A Implementation Details

The implementation details often matter with reinforcement learning [12]. For this reason, full source code is provided in the supplementary material for all experiments. This section details some of the more important implementation decisions made, most of which match those found in [3]. We ran all experiments (PPO, DNA, PPG) using these same implementation choices to confirm that performance differences were not due differences in implementation.

Reward normalization We normalized rewards such that returns have unit variance, as is common with PPO.[17] Even though we used two separate models, and therefore had less reason to balance the magnitude of the value and policy loss, we still kept reward normalization so that distillation loss, and its interaction with the policy constraint, would be of a similar scale between environments.

Observation normalization We also adopted observation normalization. Each state s was normalized by $s' = \text{clip}((s - s_\mu)/s_\sigma, -3, 3)$ where s_μ was the element wide mean over states seen by the agent so far, and s_σ was the standard deviation. Normalization constants were shared between the policy and value networks.

Repeat action penalty We found that our policy would occasionally get stuck repeating a single action, causing the game to freeze until the time limit occurred. This could occur if the agent mistakenly thought it would get a slightly negative score for continuing and therefore acted to postpone that reward as long as possible.[18] Q-learning algorithms, such as DQN, which make use of ϵ-greedy, do not experience this problem so long as $\epsilon > 0$. To address this, we implemented a reward penalty of 0.25 (normalized) if the agent repeated the same action more than 100 times. We leave finding a better solution to this problem for future work.[19]

Integrating time and action information We added a watermark to the least recent frame in the 4-frame stack indicating the proportion of time which has occurred as a 'progress bar' as well as markers on each frame indicating which action the agent selected on the previous frame. Inclusion of time is necessary to avoid violation of the Markovian property in time-limited environments [26]. Action indicators were added to allow the agent to understand when it repeated the same action multiple times, which is not always possible to determine from the state itself (due to multiple actions causing identical outcomes).

Warmup / desyncing environments When initializing our environments, we ran each of the parallel 128 environments for $t \sim U(1, 1000)$ interactions with actions selected uniformly over the action space. This served two purposes: to provide initial normalization parameters and to desynchronize the environments. We found that if we did not do this, agents would terminate around the same time on some environments, causing parallel rollouts to become correlated. While we found this made very little difference to the agent's performance, it removed oscillating scoring artefacts found early in training in some environments.

B Hyperparameters and Environmental Settings

We selected initial hyperparameters from an initial coarse hyperparameter search on the Atari-3 validation set. In some cases, where only small differences in performances were observed, we prefered settings that had been used in previous papers or were likely to be more efficient. For example, our search found a mini-batch size of 256 optimal for value and distil updates. However, we selected 512 instead, as the difference was not large and found this a more computationally efficient mini-batch size when trained on a GPU. A full list of hyperparameters are given in Table 1. We also provide hyperparameters for our PPG experiments in Table 2. The environmental settings we used are given in Table 3.

[17] We also clipped normalized rewards to $[-5, 5]$ but found that this occurred exceedingly rarely, especially after the first 1 million frames.

[18] An example of this would be the agent failing to press the reset button after losing a life in Breakout.

[19] Adding some kind of exploration strategy, such as Random Network Distillation [7] would likely solve this problem more elegantly.

Setting	DNA	PPO	PPO_{orig}
Entropy bonus (c_{eb})	0.01	0.01	$\alpha \times 0.01$
Rollout horizon (N)	128	128	128
Parallel agents (A)	128	128	8
PPO epsilon ϵ	0.2	0.2	0.1
Discount gamma (γ)	0.999	0.999	0.99
Learning Rate	2.5×10^{-4}	2.5×10^{-4}	$\alpha \times 2.5 \times 10^{-4}$
Policy lambda (λ_π)	0.95	0.95	0.95
Value lambda (λ_V)	0.95	0.95	0.95
Policy epochs (E_π / E_{ppo})	2	2	3
Value epochs (E_V)	2	-	-
Distil epochs (E_D)	2	-	-
Distil beta (β)	1.0	-	-
Policy mini-batch size	2048	2048	256
Value mini-batch size	512	-	-
Distil mini-batch size	512	-	-
Repeated action penalty	0.25	0.25	0.25
Global gradient clipping	5.0	5.0	5.0

Table 1: Summary of hyperparameters found in coarse hyperparameter search. Epoch counts, and λ_* values were further fine-tuned as detailed in the main study. For PPO_{orig}, α was linearly annealed over training from $[1, 0]$.

Setting	PPG	PPG (tuned)
Policy epochs (E_π)	1	2
Value epochs (E_V)	1	1
Distil epochs (E_D)	0	0
Auxiliary epochs (E_{aux})	6	2
Auxiliary Period (N_π)	32	32

Table 2: Summary of hyperparameters used in the Phasic Policy Gradient experiments. All other hyperparameters were set according to the DNA settings in Table 1.

Setting	Easy	Hard
Terminal on Loss of Life	True	False
Action Space	Minimal	Full
Repeat Action Probability	0.0	0.25
Training frames	200M	
Color / grayscale	Grayscale	
Frame stacked	4	
Action repetitions	4	
Reward clipping	No	
Episode timeout	108K	
Resolution	84×84	
Noop Starts	1-30	

Table 3: Environmental Settings used in experiments. 'Hard' mode settings follow best practice by [22], 'easy' mode correspond to those used in the Rainbow DQN paper [16], with the exception that we do not apply the domain specific reward clipping modification. Training frames includes skipped frames, that is our agents performed 50M interactions with the environment.

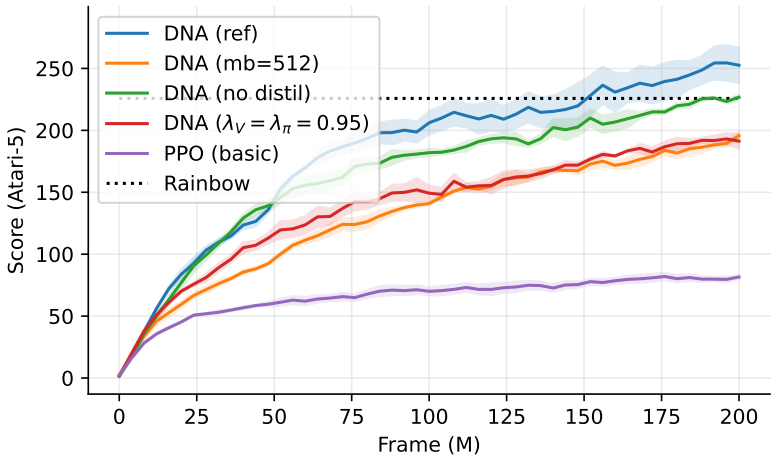

Figure 7: Training curves for the ablation studies. Shading indicates standard error over 3 seeds.

