PPG Reloaded: An Empirical Study on What Matters in Phasic Policy Gradient

Kaixin Wang [1] Daquan Zhou [2] Jiashi Feng [2] Shie Mannor [1 3]

Abstract

In model-free reinforcement learning, recent methods based on a phasic policy gradient (PPG) framework have shown impressive improvements in sample efficiency and zero-shot generalization on the challenging Procgen benchmark. In PPG, two design choices are believed to be the key contributing factors to its superior performance over PPO: the high level of value sample reuse and the low frequency of feature distillation. However, through an extensive empirical study, we unveil that *policy regularization* and *data diversity* are what actually matters. In particular, we can achieve the same level of performance with low value sample reuse and frequent feature distillation, as long as the policy regularization strength and data diversity are preserved. In addition, we can maintain the high performance of PPG while reducing the computational cost to a similar level as PPO. Our comprehensive study covers all 16 Procgen games in both sample efficiency and generalization setups. We hope it can advance the understanding of PPG and provide insights for future works.

1. Introduction

In recent years, model-free deep reinforcement learning (RL) has achieved great success in multiple domains ranging from video games (Badia et al., 2020; Cobbe et al., 2021) to robotics control (Haarnoja et al., 2018; OpenAI et al., 2019). One family of high-performing model-free algorithms is the actor-critic methods, such as PPO (Schulman et al., 2017), A3C (Mnih et al., 2016) and IMPALA (Espeholt et al., 2018). These methods learn a policy (*actor*) and a value function (*critic*). In deep RL with image observation (*e.g.*, video games), the policy and the value function often share a torso network to encode the observations into latent features. While this benefits sharing features between policy and value, jointly training the policy and value function might incur optimization interference and impose artificial restrictions on sample reuse (Cobbe et al., 2021).

Recently, Cobbe et al. (2021) proposes a phasic policy gradient (PPG) framework, which circumvents the disadvantages of using a shared encoder while preserving its benefits. The PPG framework is the core behind recent improvements (Cobbe et al., 2021; Raileanu & Fergus, 2021; Moon et al., 2022) in sample efficiency and zero-shot generalization on the large-scale Procgen benchmark (Cobbe et al., 2020). Therefore, it is worthwhile to take a closer look at what makes PPG so effective.

The stark difference between PPG and PPO is that PPG decouples the joint training of policy and value function into two separate phases (a policy phase and a distillation phase, see Section 2.2 for details). With this decoupling, one can make different algorithmic choices for policy and value learning (Table 1). In existing works (Cobbe et al., 2021; Moon et al., 2022), two design choices are believed to be the key factors to the superior performance of PPG:

- *High level of value sample reuse,*
- *Low frequency of feature distillation.*

However, these conclusions are drawn from ablation experiments that do not properly disentangle different factors. For example, in (Cobbe et al., 2021), increasing the frequency of feature distillation also shrinks the size of the off-policy buffer, consequently reducing the diversity of the buffered data.

To get a clear understanding of what matters in PPG, we conduct a large-scale empirical study on Procgen. Specifically, we focus on the three aspects in Table 1 and run ablation experiments with proper control of other (possibly confounding) hyperparameters. We aim to elucidate the core contributing factors in PPG and provide insights for future works built upon it. Our study consists of comprehensive experiments covering all 16 Procgen games in both sample efficiency and generalization setups.

Below is a summary of our **main findings**:

- The low frequency of feature distillation is actually

[1]Faculty of Electrical And Computer Engineering, Technion, Haifa, Israel [2]ByteDance, Singapore [3]NVIDIA Research, Haifa, Israel. Correspondence to: Kaixin Wang <kaixin96.wang@gmail.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

Table 1. Main differences between the policy training (*i.e.*, the policy phase) and value training (*i.e.*, the distillation phase) in PPG. In comparison, the joint training in PPO restricts the policy and value to be trained with the same (on-policy) data, the same level of sample reuse, and the same update frequency.

Aspects	π (policy phase)	V (distillation phase)
Data	on-policy	off-policy
Sample reuse	low	high
Update frequency	high	low

not critical. Feature distillation can be performed frequently without degrading the performance, as long as there is sufficiently strong policy regularization.

