Leveraging Procedural Generation to Benchmark Reinforcement Learning

Karl Cobbe [1] **Christopher Hesse** [1] **Jacob Hilton** [1] **John Schulman** [1]

Abstract

We introduce Procgen Benchmark, a suite of 16 procedurally generated game-like environments designed to benchmark both sample efficiency and generalization in reinforcement learning. We believe that the community will benefit from increased access to high quality training environments, and we provide detailed experimental protocols for using this benchmark. We empirically demonstrate that diverse environment distributions are essential to adequately train and evaluate RL agents, thereby motivating the extensive use of procedural content generation. We then use this benchmark to investigate the effects of scaling model size, finding that larger models significantly improve both sample efficiency and generalization.

1. Introduction

Generalization remains one of the most fundamental challenges in deep reinforcement learning. In several recent studies (Zhang et al., 2018c; Cobbe et al., 2019; Justesen et al., 2018; Juliani et al., 2019), agents exhibit the capacity to overfit to remarkably large training sets. This evidence raises the possibility that overfitting pervades classic benchmarks like the Arcade Learning Environment (ALE) (Bellemare et al., 2013), which has long served as a gold standard in RL. While the diversity between games in the ALE is one of the benchmark's greatest strengths, the low emphasis on generalization presents a significant drawback. Previous work has sought to alleviate overfitting in the ALE by introducing sticky actions (Machado et al., 2018) or by embedding natural videos as backgrounds (Zhang et al., 2018b), but these methods only superficially address the underlying problem — that agents perpetually encounter near-identical states. For each game the question must be asked: are agents robustly learning a relevant skill, or are they approximately memorizing specific trajectories?

There have been several investigations of generalization in RL (Farebrother et al., 2018; Packer et al., 2018; Zhang et al., 2018a; Lee et al., 2019), but progress has largely proved elusive. Arguably one of the principal setbacks has been the lack of environments well-suited to measure generalization. While previously mentioned studies (Zhang et al., 2018c; Cobbe et al., 2019; Justesen et al., 2018; Juliani et al., 2019) reveal intriguing trends, it is hard to draw general conclusions from so few environments.

We seek the best of both worlds: a benchmark with overall diversity comparable to the ALE, comprised of environments that fundamentally require generalization. We have created Procgen Benchmark to fulfill this need. This benchmark is ideal for evaluating generalization, as distinct training and test sets can be generated for each environment. This benchmark is also well-suited to evaluate sample efficiency, as all environments pose diverse and compelling challenges for RL agents. The environments' intrinsic diversity demands that agents learn robust policies; overfitting to narrow regions in state space will not suffice. Put differently, the ability to generalize becomes an integral component of success when agents are faced with ever-changing levels. All environments are open-source and can be found at https://github.com/openai/procgen.

2. Procgen Benchmark

Procgen Benchmark consists of 16 unique environments designed to measure both sample efficiency and generalization in reinforcement learning. These environments greatly benefit from the use of procedural content generation, the algorithmic creation of a near-infinite supply of highly randomized content. In these environments, employing procedural generation is far more effective than relying on fixed, human-designed content.

Procedural generation logic governs the level layout (Johnson et al., 2010), the selection of game assets, the location and spawn times of entities, and other game-specific details. To master any one of these environments, agents must learn a policy that is robust across all axes of variation. Learning such a policy is both more challenging and more relevant than overfitting to a handful of fixed levels.

[1]OpenAI, San Francisco, CA, USA. Correspondence to: Karl Cobbe <karl@openai.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

Figure 1. Screenshots from each game in Procgen Benchmark.

Screenshots from each environment are shown in Figure 1. We note that the state transition function is deterministic in all environments.[1]

2.1. Environment Desiderata

We designed all environments to satisfy the following criteria.

High Diversity: Procedural generation logic is given maximal freedom, subject to basic design constraints. The diversity in the resulting level distributions presents agents with meaningful generalization challenges.