C Ablation Study

This appendix quantifies the performance contribution of several important components of DNA. We considered the following changes:

- **DNA (mb=512)** Our analysis of noise levels suggested a much larger mini-batch size for policy updates than for value updates. We measure the impact of this change by evaluating DNA with mini-batch sizes for all three training objectives set to 512.
- **DNA (no distil)** Validation scores indicated an improvement in performance using distillation over no distillation. We verify that this result is replicated on our test set.
- **DNA ($\lambda_V = \lambda_\pi = 0.95$)** We measured the impact of using non-homogeneous values for λ_V and λ_π by testing with these both set to 0.95.

We also include the reference run from the main study and a "PPO (basic)" run, which was a single network with the 'Nature-CNN' encoder, and a mini-batch size of 512, and can be thought of as DNA with all novel components turned off. Results are provided in Figure 7, along with the score, and performance regression in Table 4. We found non-homogeneous values for λ_V and λ_π, and a larger policy mini-batch size, to be the most significant changes, with distillation also providing some benefit.

Table 4: Atari-5 scores for each of the ablation runs.

Run	Atari-5 Score	Regression
DNA (ref)	252	0.0
DNA (no distil)	226	-10.2%
DNA (mb=512)	195	-22.5%
DNA ($\lambda_V = \lambda_\pi = 0.95$)	191	-24.3%
PPO (basic)	81	-67.7%

D Noise Scale for Distillation Learning

In this appendix, we present the noise scale results for distillation learning. We expected distillation to have a low noise level because the targets are drawn from the relatively noise-free value network estimations and not from the higher variance value targets. Our results in Figure 8 confirm this hypothesis. Of note is that the difference in noise scale between environments was much more significant for distillation loss than it was for the value or policy loss. These results indicate that distillation may benefit from smaller mini-batch sizes. However, the decreased efficiency of processing these smaller batches on GPU hardware may out-weigh any potential advantages.

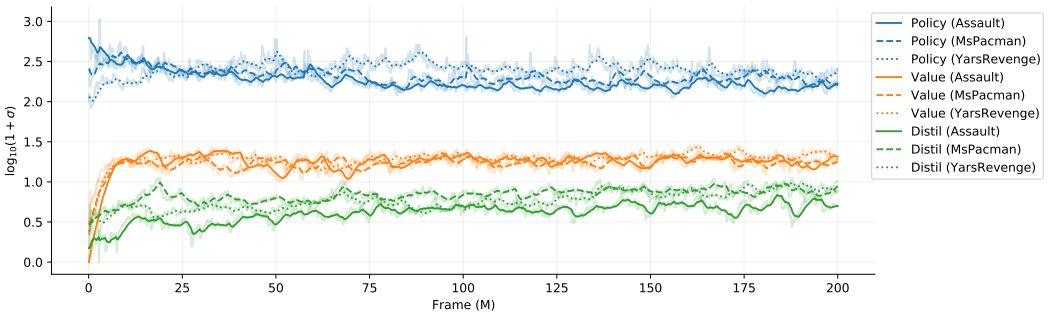

Figure 8: Noise level of the three tasks, policy learning, value learning, and distillation, over the three games in our validation set.

E Proof of Relationship between GAE and TD(λ)

We provide a proof that calculating the General Advantage Estimate [30] is equivalent to calculating TD(λ) returns, then subtracting the state value estimate. Concretely, for some $\lambda \in [0..1)$ and $\gamma \in [0..1]$ we have from [30]

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + \Big(\sum_{i=0}^{k-1} \gamma^i r_{t+i} \Big) + \gamma^k V(s_{t+k}) \tag{13}$$

$$= -V(s_t) + \text{NSTEP}^{(\gamma,k)}(s_t). \tag{14}$$

The GAE advantage estimate is defined as an exponentially weighted sum of these $A^{(k)}$'s as follows

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} := \sum_{i=0}^{\infty} (1-\lambda)\lambda^i \hat{A}_t^{(i+1)} \tag{15}$$

$$= (1-\lambda) \sum_{i=0}^{\infty} \lambda^i (-V(s_t) + \text{NSTEP}^{(i+1)}(s_t)) \tag{16}$$

$$= \left[(1-\lambda) \sum_{i=0}^{\infty} \lambda^i (-V(s_t)) \right] + \left[(1-\lambda) \sum_{i=0}^{\infty} \lambda^i \text{NSTEP}^{(i)}(s_t) \right] \tag{17}$$

$$= -V(s_t) + \text{TD}^{(\gamma,\lambda)}(s_t) \tag{18}$$

as required.

F Results under Rainbow DQN Style Environmental Settings.

In our main study, we compared DNA to PPO on the Atari-5 benchmark under the recommended settings given by [22]. Many prior results have been generated using the simpler, non-stochastic

version of the environments and with domain-specific knowledge, such as loss of life as a terminal state and a custom clipped reward modifier. We evaluated DNA and our implementation of PPO here in the simplified environments and found a modest improvement under these settings. We provide these results for better comparison against previous works.

In these experiments we did not use reward clipping. Reward clipping is a domain-specific reward modification that reduces all positive rewards to +1 and all negative rewards to -1. We were concerned that clipping rewards would bias the algorithm, as the agent is optimizing a reward structure that may not match the true rewards of the game. It could be that reward clipping may be necessary for Deep Q-learning approaches to reduce high variance returns.[20] However, we have not found this to be an advantage over reward normalization for PPO or DNA.

We found that DNA outperformed Rainbow DQN on all five environments, and obtained a better Atari-5 score after just 49M environmental frames. Proximal Policy Optimization, with a single policy update, also outperformed Rainbow DQN on this task.