- High sample reuse is also not the key factor. Reducing the minibatch size while fixing the number of gradient updates (hence lower sample reuse) can achieve similar or even better performance. Besides, we can maintain PPG's high performance while reducing the computational cost to a similar level as PPO.
- Apart from policy regularization, the diversity of training data in the distillation phase is another key contributing factor to the performance.

2. Backgrounds

2.1. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is a model-free policy gradient method that optimizes a clipped surrogate objective

$$L^{clip} = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio to correct the discrepancy between current policy π_θ and the behavior policy $\pi_{\theta_{old}}$, $\hat{A}_t$ is the estimated advantage at timestep t, and $\hat{\mathbb{E}}[\dots]$ indicates the empirical average over a finite batch of timesteps t. An entropy bonus $H[\pi_\theta(\cdot|s)]$ is often used to ensure sufficient exploration. The total loss for training policy π_θ is

$$L_\pi = L^{clip} + \beta_H \hat{\mathbb{E}}_t \left[H[\pi_\theta(\cdot|s_t)] \right]$$

where β_H is a scalar hyper-parameter. PPO is commonly implemented in the actor-critic style (Konda & Tsitsiklis, 1999), where a state value function V_θ is learned for computing variance-reduced advantage. The value function is trained by optimizing

$$L_V = \hat{\mathbb{E}}_t \left[\frac{1}{2} \left(V_\theta - \hat{V}_t^{\text{targ}} \right)^2 \right]$$

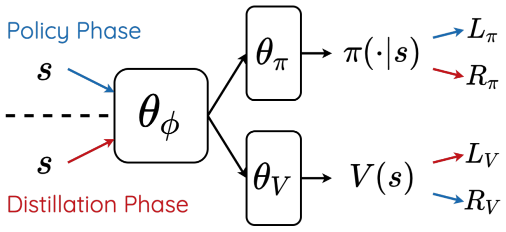


Figure 1. The network architecture of the single-network PPG.

where $\hat{V}_t^{\text{targ}}$ are value function targets. Both $\hat{A}$ and $\hat{V}^{\text{targ}}$ are often calculated using Generalized Advantage Estimation (GAE) (Schulman et al., 2016).

In practice, especially when it comes to high-dimensional visual inputs (*e.g.*, video games), the policy and the value function often share a torso network for feature extraction. The state inputs are first mapped to latent features by the torso network, and then the latent features are mapped to policy and value output by separate heads. Empirical evidence on Procgen (Cobbe et al., 2021) suggests that sharing parameters between the policy and the value function leads to clearly better performance than using disjoint networks, probably because this allows features trained by one objective to help better optimize the other.

In the rest of the paper, we denote the parameters of the shared feature encoder, the policy head, and the value head as θ_ϕ, θ_π, and θ_V respectively. For notation clarity, we slightly abuse θ in the subscript to denote all parameters in the policy and value function, *i.e.*, θ in V_θ refers to (θ_ϕ, θ_V) and θ in π_θ refers to $(\theta_\phi, \theta_\pi)$.

2.2. Phasic Policy Gradient (PPG)

Despite the benefits of feature sharing, the joint training of policy and value in PPO suffers two problems: potential interference between policy and value function optimization and the artificial restriction that policy and value function are trained with the same data. Phasic Policy Gradient (PPG) (Cobbe et al., 2021) addresses these problems by decoupling the training of the policy and value function. While PPG was initially developed to use disjoint policy and value networks, this is not the key to its superior performance, and using the single-network version can achieve comparable performance (see Section 3.6 in (Cobbe et al., 2021)). Therefore, in this paper, we concentrate our focus on the single-network PPG (illustrated in Figure 1).

Different from PPO which jointly trains the policy and value function, PPG decouples its training by alternating between a policy phase and a distillation phase.

In the policy phase, PPG optimizes the following loss:

$$L_\pi + \beta_V R_V$$

Algorithm 1 Single-network PPG framework

```
1:  Initialize a FIFO buffer $\mathcal{B}$ of size $T_{off} \times N_{env} \times M$ to store off-policy data
2:  for iteration $i = 1, 2, \ldots$ do
3:      // Data collection
4:      Roll out policy $\pi$ to obtain on-policy data $\mathcal{D}$ of size $N_{env} \times M$
5:      Add on-policy data $\mathcal{D}$ to buffer $\mathcal{B}$
6:      // Policy phase
7:      for policy update $= 1, 2, \ldots, K_\pi$ do
8:          From on-policy data $\mathcal{D}$, sample a minibatch $B$ of size $M_\pi$
9:          Optimize $\theta_\phi$, $\theta_\pi$ and $\theta_V$ with loss $L_\pi + \beta_V R_V$ on data $B$
10:     end for
11:     if $t \bmod T_{freq} = 0$ then
12:         // Distillation phase
13:         for value update $= 1, 2, \ldots, K_V$ do
14:             From off-policy buffer $\mathcal{B}$, sample a minibatch $B$ of size $M_V$
15:             Optimize $\theta_\phi$, $\theta_\pi$ and $\theta_V$ with loss $L_V + \beta_\pi R_\pi$ on data $B$
16:         end for
17:     end if
18: end for
```

T_{off}: buffer size, *i.e.*, how many recent iterations of data stored in the buffer
T_{freq}: number of iterations between two consecutive distillation phase
N_{env}: number of parallel environments
M: number of roll-out steps
K_π: number of policy updates per iteration
K_V: number of value updates per iteration
β_V: value regularization strength
β_π: policy regularization strength

where R_V is a regularizer that updates θ_V while minimizing interference with θ_ϕ. In the original PPG, R_V is L_V with gradients detached at the last layer of the shared encoder, such that it only updates θ_V. Alternatively, (Moon et al., 2022) proposes

$$R_V = \hat{\mathbb{E}}_t \left[\frac{1}{2} \left(V_\theta(s_t) - V_{\theta_{old}}(s_t) \right)^2 \right],$$

which regularizes the deviation between the updated value function and the old one before the policy phase. In both cases, β_V is a coefficient to balance the losses.

In the distillation phase, PPG optimizes the following loss:

$$L_V + \beta_\pi R_\pi,$$
$$\text{where } R_\pi = \hat{\mathbb{E}}_t \left[D_{\text{KL}} \left[\pi_{\theta_{old}}(\cdot|s_t) \mid \pi_\theta(\cdot|s_t) \right] \right]$$

regularizes the KL divergence between the updated policy and the old policy before the distillation phase. This phase essentially distills useful features learned from the value function objective to the policy. β_π is a hyperparameter to balance the losses.

Putting it together, PPG decouples the optimization of the policy and the value function: in the policy phase, we optimize L_π while regularizing V, and in the distillation phase, we optimize L_V while regularizing π. In each phase, the shared encoder is only updated by either L_π or L_V, thus reducing interference.

3. Experiments

By decoupling the optimization of the policy and value function objectives, PPG is able to train these two objectives differently to boost performance. The differences lie mainly in three aspects. Compared to the policy objective, the value function objective is trained

- much less frequently,
- using additional off-policy data,
- with a higher level of sample reuse.

In comparison, PPO trains both objectives at the same frequency, with the same (on-policy) data, and at the same level of sample reuse.

Among the three points listed above, the lower frequency and the higher sample reuse of the distillation phase (*i.e.*, where the value function is trained), are believed to be key to PPG's superior performance (Cobbe et al., 2021; Moon et al., 2022). However, the conclusion is drawn from ablation experiments that do not carefully disentangle different factors. For example, in (Cobbe et al., 2021), increasing the distillation frequency is coupled with reducing the amount of additional off-policy data. It is hard to tell which is the actual cause of the performance change.

To gain a clear understanding of what matters in PPG, in this section, we carefully control different factors and conduct a large-scale empirical study. Section 3.1 introduces the experimental settings. Section 3.2, 3.3, and 3.4 cover our empirical investigations regarding the three aspects respectively.

3.1. Settings

3.1.1. Algorithm Framework

The PPG framework used in our experiments is presented in Algorithm 1. As introduced in Section 2.2, there are