Fast Evaluation: Environment difficulty is calibrated such that baseline agents make significant progress training over 200M timesteps. Moreover, the environments are optimized to perform thousands of steps per second on a single CPU core, including the time required to render observations. This enables a fast experimental pipeline.

Tunable Difficulty: All environments support two well-calibrated difficulty settings: easy and hard. This difficulty refers to the level distribution and not to individual levels; in both settings, the difficulty of individual levels has high variance. Unless otherwise specified, we report results using the hard difficulty setting. We make the easy difficulty setting available for those with limited access to compute power, as it reduces the resources required to train agents by roughly a factor of 8.

Level Solvability: The procedural generation in each environment strives to make all levels solvable, but this is not strictly guaranteed. For each environment, greater than 99% of levels are believed to be solvable.

Emphasis on Visual Recognition and Motor Control: In keeping with precedent, environments mimic the style of many Atari and Gym Retro (Pfau et al., 2018) games. Performing well primarily depends on identifying critical assets in the observation space and enacting appropriate low level motor responses.

Shared Action and Observation Space: To support a unified training pipeline, all environments use a discrete 15 dimensional action space and produce $64 \times 64 \times 3$ RGB observations. Some environments include no-op actions to accommodate the shared action space.

Tunable Dependence on Exploration: These environments were designed to be tractable for baseline RL agents without the need for custom exploratory rewards. However, many of these environments can be made into more challenging exploration tasks if desired. See Appendix B.1 for a discussion on evaluating exploration capability.

Tunable Dependence on Memory: These environments were designed to require minimal use of memory, in order to isolate the challenges in RL. However, several environments include variants that do test the use of memory, as we discuss in Appendix B.2.

By satisfying these requirements, we believe Procgen Benchmark will be a valuable tool in RL research. Descriptions of each specific environment can be found in Appendix A.

2.2. Experimental Protocols

By default, we train agents using Proximal Policy Optimization (Schulman et al., 2017) for 200M timesteps. While this timestep choice is arbitrary, it follows the precedent set by the ALE. It is also experimentally convenient: training for 200M timesteps with PPO on a single Procgen environment requires approximately 24 GPU-hrs and 60 CPU-hrs. We consider this a reasonable and practical computational cost. To further reduce training time at the cost of experimental complexity, environments can be set to

[1] Although the Chaser environment is deterministic, the enemy AI will make pseudorandom decisions conditioned on the level seed.