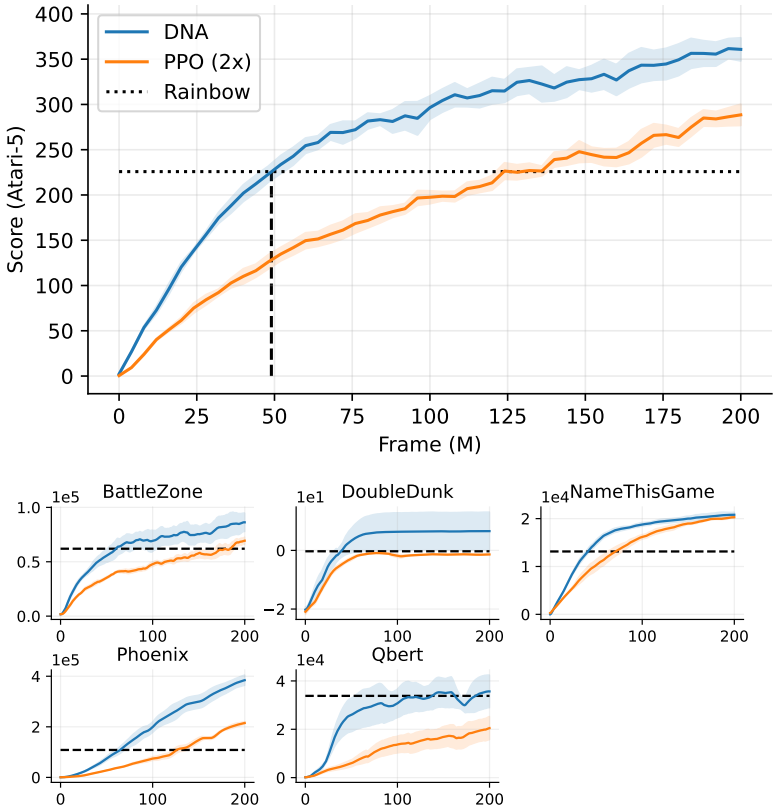

Figure 9: Results on the Atari-5 benchmark, with 'easy' environmental settings matched to those used by Rainbow DQN [16]. Shaded regions indicate standard error over three seeds.

G Supplementary Results

We investigated some supplementary questions. Specifically, we wanted to validate that the performance improvement of DNA compared to PPO and PPG did not result solely from the hyperparameter choices. We, therefore, evaluated PPO and PPG under a range of alternative hyperparameters on Atari-5 and note the results here.

We tested a variety of alternatives for PPO as described below. We found that none of the alternative settings resulted in improved performance (Figure 10 *left*).

[20] Or squashing the value function, see Appendix A of [4].

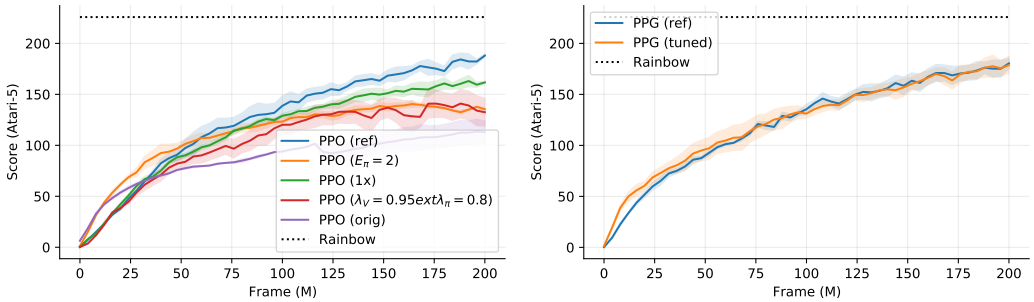

Figure 10: Training curves for the supplementary studies. Shading indicates standard error over 3 seeds. Reference runs are from the main study.

- **PPO** ($E_\pi = 2$) Our tuning process found a single epoch optimal for PPO, but two epochs optimal for DNA, would PPO have performed better if it was given two epochs instead of one? We found that while initial performance was stronger, this change ultimately regressed the performance.
- **PPO** ($\lambda_V = 0.95\, \lambda_\pi = 0.8$) Using separate return estimations for advantages and value targets does not require a dual network setup. Therefore we checked if the performance of PPO can be improved by using the non-homogeneous λ values used in our DNA experiment. We found these settings regressed performance.
- **PPO (1x)** In our experiments DNA and PPO used different network encoders (PPO used twice as many channels). It is possible that increasing the parameters made training more difficult for PPO? We tested PPO with the standard NatureCNN encoder, and found this change regressed performance.
- **PPO (orig)** Our settings for PPO deviated from those used by [31]. For completeness we also provide results using these settings. We found that while the these original settings performed well initially, they eventually unperformed the reference by a large margin. We also note that these settings took much longer to train (see Appendix H).

We also evaluated PPG with alternative settings (Figure 10 *right*).

- **PPG (tuned)** In our main experiment we evaluated PPG using $E_\pi = 1, E_V = 1, E_{\text{aux}} = 6$ taken from [11]. These differ significantly from those used for DNA. We therefore reevaluated PPG using $E_\pi = 2, E_V = 1, E_{\text{aux}} = 2$ which more closely match the settings used by DNA. We found this change had little impact on the performance of the algorithm.

H Training Time

DNA makes use of two independent networks and three training phases, which may have a negative effect on training time. We examine this here. All times are approximate and for comparative purposes only. Rainbow DQN times are on different hardware and using a different codebase.

Our implementation of DNA ran very quickly and is faster than PPO when PPO is configured as per [31]. This is due to our use of more parallel agents (128 vs 8) coupled with a larger mini-batch size. We give approximate training times in Table 5 which were taken from a 24-core machine with four 2080-TIs. We found we could train 8 DNA models in 8-hours on our 4-GPU machine, giving a rate of 4 GPU hours per game learned.

I Tuning for Proximal Policy Optimization

We repeated the same hyperparameter sweep on PPO as we did for DNA for a fair comparison. We found that, like DNA, PPO also benefited greatly from reduced epochs during training. We present

[21] There are faster ways of training DQN like algorithms, for example Ape-X [18].

Table 5: Approximate training times for the algorithms used in this paper.

Algorithm	GPU hours per game
PPO (our settings)	3
DNA	4
PPG	4.5
PPO ([31] settings)	7.5
Rainbow DQN	240[21]

the results here for 1,2,3 and 4 epochs, along with a search over the choice for λ used in the General Advantage Estimate. Results from the main study used the best performing model, which was found to be $\lambda = 0.95$, and $E_\pi = 1$. These hyperparameters differ from those used by [31], and achieve a significant improvement in performance with less computation (see Appendix F, H).

We found the performance of PPO to plateau after a point in training which decreases with the number of training epochs. This is consistent with [31], who trained for 40M frames and whose results show performance levelling off around 20M. However, when fewer epochs are used, performance continues to increase after these points. We also performed some quick experiments using partial epochs (0.5 and 0.75) but found these under-performed a single epoch and have not included the results here. We also found that the single network setup of PPO did not benefit as much as DNA from tuning the λ parameter and that the commonly used $\lambda = 0.95$ was optimal for our training set.

