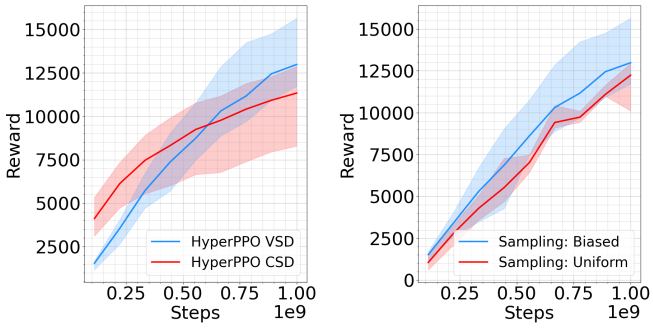


Fig. 4: **Ablations.** Average reward across all architectures during training. **Left:** Action Standard Deviation; **Right**: Architecture Sampling.

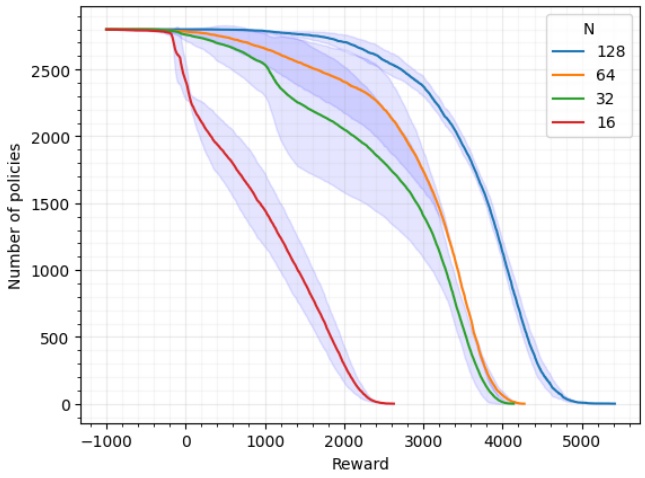


Fig. 5: **Scaling HyperPPO**: The X-axis corresponds to a total reward, and the Y axis shows the number of architectures that at least achieves x reward. With more environment instances, the performance of all architectures increases. N represents the number of parallel environment instances.

results of this experiment. For each task, the left plot depicts rewards attained by the max, 90%, 80% architectures, and the baseline. The right plot shows the size of these architectures on a log scale. For all tasks, we see that the number of parameters required to achieve 90% and 80% of maximum performance reduces considerably.

Further, by taking the average reward over all seeds, we identify 80% architectures for each task as (64) for halfcheetah, (64) for walker2d, (32) for humanoid, and (64) for Ant. These are all single hidden layer architectures with either 64 or 32 neurons in them. We trained policies with these architectures with regular PPO and compared their performance with policies of the same architectures estimated by the GHP in HyperPPO-VSD. Table I shows that the policies estimated by the GHP obtain considerably more reward on the Halfcheetah, Walker2d, and Humanoid tasks, while the performance is comparable on the Ant task, figure 3 suggests that the model has not converged for Ant.

These results show that HyperPPO-VSD can provide multiple architecture policies with the same sample complexity as a single PPO run, and further provides higher performing smaller policies than its regular PPO counterparts. We believe this increase in performance has two reasons: (a)

Task	80% Architecture	PPO (x10^2)	HyperPPO (x10^2)
Halfcheetah	[64]	80.30 ± 49.23	$\mathbf{144.80 \pm 13.36}$
Walker2D	[64]	19.84 ± 7.18	$\mathbf{58.50 \pm 6.64}$
Humanoid	[32]	182.85 ± 25.35	$\mathbf{207.69 \pm 49.12}$
Ant	[64]	71.88 ± 11.46	70.49 ± 8.85

TABLE I: Comparison of small policies

Better exploration: The policies are now more stochastic with HyperPPO-VSD probabilistically choosing different action distributions during data collection. (b) Distillation between architectures: Gradients to the hypernetwork from data of larger architectures can improve policies estimated for smaller architectures.

D. Quadrotor Drones

We train HyperPPO-CSD on the Quadrotor environment designed for a Crazyflie 2.1, QuadSwarm [6]. The Crazyflie 2.1 is a severely compute-constrained quadrotor with an onboard microcontroller running at 168MHz with 168 Kb RAM. We train the control policy in simulation on a mixture of single drone goal-based scenarios [34] (static goal, dynamic goal, random 3D Lissajous trajectory tracking, and random 3D Bezier curve trajectory tracking), for 500 million state transition steps, and we zero-shot transfer our control policy to the physical Crazyflie quadrotor. We test our control policies on the Bezier curve trajectory tracking on the physical Crazyflie quadrotor, one of the most challenging scenarios in the simulation, to showcase the flying performance of our control policy. As a baseline, we train a policy with architecture (512,512) (i.e., two hidden layers with 512 neurons each), with the same hyperparameters and scenarios. Similar to Figure 3, we analyze the training performance in Figure 6. We see that the best-performing architecture estimated with HyperPPO-CSD achieves more reward than the baseline, whose performance is comparable to that of 80% architectures. Across seeds, for this task, we identified the 80% architecture as (4) (i.e., a single hidden layer 4 neuron network). This small policy was estimated at the end of training and deployed on the Crazyflie. For evaluating the physical deployment performance, we generate a random 3D Bezier curve as the desired trajectory and use the neural network to control rotor thrusts, to track this trajectory. From Figure 7 we see that the quadrotor is capable of tracking the desired trajectory with a HyperPPO estimated neural network, with high success rates. If we wanted to test a different architecture for physical deployment, instead of retraining a new network from scratch, we can estimate the weights for that architecture with one inference step of the trained GHP model.