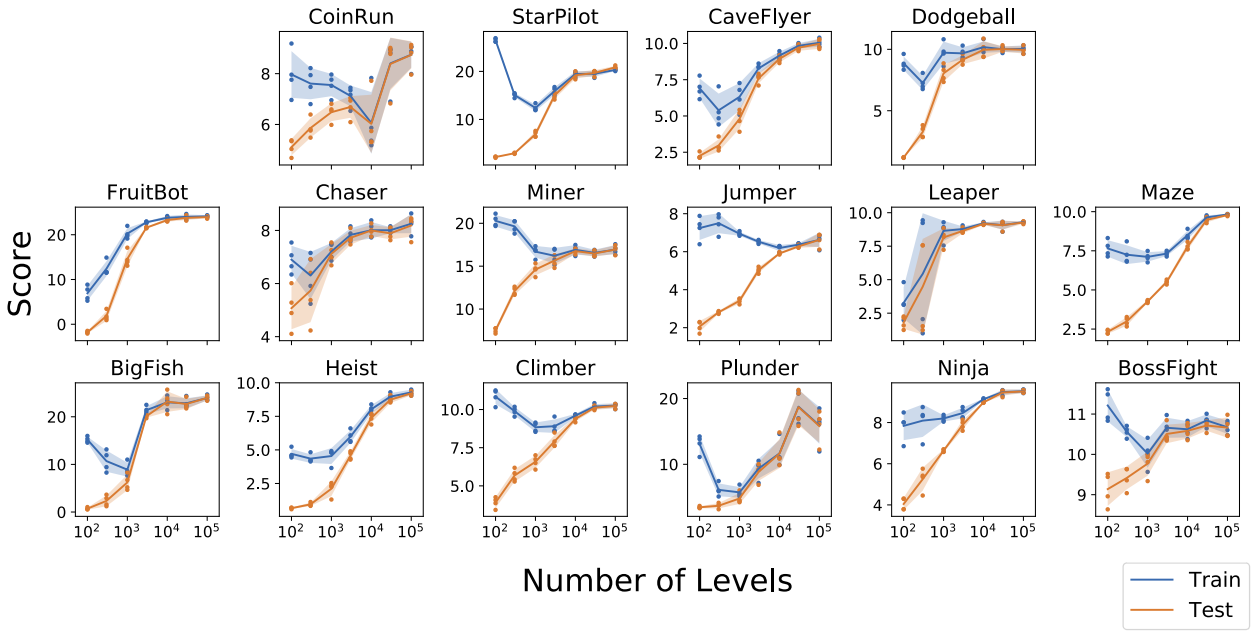

Figure 2. Generalization performance in each environment as a function of training set size. We report the mean raw episodic return, where each episode includes a single level. The mean and standard deviation is shown across 4 seeds.

the easy difficulty. We recommend training easy difficulty environments for 25M timesteps, which requires approximately 3 GPU-hrs with our implementation of PPO.

When evaluating sample efficiency, we train and test agents on the full distribution of levels in each environment. When evaluating generalization, we train on a finite set of levels and we test on the full distribution of levels. Unless otherwise specified, we use a training set of 500 levels to evaluate generalization in each environment. For easy difficulty environments, we recommend using training sets of 200 levels. We report results on easy difficulty environments in Appendix I.

When it is necessary to report a single score across Procgen Benchmark, we calculate the mean normalized return. For each environment, we define the normalized return to be $R_{norm} = (R - R_{min})/(R_{max} - R_{min})$, where R is the raw expected return and R_{min} and R_{max} are constants chosen to approximately bound R. Under this definition, the normalized return will almost always fall between 0 and 1. We use the mean normalized return as it provides a better signal than the median, and since there is no need to be robust to outliers. We designed all environments to have similar difficulties in order to prevent a small subset from dominating this signal. See Appendix C for a list of normalization constants and a discussion on their selection.

2.3. Hyperparameter Selection

In deep RL, hyperparameter tuning is often the difference between great and mediocre results. Unfortunately, this process can be costly in both time and computation. For those who are more comfortable with the existing ALE benchmark, minimal hyperparameter tuning should be required to train on Procgen environments. This is partially by design, as Procgen Benchmark heavily draws inspiration from the ALE and Gym Retro. To provide a point of comparison, we evaluate our Procgen-tuned implementation of PPO on the ALE, and we achieve competitive performance. Detailed results are shown in Appendix F.

As a convenience, we choose not to use any frame stacking in Procgen experiments, as we find this only minimally impacts performance. See Appendix H for further discussion. By default, we train agents with the convolutional architecture found in IMPALA (Espeholt et al., 2018), as we find this architecture strikes a reasonable balance between performance and compute requirements. We note that smaller architectures often struggle to train when faced with the high diversity of Procgen environments, a trend we explore further in Section 4.