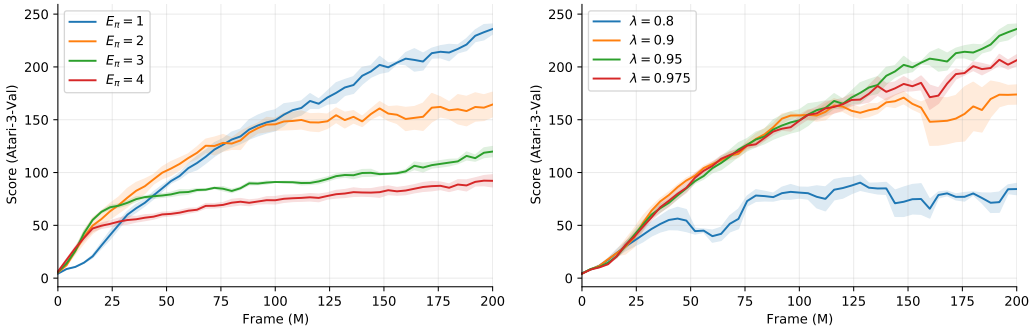

Figure 11: Training Curves for Proximal Policy Optimization over various epoch counts, and settings for λ_{GAE}. Shaded area indicates standard error over three seeds.

J Distillation Targets

We evaluated two distillation targets for our distillation phase: a *random projection* into $\mathbb{R}^{16}$, and the value networks *value estimates* as well as a third strategy, *feature matching*.[22] Random projection and value estimate outperformed the no distillation baseline, and feature matching underperformed the baseline. Results (Figure 12) are from a single seed on the Atari-3 validation dataset. In all cases, distillation was trained on two passes of trajectories sampled from the rollout and used a policy constraint.

K Additional Results on MuJoCo

We applied DNA to the robotics task MuJoCo [34]. We used hyperparameters based on the work of [31]. However, we used the slightly different 'v2' versions of the environments rather than the 'v1' versions used in their experiments. Scores are taken during training. PPO and DNA learned a per action standard deviation independent of the state.

[22] That is, the distillation step minimized the mean squared error between the features outputted by the policy network and the features output by the value network. As with our other targets, gradients were only propagated through the policy network.

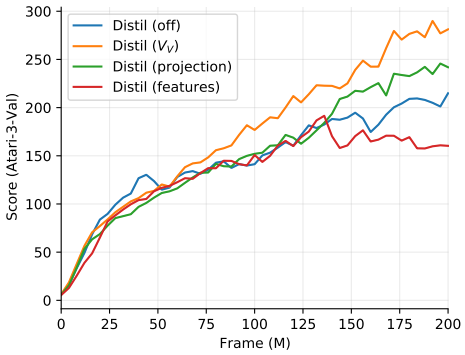

Figure 12: Performance of the four distillation strategies trailed. Only a single seed was used.

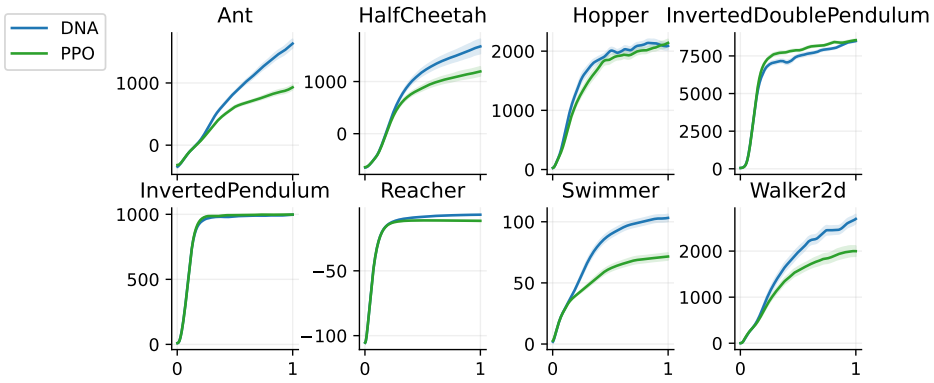

Figure 13: Results from the MuJoCo experiments for PPO and DNA over 30 seeds, with one standard error shown shaded.

DNA shows an improvement over PPO on five of the eight of the environments tested (*Ant*, *HalfCheetah*, *Reacher*, *Swimmer*, and *Walker2d*) (see Figure 13). In the remaining three environments, DNA and PPO produce similar results. We performed only basic hyperparameter tuning, using the *Walker2D* environment. Hyperparameters for these experiments are given in Table 6.

Unlike in our Atari experiments, we found setting λ_π to 0.8 to produce poor results, and so reverted this setting back to 0.95. We removed the KL divergence penalty for distillation and replaced it with the mean-squared error between the center of the Gaussian distribution outputs for the policy and value networks.

Table 6: Hyperparameters used for MuJoCo. Hyperparameters follow closely to that of [31]. Agents where trained for one-million interactions. † Learning rate was annealed linearly to 0.0 over training.

Setting	PPO	DNA
Entropy bonus (c_{eb})	0.01	
Rollout horizon (N)	2048	
Parallel agents (A)	1	
PPO epsilon ϵ	0.2	
Discount gamma (γ)	0.99	
Learning Rate	3.0×10^{-4}†	
Policy lambda (λ_π)	0.95	
Value lambda (λ_V)	0.95	
Policy epochs (E_π / E_{ppo})	10	10
Value epochs (E_V)	-	10
Distil epochs (E_D)	-	10
Distil beta (β)	-	1.0
Policy mini-batch size	64	64
Value mini-batch size	-	64
Distil/Aux mini-batch size	-	64
Global gradient clipping	5.0	5.0

L Additional Results on Procgen

We ran additional experiments on the Procgen benchmark [10]. Each environment within this benchmark requires learning across a set of 200 procedurally generated environments. Because of this, algorithms benefit from large replay buffers able to capture the diverse conditions under which the agent must act. Unlike PPG, DNA does not make use of replay buffer and so is not likely to perform well at this task.

Despite this, we found DNA to be competitive with PPG on many of the environments tested (*coinrun*, *fruitbot*, *leaper*, *maze*, *miner*, *ninja*, *dodgeball* and *heist*), as shown in Figure 15. When scores across all environments are normalized, DNA produces an average normalized score of 0.65 compared to 0.49 for PPO and 0.75 for PPG (see Figure 14).

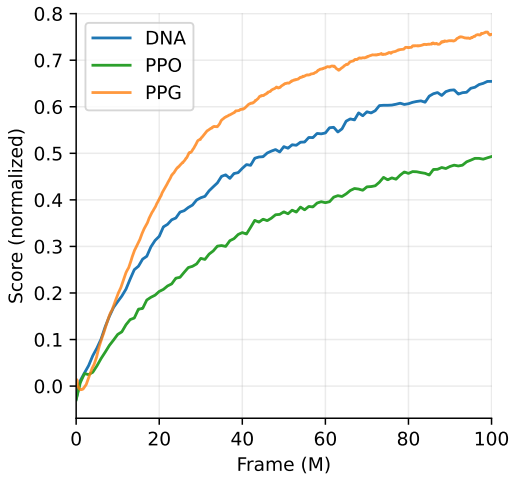

Figure 14: Results on the Procgen benchmark (hard distribution settings). Scores are average normalized score over all 16 environments. PPG results are taken from [11].