While we maintain sample efficiency, we note that a limitation of our method is a $\sim$ 2-3x training time increase as compared to regular PPO. At present, we limit ourselves to Multi-Layer Perceptrons, however, we plan to experiment with architecture search spaces with different types of networks such as CNNs, LSTM, and Transformers in the future. Finally, identifying the performance of a candidate

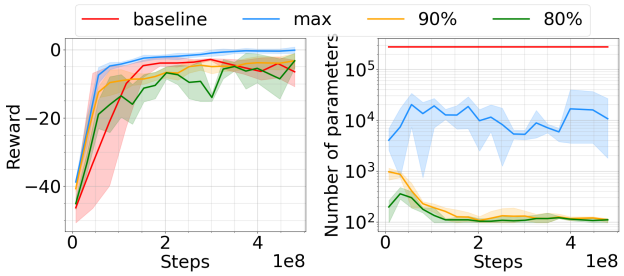


Fig. 6: Analysis of Quadrotor Training

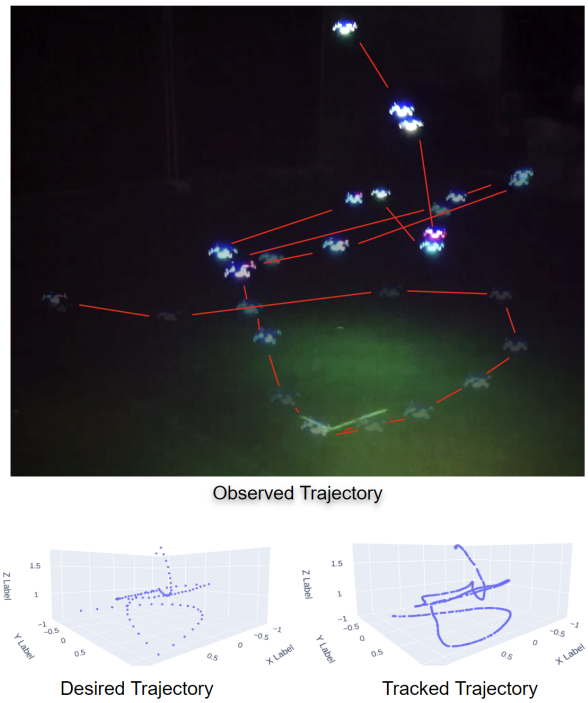


Fig. 7: Evaluating a single layer 4 neuron network estimated by HyperPPO-CSD on the Crazyflie2.1. **Left:** The desired trajectory created with a random bezier curve. **Right:** Actual trajectory of the drone. **Top:** Frames stacks of the actual footage of drone flight.

architecture involves estimating it with the GHP and evaluating it with a rollout. Identifying the desired architecture algorithmically during training is a possible future avenue.

V. Conclusion and Future Work

We present HyperPPO, an on-policy algorithm that learns multiple architecture policies simultaneously. We show that the algorithm is fast, sample efficient, and scales with added computation. We provide two versions: HyperPPO-VSD, which can be used when data collection is accelerated; and HyperPPO-CSD, which can be used when computation is limited and for faster convergence. We show that on Brax benchmarks, HyperPPO-VSD can quickly estimate thousands of working policy architectures, and the estimated small policies outperform PPO on most tasks. Finally, we show that small policies estimated by HyperPPO-CSD can be successfully deployed on an actual compute-constrained platform - the Crazyflie - for neural control.

REFERENCES

[1] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning." in *Robotics: Science and Systems*, 2019.

[2] W. Yu, D. Jain, A. Escontrela, A. Iscen, P. Xu, E. Coumans, S. Ha, J. Tan, and T. Zhang, "Visual-locomotion: Learning to walk on complex terrains with vision," in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Faust, D. Hsu, and G. Neumann, Eds., vol. 164. PMLR, 08–11 Nov 2022, pp. 1291–1302. [Online]. Available: https://proceedings.mlr.press/v164/yu22a.html

[3] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2017, pp. 37–42.

[4] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural architecture search: Insights from 1000 papers," *arXiv preprint arXiv:2301.08727*, 2023.

[5] S. Hegde and G. S. Sukhatme, "Efficiently learning small policies for locomotion and manipulation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5909–5915.