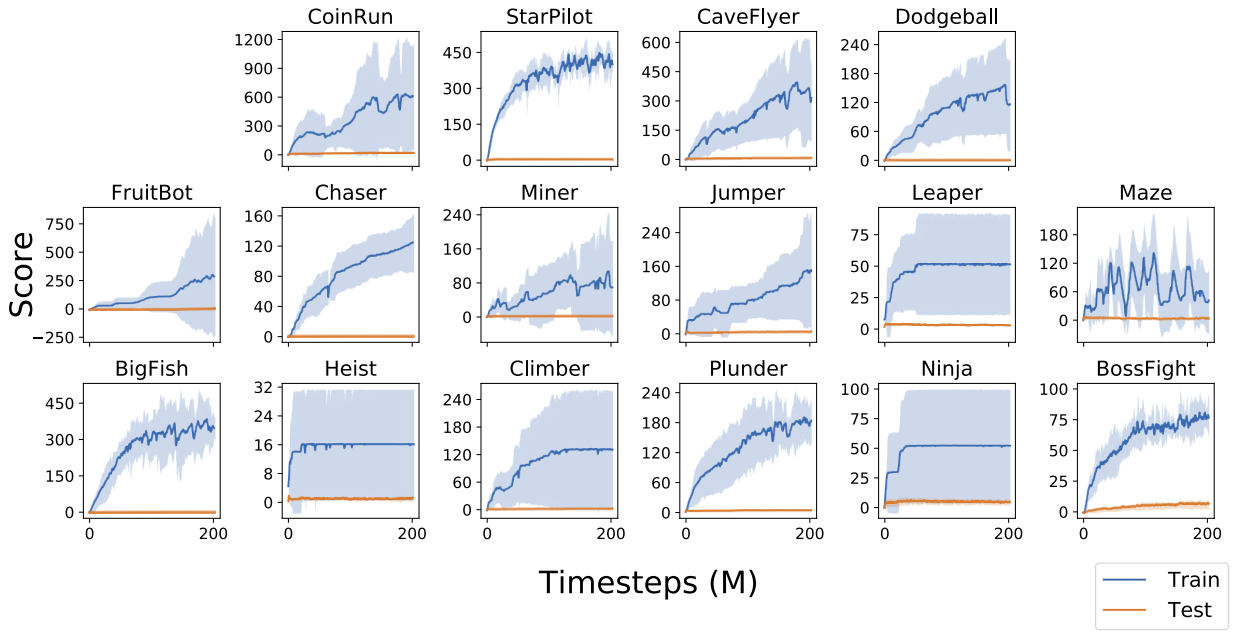

Figure 3. Train and test performance when training with a deterministic sequence of levels. We report the mean raw episodic return, where each episode may include many sequential levels. The mean and standard deviation is shown across 4 seeds.

3. Generalization Experiments

3.1. Level Requirements

We first evaluate the impact of training set size on generalization. For each environment, we construct several training sets ranging in size from 100 to 100,000 levels. We train agents for 200M timesteps on each training set using PPO, and we measure performance on held out levels. Results are shown in Figure 2. See Appendix D for a list of hyperparameters and Appendix E for test curves from each training set.

We find that agents strongly overfit to small training sets in almost all cases. To close the generalization gap, agents need access to as many as 10,000 levels. A peculiar trend emerges in many environments: past a certain threshold, training performance improves as the training set grows. This runs counter to trends found in supervised learning, where training performance commonly decreases with the size of the training set. We attribute this trend to the implicit curriculum provided by the distribution of levels. A larger training set can improve training performance if the agent learns to generalize even across levels in the training set. This effect was previously reported by (Cobbe et al., 2019), and we now corroborate those results with a larger number of environments.

3.2. An Ablation with Deterministic Levels

To fully emphasize the significance of procedural generation, we conduct a simple ablation study. Instead of resampling a new level at the start of every episode, we train agents on a fixed sequence of levels. In each episode, the agent begins on the first level. When the agent successfully completes a level, it progresses to the next level. If the agent fails at any point, the episode terminates. With this setup, the agent can reach arbitrarily many levels, though in practice it rarely progresses beyond the 20^{th} level in any environment. This approximately mimics the training setup of the ALE. To make training more tractable in this setting, we use the easy environment difficulty.

At test time, we simply remove the determinism in the level sequence, instead choosing level sequences at random. Results are shown in Figure 3. We find that agents become competent over the first several training levels in most environments, giving an illusion of meaningful progress. However, test performance demonstrates that the agents have in fact learned almost nothing about the underlying level distribution. We believe this vast gap between train and test performance is worth highlighting. It reveals a crucial hidden flaw in training on environments that follow a fixed sequence of levels. These results emphasize the importance of both training and evaluating with diverse environment distributions.