We kept hyperparameters close to that of [11] and used the *bigfish* environment to select one policy, one value, and two distil epochs as optimal for DNA. We note that these settings require the observations to be forwarded through a network only six times, compared to fourteen times in PPG. This makes DNA similar in computational requirements to PPO.[23] PPG's unusually high score early on in heist appears to be an error, but is presented verbatim from the source.[24]

We generated results for DNA and PPO and used the results supplied by [11] for PPG.[25] We also noticed a modest decline in the performance of our implementation of PPO compared to that of [11] (0.49 vs 0.58). This could be due to our use of observation normalization. The difference in performance is especially apparent in the environment *plunder* (for both DNA and PPO). This might indicate that performance could be improved further by adopting the fixed scaling preprocessing procedure used in PPG. We give hyperparameters for our experiments in Table 7.

[23] PPG's auxiliary phase requires forwarding through both the policy and value networks, whereas DNA's distillation update only requires a forward through the policy network.

[24] We suspect this is due to a bug in the environment where the first episode is identical (and trivial to solve) regardless of the seed.

[25] As recorded in the corresponding Github repository `https://github.com/openai/procgen`. We note some artefacts in the scores they supply, most notable in *heist* and *maze*. We believe this may be due to how their environments were initialized.

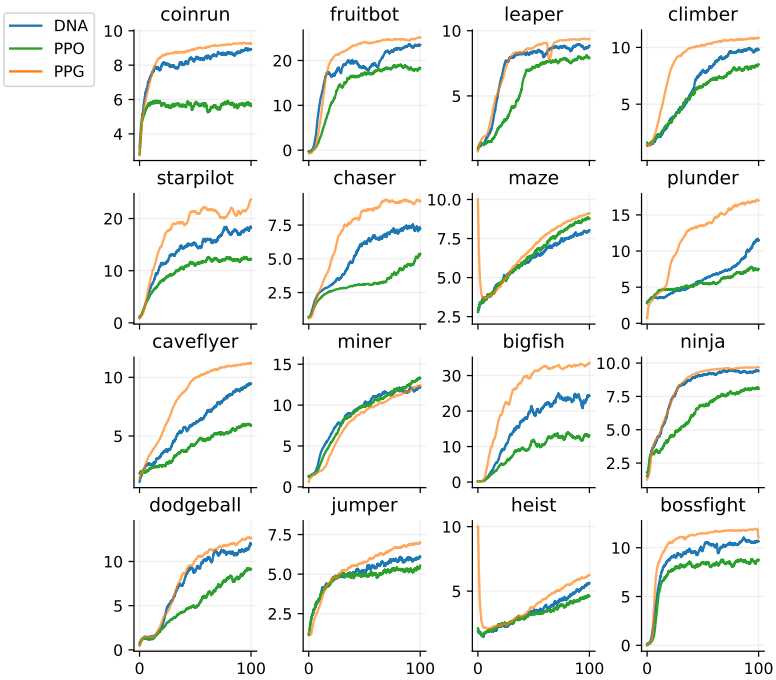

Figure 15: Results on each game in the Procgen benchmark (hard distribution settings). Scores smoothed for clarity. PPG results are taken from [11] and are an average over three seeds. PPO and DNA are our results over one seed.

Table 7: Hyperparameters used for ProcGen. PPG hyperparameters are taken from [11].

Setting	PPO	DNA	PPG
Entropy bonus (c_{eb})		0.01	
Rollout horizon (N)		256	
Parallel agents (A)		256	
PPO epsilon ϵ		0.2	
Discount gamma (γ)		0.999	
Learning Rate		5.0×10^{-4}	
Policy lambda (λ_π)		0.95	
Value lambda (λ_V)		0.95	
Repeated action penalty		0.0	
Policy epochs (E_π / E_{ppo})	3	2	1
Value epochs (E_V)	-	1	1
Distil/Aux epochs (E_D)	-	2	6
Distil/Aux beta (β)	-	1.0	1.0
Policy mini-batch size	8192	8192	8192
Value mini-batch size	-	2048	8192
Distil/Aux mini-batch size	-	512	4096
Global gradient clipping	5.0	5.0	off

M Additional Results on ALE

Our main study used Atari-5 to allow enough time for seeded runs and because it provides an established training/test split. Because Atari-5 is a new benchmark, we thought it important to validate our algorithm's performance on the full 57-game suite. We, therefore, provide results for both PPO (2x) and DNA on all 57 games in the ALE using both the 'easy' settings similar to [16] and the more difficult settings used in our main study.

We measured the median score as the median human-normalized score over the past 100-episodes and report the final median scores as the average for this measure over the last 10M frames (5% of training frames). Individual game scores are also reported as the average over the final 10M frames.

We found, under both the hard and the easy settings, DNA outperformed PPO by a wide margin (see Table 8). DNA also outperformed Rainbow DQN on the easy settings after just 85.5M training frames (Figure 16). Training plots are provided in Figures 17, 18. Results for each game are given in Tables 9, 10. Most surprising is that when trained with only a single policy epoch, a larger batch size, and more parallel agents, PPO becomes comparable to Rainbow DQN on the easy settings despite being a much simpler algorithm and being 80-times faster to train.

Table 8: Summary of results on the Atari-57 benchmark.

Algorithm	Median (easy)	Median (hard)
Rainbow	223	-
PPO	224	155
DNA (ours)	**311**	**207**

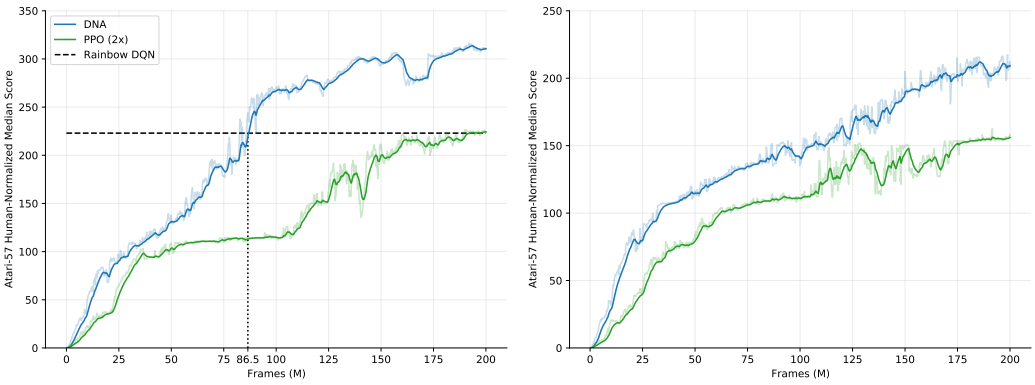

Figure 16: Median Score over all 57 games for DNA and PPO with 2x parameters. Left: performance under easy settings. DNA matches Rainbow DQN performance after just 86.5M frames. Right: performance under hard settings.