[6] Z. Huang, S. Batra, T. Chen, R. Krupani, T. Kumar, A. Molchanov, A. Petrenko, J. A. Preiss, Z. Yang, and G. S. Sukhatme, "Quadswarm: A modular multi-quadrotor simulator for deep reinforcement learning with direct thrust control," *arXiv preprint arXiv:2306.09537*, 2023.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[8] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," in *ICLR 2021-Ninth International Conference on Learning Representations*, 2021.

[9] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *ICLR Blog Track*, 2022, https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[10] J. Weng, M. Lin, S. Huang, B. Liu, D. Makoviichuk, V. Makoviychuk, Z. Liu, Y. Song, T. Luo, Y. Jiang *et al.*, "Envpool: A highly parallel reinforcement learning environment execution engine," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 409–22 421, 2022.

[11] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "Brax-a differentiable physics engine for large scale rigid body simulation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[12] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu based physics simulation for robot learning," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[13] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *5th Annual Conference on Robot Learning*, 2021.

[14] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2016.

[15] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[16] Y. Miao, X. Song, J. D. Co-Reyes, D. Peng, S. Yue, E. Brevdo, and A. Faust, "Differentiable architecture search for reinforcement learning," in *Proceedings of the First International Conference on Automated Machine Learning*, ser. Proceedings of Machine Learning Research, I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, Eds., vol. 188. PMLR, 25–27 Jul 2022, pp. 20/1–17. [Online]. Available: https://proceedings.mlr.press/v188/miao22a.html

[17] I. Akinola, A. Angelova, Y. Lu, Y. Chebotar, D. Kalashnikov, J. Varley, J. Ibarz, and M. S. Ryoo, "Visionary: Vision architecture discovery for robot learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 10 779–10 785.

[18] X. Song, K. Choromanski, J. Parker-Holder, Y. Tang, W. Gao, A. Pacchiano, T. Sarlos, D. Jain, and Y. Yang, "Reinforcement learning with chromatic networks for compact architecture search," *arXiv preprint arXiv:1907.06511*, 2019.

[19] N. Mazyavkina, S. Moustafa, I. Trofimov, and E. Burnaev, "Optimizing the neural architecture of reinforcement learning agents," in *Intelligent Computing*, K. Arai, Ed. Cham: Springer International Publishing, 2021, pp. 591–606.

[20] A. Brock, T. Lim, J. M. Ritchie, and N. J. Weston, "Smash: One-shot model architecture search through hypernetworks," in *6th International Conference on Learning Representations 2018*, 2018.

[21] D. Ha, A. M. Dai, and Q. V. Le, "Hypernetworks," *CoRR*, vol. abs/1609.09106, 2016. [Online]. Available: http://arxiv.org/abs/1609.09106

[22] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," 2019, publisher Copyright: © 7th International Conference on Learning Representations, ICLR 2019. All Rights Reserved.; 7th International Conference on Learning Representations, ICLR 2019 ; Conference date: 06-05-2019 Through 09-05-2019.

[23] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. Romero Soriano, "Parameter prediction for unseen deep architectures," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 433–29 448, 2021.

[24] Z. Xian, S. Lal, H.-Y. Tung, E. A. Platanios, and K. Fragkiadaki, "Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=pHXfe1cOmA

[25] J. von Oswald, C. Henning, B. F. Grewe, and J. Sacramento, "Continual learning with hypernetworks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJgwNerKvB

[26] M. Xu, Y. Lu, Y. Shen, S. Zhang, D. Zhao, and C. Gan, "Hyper-decision transformer for efficient online policy adaptation," 2023.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[28] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR (Poster)*, 2016.

[30] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[31] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 59–66.

[32] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[33] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1205–1212.

[34] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme, "Decentralized control of quadrotor swarms with end-to-end deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 576–586.

[35] Z. Huang, Z. Yang, R. Krupani, B. Şenbaşlar, S. Batra, and G. S. Sukhatme, "Collision avoidance and navigation for a quadrotor swarm using end-to-end deep reinforcement learning," *arXiv preprint arXiv:2309.13285*, 2023.

[36] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, pp. 3–26, 2021.

[37] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[38] A. Petrenko, Z. Huang, T. Kumar, G. S. Sukhatme, and V. Koltun, "Sample factory: Egocentric 3d control from pixels at 100000

FPS with asynchronous reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 7652–7662. [Online]. Available: http://proceedings.mlr.press/v119/petrenko20a.html

[39] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[40] S. Batra, B. Tjanaka, M. C. Fontaine, A. Petrenko, S. Nikolaidis, and G. Sukhatme, "Proximal policy gradient arborescence for quality diversity reinforcement learning," *arXiv preprint arXiv:2305.13795*, 2023.

[41] S. Hegde, S. Batra, K. Zentner, and G. S. Sukhatme, "Generating behaviorally diverse policies with latent diffusion models," *arXiv preprint arXiv:2305.18738*, 2023.