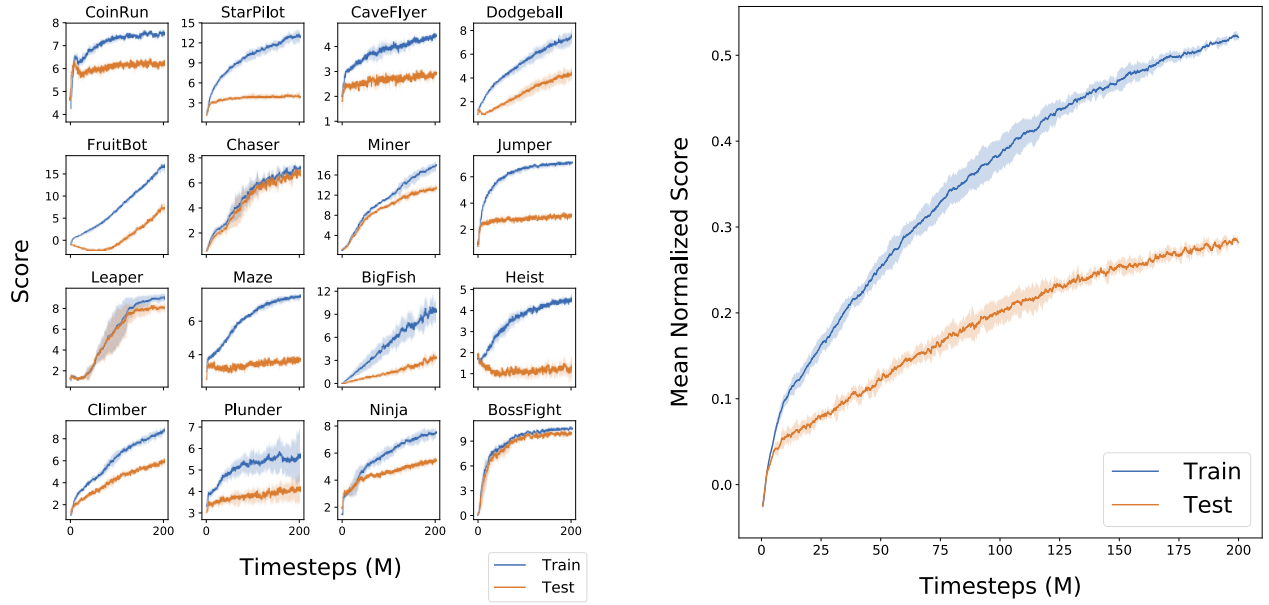

Figure 4. Generalization performance from 500 levels in each environment. The mean and standard deviation is shown across 3 seeds.

3.3. 500 Level Generalization

Due to the high computational cost, it is impractical to regularly run the experiments described in Section 3.1. To benchmark generalization, we recommend training on 500 levels from each environment and testing on held out levels, as in (Cobbe et al., 2019). We choose this training set size to be near the region where generalization begins to take effect, as seen in Figure 2. At test time, we measure agents' zero-shot performance averaged over unseen levels. When evaluating generalization, we do not explicitly restrict the duration of training, though in practice we still train for 200M timesteps.

Baseline results are shown in Figure 4. We see a high amount of overfitting in most environments. In some environments, the generalization gap is relatively small only because both training and test performance are poor, as discussed in Section 3.1. In any case, we expect to see significant improvement on test performance as we develop agents more capable of generalization.

4. Scaling Model Size

We now investigate how scaling model size impacts both sample efficiency and generalization in RL. We conduct these experiments to demonstrate the usefulness of Procgen Benchmark metrics, and because this is a compelling topic in its own right. We follow the experimental protocols described in Section 2.2, evaluating the performance of 4 different models on both sample efficiency and generalization.

The first 3 models use the convolutional architecture found in IMPALA (Espeholt et al., 2018) with the number of convolutional channels at each layer scaled by 1, 2 or 4. Note that scaling the number of channels by k results in scaling the total parameter count by approximately k^2. The final model uses the smaller and more basic convolutional architecture found in (Mnih et al., 2015), which we call Nature-CNN. We include this architecture as it is often used to train agents in the ALE.