N Pseudocode for Generating Noise Scale Estimates.

In their paper, McCandlish et al. [23] define the noise scale as,

$$\mathcal{B}_{\text{noise}} := \frac{\text{tr}(H\Sigma)}{G^T H G} \tag{19}$$

where at parameter values θ, G is the true gradient, H is the true Hessian, and Σ is the per-example covariance matrix. For large models, the calculation of the Hessian is not practical. We, therefore, use their simplified measure

$$\mathcal{B}_{\text{simple}} := \frac{\text{tr}(\Sigma)}{|\vec{G}|^2}. \tag{20}$$

Even though this formulation makes the unrealistic assumption that the Hessian is a multiple of the identity matrix, empirical studies by McCandlish et al. have shown it to provide a surprisingly good approximation for $\mathcal{B}_{\text{noise}}$.

Our method for generating estimates of $\mathcal{B}_{\text{simple}}$ follows closely that of [23] and is formalized in Algorithm 2. While the estimates for $\text{tr}(\Sigma)$, and $|\vec{G}|^2$ are both unbiased their ratio may not be. We mitigate this by averaging over multiple samples when calculating $\vec{G}_{B_{\text{small}}}$ and applying smoothing to our estimate for $|\mathcal{G}|^2$.

Algorithm 2 Estimate Noise Scale

function ESTIMATENOISESCALE(
D, a batch of data.
L, a loss function.
θ, the model parameters.
N_{samples}, the number of small mini-batches to use.
$B_{\text{big}}, B_{\text{small}}$, the big and small mini-batch sizes.
$|\mathcal{G}|^2_{\text{old}}$, the previous smoothed $|\mathcal{G}|^2$ value.
α, smoothing factor to use for $|\mathcal{G}|^2$)
  Define SAMPLE(x,y), to draw y samples from x without replacement.
  $D_{\text{big}} \leftarrow \text{SAMPLE}(D, B_{\text{big}})$
  $|\vec{G}_{B_{\text{big}}}|^2 \leftarrow |\nabla_\theta L_{D_{\text{big}}}(\theta)|^2$
  $|\vec{G}_{B_{\text{small}}}|^2 \leftarrow 0$
  for $i = 1..N_{\text{samples}}$ **do**
    $D_{\text{small}} \leftarrow \text{SAMPLE}(D, B_{\text{small}})$
    $|\vec{G}_{B_{\text{small}}}|^2 \leftarrow |\vec{G}_{B_{\text{small}}}|^2 + (1/N_{\text{samples}})|\nabla_\theta L_{D_{\text{small}}}(\theta)|^2$
  $|\mathcal{G}|^2_{\text{new}} \leftarrow (B_{\text{big}}|\vec{G}_{B_{\text{big}}}|^2 - B_{\text{small}}|\vec{G}_{B_{\text{small}}}|^2)/(B_{\text{big}} - B_{\text{small}})$
  $\mathcal{S} \leftarrow (|\vec{G}_{B_{\text{small}}}|^2 - |\vec{G}_{B_{\text{big}}}|^2)/(1/B_{\text{small}} - 1/B_{\text{big}})$
  $|\mathcal{G}|^2 \leftarrow \alpha|\mathcal{G}|^2_{\text{old}} + (1-\alpha)|\mathcal{G}|^2_{\text{new}}$ ▷ Perform smoothing over $|\mathcal{G}|^2$, to reduce variance
  $\mathcal{B}_{\text{simple}} \leftarrow \mathcal{S}/|\mathcal{G}|^2$
  return $\mathcal{B}_{\text{simple}}$ ▷ the (squared) noise scale estimate.
  return $|\mathcal{G}|^2$ ▷ pass to next call of EstimateNoiseScale