We train the Nature-CNN model with the same learning rate as the smallest IMPALA model. When we scale the number of IMPALA channels by k, we also scale the learning rate by $\frac{1}{\sqrt{k}}$ to match the scaling of the weights, initialized with the method from (Glorot and Bengio, 2010). The learning rate is the only hyperparameter we vary between architectures. We performed sweeps over other hyperparameters, including the batch size and the number of epochs per rollout, and we found no other obvious gains.

Results are shown in Figure 5. We find that larger architectures significantly improve both sample efficiency and generalization. It is notable that the small Nature-CNN model almost completely fails to train. These results align with the results from (Cobbe et al., 2019), and we now establish that this trend holds across many diverse environments. Although larger models offer fairly consistent improvements, we note that some environments benefit from the larger models to a greater extent. See Appendix G for detailed training curves from each environment.

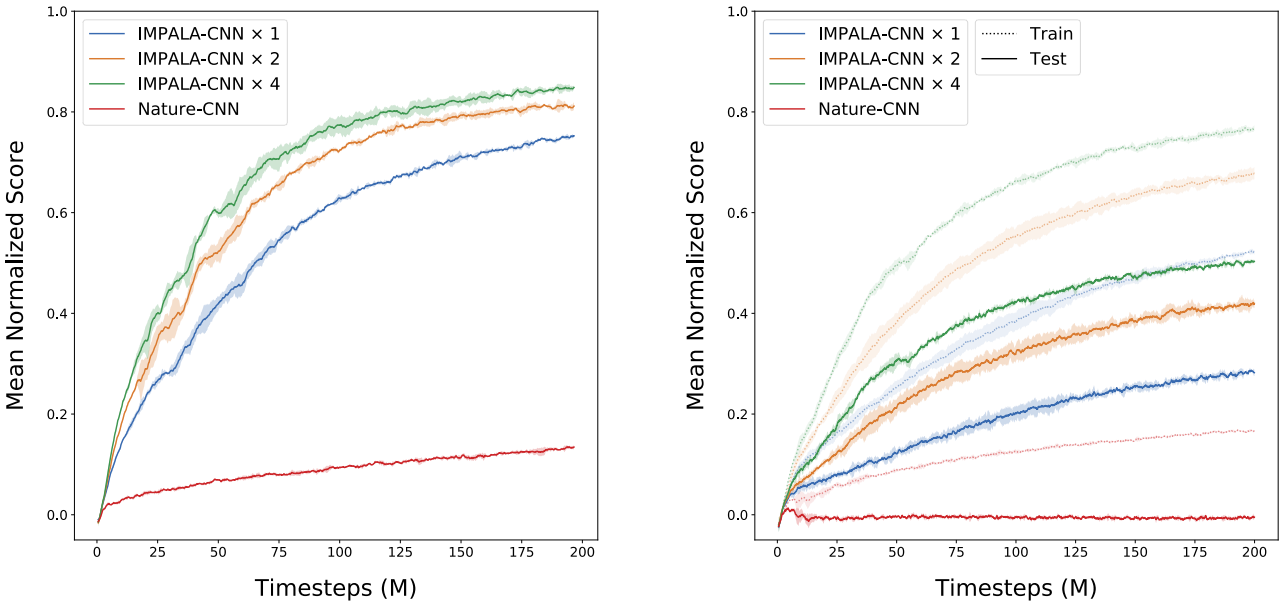

Figure 5. Performance of different model sizes, measuring both sample efficiency (left) and generalization (right). The mean and standard deviation is shown across 3 seeds.

5. Comparing Algorithms

We next compare our implementation of PPO to our implementation of Rainbow (Hessel et al., 2018) on Procgen Benchmark. We evaluate sample efficiency, training and testing on the full distribution of levels in each environment. As with PPO, we train Rainbow agents using the IMPALA convolutional architecture, collecting experience from 64 parallel environment copies into a single replay buffer. We first experimented with the default Rainbow hyperparameters (with an appropriate choice for distributional min/max values), but we found that agents struggled to learn any non-trivial behaviour. We hypothesize that the diversity of our environments can lead to high variance gradients that promote instability. We therefore reduced gradient variance by running the algorithm on 8 parallel workers, using shared model parameters and averaging gradients between workers. This greatly improved performance.

To improve wall-clock time for Rainbow, we also increased the batch size and decreased the update frequency each by a factor of 16, while increasing the learning rate by a factor of 4. While this change significantly reduced wall-clock training time, it did not adversely impact performance. We confirmed that the new learning rate was roughly optimal by sweeping over nearby learning rates. See Appendix D for a full list of Rainbow hyperparameters.

Results are shown in Figure 6. PPO performs much more consistently across the benchmark, though Rainbow offers a significant improvement in several environments. We're not presently able to diagnose the instability that leads to Rainbow's low performance in some environments, though we consider this an interesting avenue for further research.

6. Related Work

Many recent RL benchmarks grapple with generalization in different ways. The Sonic benchmark (Nichol et al., 2018) was designed to measure generalization in RL by separating levels of the *Sonic the Hedgehog*™ video game into training and test sets. However, RL agents struggled to generalize from the few available training levels, and progress was hard to measure. The CoinRun environment (Cobbe et al., 2019) addressed this concern by procedurally generating large training and test sets to better measure generalization. CoinRun serves as the inaugural environment in Procgen Benchmark.

The General Video Game AI (GVG-AI) framework (Perez-Liebana et al., 2018) has also encouraged the use of procedural generation in deep RL. Using 4 procedurally generated environments based on classic video games, (Justesen et al., 2018) measured generalization across different level distributions, finding that agents strongly overfit to their particular training set. Environments in Procgen Benchmark are designed in a similar spirit, with two of the environments (Miner and Leaper) drawing direct inspiration from this work.

The Obstacle Tower environment (Juliani et al., 2019) attempts to measure generalization in vision, control, and planning using a 3D, 3rd person, procedurally generated environment. Success requires agents to solve both low-

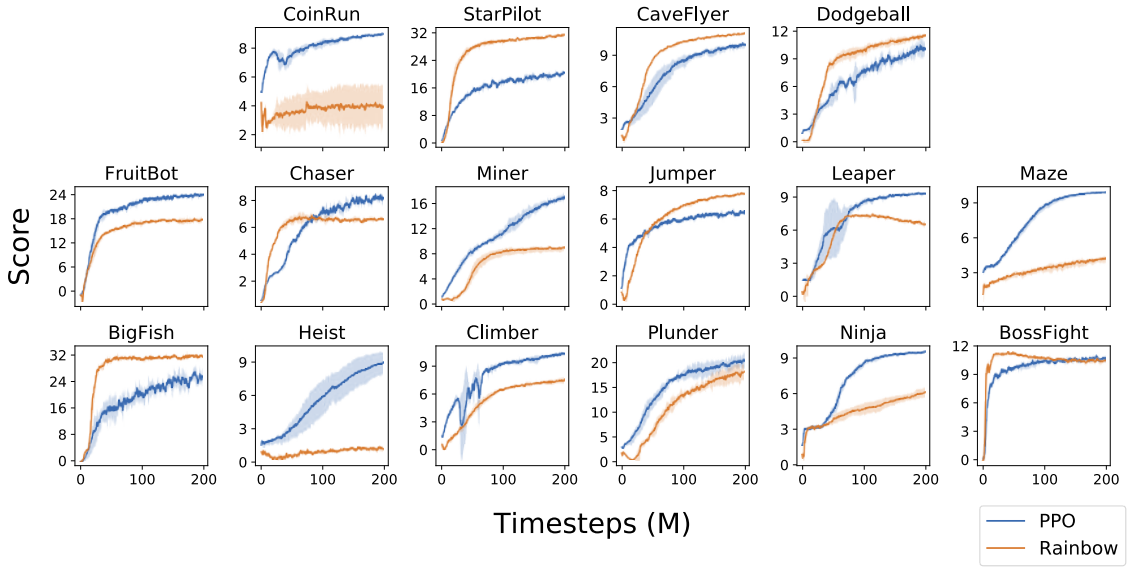

Figure 6. A comparison between Rainbow and PPO. In both cases, we train and test on the full distribution of levels from each environment. The mean and standard deviation is shown across 3 seeds.