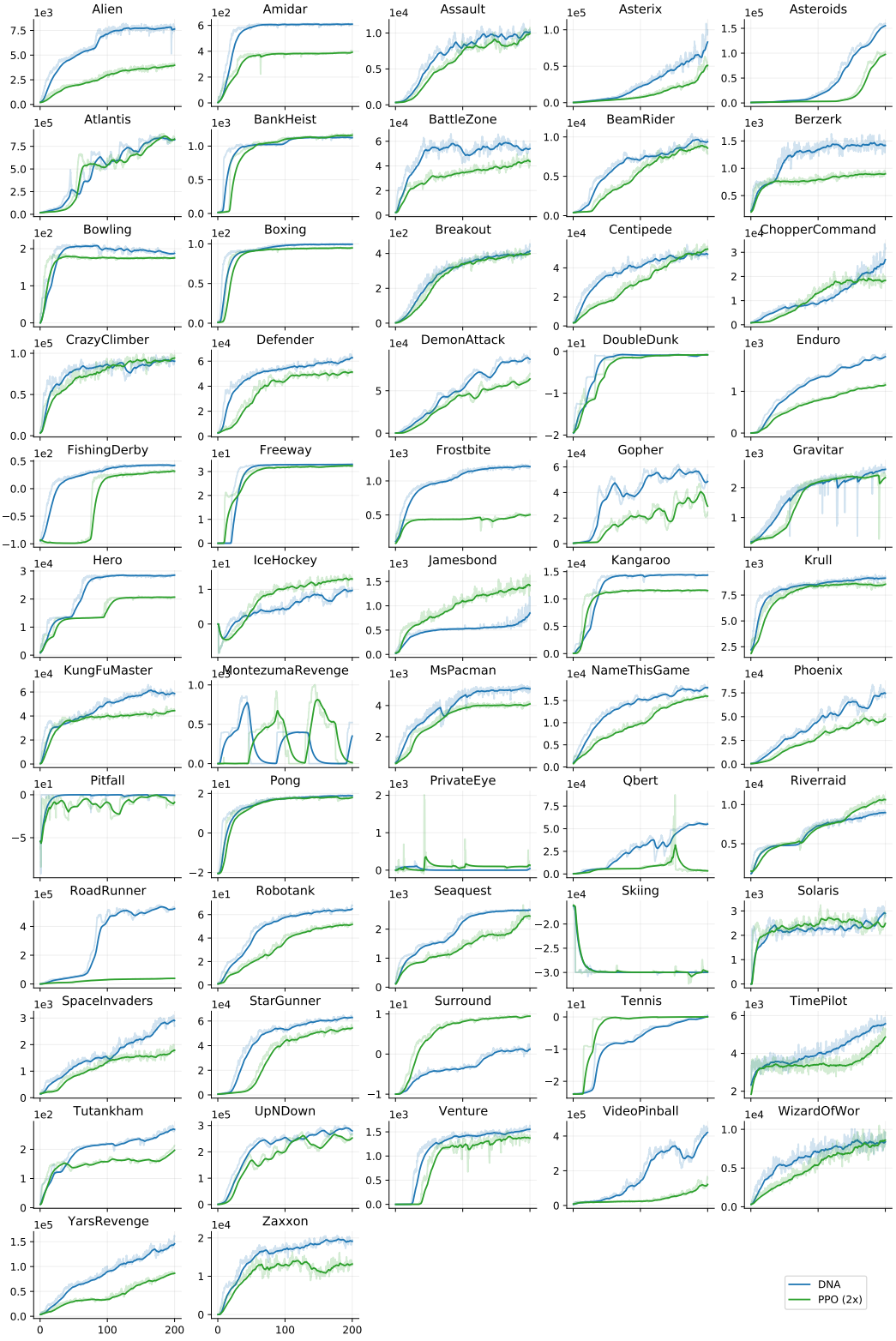

Figure 17: Training plots for DNA on all 57 games in the Atari-57 benchmark under 'hard' settings. Results are from a single seed, with smoothed results in bold, and non-smoothed results shown faded.

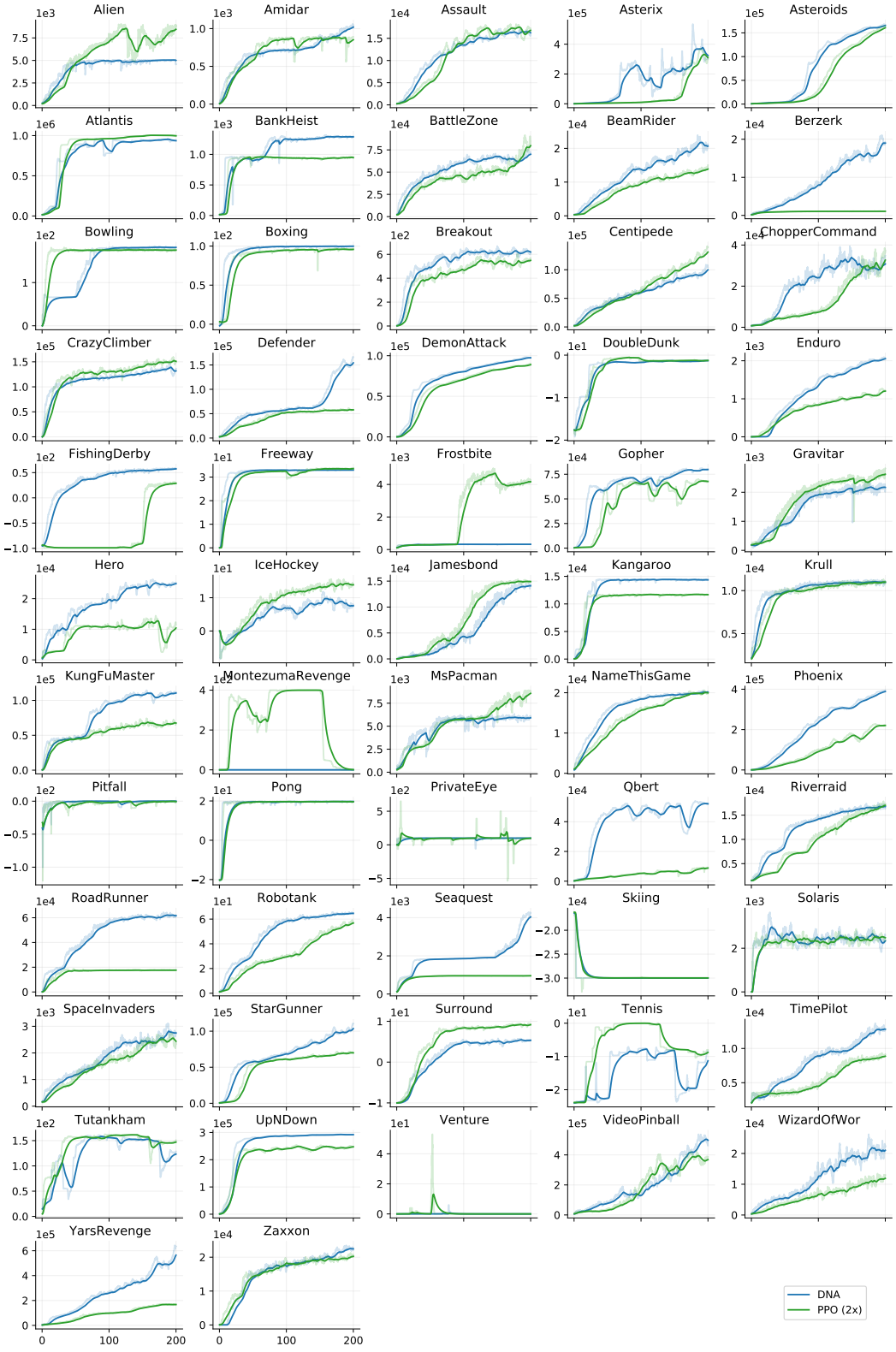

Figure 18: Training plots for DNA on all 57 games in the Atari-57 benchmark under 'easy' settings. Results are from a single seed, with smoothed results in bold, and non-smoothed results shown faded.

Table 9: Final scores for all 57 games in ALE under 'hard' settings. Reported as mean score over the final 10M frames of training.