level control and high-level planning problems. While studying generalization in a single complex environment offers certain advantages, we opted to design many heterogeneous environments for Procgen Benchmark.

bsuite (Osband et al., 2019) is a set of simple environments designed to serve as "an MNIST for reinforcement learning." Each environment targets a small number of core RL capabilities, including the core capability of generalization. bsuite includes a single environment that requires visual generalization in the form of an MNIST contextual bandit, whereas visual generalization is a primary source of difficulty across all Procgen environments.

Safety Gym (Achiam et al., 2019) provides a suite of benchmark environments designed for studying safe exploration and constrained RL. While generalization is not an explicit focus of this benchmark, all Safety Gym environments perform extensive randomization to prevent agents from overfitting to specific environment layouts. In doing so, these environments enforce a need for generalization.

The Animal-AI Environment (Beyret et al., 2019) uses tasks inspired by the animal cognition literature to evaluate agent intelligence. Since these tests are not encountered during training, high performance depends on generalizing well from the specific training configurations. The use of a single unified environment makes the prospect of generalization significantly more plausible.

Meta-World (Yu et al., 2019) proposes several meta-learning benchmarks, using up to 50 unique continuous control environments for training and testing. As with the Animal-AI Environment, the shared physics and mechanics between train and test environments gives rise to the plausible expectation of generalization, even when the details of the test task are novel.

7. Conclusion

Training agents capable of generalizing across environments remains one of the greatest challenges in reinforcement learning. We've designed Procgen Benchmark to help the community to contend with this challenge. The intrinsic diversity within level distributions makes this benchmark ideal for evaluating both generalization and sample efficiency in RL. We expect many insights gleaned from this benchmark to apply in more complex settings, and we look forward to leveraging these environments to design more capable and efficient algorithms.

References

J. Achiam, A. Ray, and D. Amodei. Safety gym. `https://openai.com/blog/safety-gym/`, 2019.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

B. Beyret, J. Hern'andez-Orallo, L. Cheke, M. Halina, M. Shanahan, and M. Crosby. The animal-ai environment: Training and testing animal-like artificial cognition. 2019.

K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1282–1289, 2019. URL http://proceedings.mlr.press/v97/cobbe19a.html.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018.

J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in DQN. *CoRR*, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 10. ACM, 2010.

A. Juliani, A. Khalifa, V.-P. Berges, J. Harper, H. Henry, A. Crespi, J. Togelius, and D. Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *arXiv preprint arXiv:1902.01378*, 2019.

N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *CoRR*, abs/1806.10729, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society, 7*, pages 48–50, 1956.

K. Lee, K. Lee, J. Shin, and H. Lee. A simple randomization technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019.

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in RL. *CoRR*, abs/1804.03720, 2018. URL http://arxiv.org/abs/1804.03720.

I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvári, S. Singh, B. Van Roy, R. Sutton, D. Silver, and H. van Hasselt. Behaviour suite for reinforcement learning. 2019.

C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *CoRR*, abs/1810.12282, 2018.

D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *arXiv preprint arXiv:1802.10363*, 2018.

V. Pfau, A. Nichol, C. Hesse, L. Schiavo, J. Schulman, and O. Klimov. Gym retro. https://openai.com/blog/gym-retro/, 2018.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta-reinforcement learning, 2019. URL https://github.com/rlworkgroup/metaworld.

A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *CoRR*, abs/1806.07937, 2018a.

A. Zhang, Y. Wu, and J. Pineau. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032*, 2018b.

C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. *CoRR*, abs/1804.06893, 2018c. URL http://arxiv.org/abs/1804.06893.