Game	Random	Human	PPO (2x)	DNA
Alien	228	7,128	3,976	**7,617**
Amidar	5.8	**1,720**	391	610
Assault	222	742	10,098	**10,282**
Asterix	210	8,503	52,958	**85,070**
Asteroids	719	47,389	99,458	**157,926**
Atlantis	12,850	29,028	**816,033**	813,564
BankHeist	14.2	753	**1,159**	1,125
BattleZone	2,360	37,188	44,227	**54,462**
BeamRider	364	**16,926**	8,587	9,369
Berzerk	124	**2,630**	897	1,429
Bowling	23.1	161	175	**187**
Boxing	0.1	12.1	94.7	**99.4**
Breakout	1.7	30.5	399	**416**
Centipede	2,091	12,017	**53,676**	49,444
ChopperCommand	811	7,388	18,160	**26,998**
CrazyClimber	10,780	35,829	**94,695**	89,864
Defender	2,874	18,689	51,002	**62,935**
DemonAttack	152	1,971	64,180	**88,673**
DoubleDunk	-18.6	-16.4	-0.8	**-0.8**
Enduro	0.0	860	1,150	**1,819**
FishingDerby	-91.7	-38.7	32.0	**41.9**
Freeway	0.0	29.6	32.3	**33.0**
Frostbite	65.2	**4,335**	497	1,211
Gopher	258	2,412	28,785	**46,348**
Gravitar	173	**3,351**	2,209	2,627
Hero	1,027	**30,826**	20,673	28,526
IceHockey	-11.2	0.9	**12.9**	9.4
Jamesbond	29.0	303	**1,444**	861
Kangaroo	52.0	3,035	11,556	**14,367**
Krull	1,598	2,666	8,539	**9,161**
KungFuMaster	258	22,736	44,648	**58,895**
MontezumaRevenge	0.0	**4,753**	0.0	385
MsPacman	307	**6,952**	4,102	5,067
NameThisGame	2,292	8,049	16,050	**18,155**
Phoenix	761	7,243	47,804	**75,709**
Pitfall	-229.4	**6,464**	-10.4	-0.7
Pong	-20.7	14.6	17.9	**18.9**
PrivateEye	24.9	**69,571**	129	36.5
Qbert	164	13,455	3,273	**54,706**
Riverraid	1,338	**17,118**	10,642	9,005
RoadRunner	11.5	7,845	38,970	**520,458**
Robotank	2.2	11.9	51.5	**65.0**
Seaquest	68.4	**42,055**	2,494	2,655
Skiing	-17098.1	**-4336.9**	-29553.4	-29974.5
Solaris	1,236	**12,327**	2,379	2,976
SpaceInvaders	148	1,669	1,808	**2,940**
StarGunner	664	10,250	54,488	**62,760**
Surround	-10.0	6.5	**9.5**	0.9
Tennis	-23.8	-8.3	0.0	**0.2**
TimePilot	3,568	5,229	4,907	**5,554**
Tutankham	11.4	168	199	**272**
UpNDown	533	11,693	250,253	**280,014**
Venture	0.0	1,188	1,375	**1,562**
VideoPinball	0.0	17,668	120,142	**432,752**
WizardOfWor	564	4,756	**8,834**	8,480
YarsRevenge	3,093	54,577	87,267	**147,864**
Zaxxon	32.5	9,173	13,244	**19,125**

Table 10: Final scores for all 57 games in ALE under 'easy' settings. Reported as mean score over the final 10M frames of training.

Game	Random	Human	Rainbow DQN	PPO (2x)	DNA
Alien	228	7,128	**9,492**	8,525	5,021
Amidar	5.8	1,720	**5,131**	844	1,025
Assault	222	742	14,198	**16,688**	16,293
Asterix	210	8,503	**428,200**	321,207	323,965
Asteroids	719	47,389	2,713	161,787	**165,973**
Atlantis	12,850	29,028	826,660	**997,292**	932,559
BankHeist	14.2	753	**1,358**	951	1,286
BattleZone	2,360	37,188	62,010	**82,834**	71,003
BeamRider	364	16,926	16,850	13,932	**20,393**
Berzerk	124	2,630	2,546	1,083	**19,789**
Bowling	23.1	161	30.0	175	**181**
Boxing	0.1	12.1	99.6	95.6	**99.9**
Breakout	1.7	30.5	418	553	**626**
Centipede	2,091	12,017	8,167	**131,062**	100,194
ChopperCommand	811	7,388	16,654	**31,912**	31,181
CrazyClimber	10,780	35,829	**168,788**	151,937	131,623
Defender	2,874	18,689	55,105	58,201	**152,768**
DemonAttack	152	1,971	**111,185**	88,958	97,909
DoubleDunk	-18.6	-16.4	**-0.3**	-1.3	-1.3
Enduro	0.0	860	**2,126**	1,230	2,059
FishingDerby	-91.7	-38.7	31.3	29.0	**57.4**
Freeway	0.0	29.6	**34.0**	33.5	33.0
Frostbite	65.2	4,335	**9,590**	4,190	320
Gopher	258	2,412	70,355	67,850	**80,104**
Gravitar	173	**3,351**	1,419	2,632	2,190
Hero	1,027	30,826	**55,887**	11,125	24,904
IceHockey	-11.2	0.9	1.1	**13.8**	7.2
Jamesbond	29.0	303	**19,480**	14,947	14,102
Kangaroo	52.0	3,035	**14,638**	11,687	14,373
Krull	1,598	2,666	8,742	**11,007**	10,956
KungFuMaster	258	22,736	52,181	67,657	**110,962**
MontezumaRevenge	0.0	**4,753**	384	0.0	0.0
MsPacman	307	6,952	5,380	**8,712**	5,894
NameThisGame	2,292	8,049	13,136	20,053	**20,226**
Phoenix	761	7,243	108,529	220,560	**391,085**
Pitfall	-229.4	**6,464**	0.0	-0.5	0.0
Pong	-20.7	14.6	**20.9**	19.6	19.7
PrivateEye	24.9	**69,571**	4,234	99.9	100
Qbert	164	13,455	33,817	8,836	**52,398**
Riverraid	1,338	17,118	**22,500**	17,156	16,789
RoadRunner	11.5	7,845	**62,041**	17,596	61,713
Robotank	2.2	11.9	61.4	56.7	**64.8**
Seaquest	68.4	**42,055**	15,899	957	4,146
Skiing	-17098.1	**-4336.9**	-12957.8	-29974.4	-29974.0
Solaris	1,236	**12,327**	3,560	2,513	2,225
SpaceInvaders	148	1,669	**18,789**	2,497	2,731
StarGunner	664	10,250	**127,029**	70,247	104,125
Surround	-10.0	6.5	**9.7**	9.1	5.3
Tennis	-23.8	-8.3	**0.0**	-8.8	-10.9
TimePilot	3,568	5,229	**12,926**	8,918	12,774
Tutankham	11.4	168	**241**	147	127
UpNDown	533	11,693	103,600	247,994	**291,934**
Venture	0.0	**1,188**	5.5	0.0	0.0
VideoPinball	0.0	17,668	**533,936**	359,099	505,392
WizardOfWor	564	4,756	17,862	11,996	**20,851**
YarsRevenge	3,093	54,577	102,557	166,670	**564,513**
Zaxxon	32.5	9,173	22,210	20,330	**22,588**