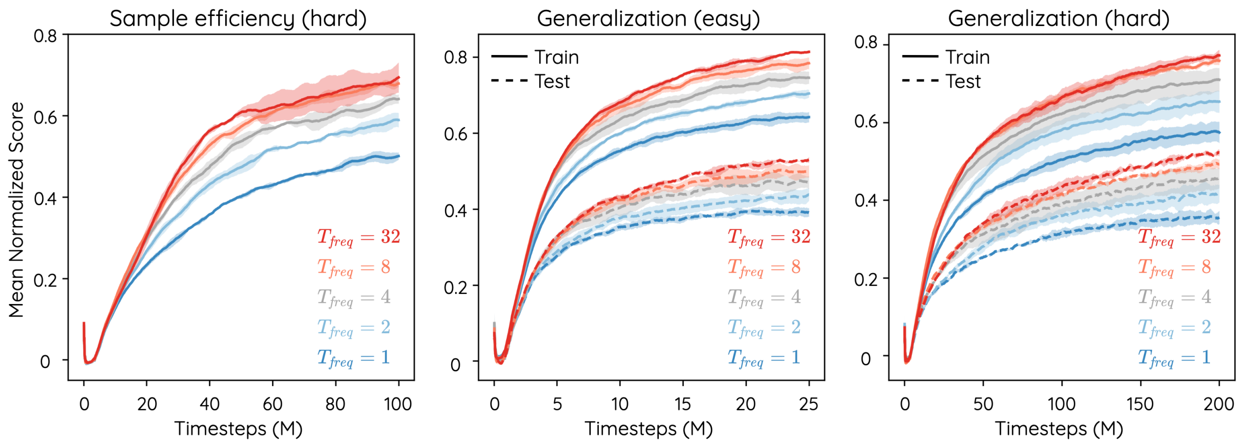


Figure 2. Performance with varying distillation interval T_{freq} and the default policy regularization strength ($\beta_\pi = 1$), normalized over all Procgen games.

two ways to regularize the value function during the policy phase (detaching the gradient or penalizing the deviation). In our study, we use the latter since it is shown to perform better (Moon et al., 2022). To facilitate later discussions, we make a small change in describing the algorithm: the number of epochs used in prior works is replaced with the number of gradient updates (Lines 7 and 13).

3.1.2. Training and Evaluation Configurations

The large-scale Procgen benchmark (Cobbe et al., 2020) is used as the testbed of our study. Procgen consists of 16 games that are designed to be highly diverse. Thus, we expect our findings on this benchmark might be useful in other RL environments. In Procgen games, the game level is randomly generated at the beginning of each episode, so the total number of levels is almost infinite. Procgen provides two difficulty choices for each game (*easy* and *hard*), which controls the complexity of the generated levels.

We consider two standard setups in Procgen:

- **sample efficiency** setup: the agent is trained and tested on the full distribution of levels,
- **generalization** setup: the agent is trained on a limited set of levels and tested on the full distribution of levels.

Previous works (Cobbe et al., 2021; Moon et al., 2022) only focus on one of the above two setups. In comparison, our study covers both, providing a comprehensive view of how different factors contribute to PPG. For the sample efficiency setup, we follow (Cobbe et al., 2021) to use the *hard* difficulty and train the agent for 100M steps. For the generalization setup, we consider both *easy* and *hard* difficulties and train the agent for 25M and 200M steps respectively. Other training details can be found in Appendix A. Notably, we reiterate here the default values of some important parameters: $T_{off} = 32$, $T_{freq} = 32$, $\beta_\pi = 1$. For a fair comparison, whenever the T_{freq} is changed, we also change the number of value updates K_V accordingly such that the total number of value updates remains the same.

The main performance metric is the mean normalized return averaged over all games in Procgen (see Section 2.2 in (Cobbe et al., 2020)). Each experiment is run with 3 random seeds and the standard deviations are plotted as shaded areas. In addition, we also report results using the rliable library (Agarwal et al., 2021) in Appendix B.4.

3.2. Distillation Frequency

We first take a closer look at how the frequency of the distillation phase affects performance. In prior works, Cobbe et al. (2021) find that performing the distillation phase too frequently has negative impacts. However, their ablation experiments do not control the amount of off-policy data in the buffer $\mathcal{B}$. The buffer size changes as the frequency changes, which may confound the results. In this subsection, we keep the buffer size fixed to eliminate its influence and focus on the interplay between the distillation frequency T_{freq} and another overlooked factor: policy regularization strength β_π. The influence of the buffer size will be investigated in the next subsection.

When the features are distilled from the value function to the policy, the policy regularization strength β_π controls the distortions to the policy between two policy phases. Thus, more frequent distillation leads to more policy distortions throughout policy optimization. We hypothesize that the increased policy distortion is the underlying cause of the performance degradation and that imposing a higher regularization strength can be a simple fix. First, we obtain the results of varying T_{freq} but the same regularization strength β_π. As shown in Figure 2, with the influence of data diversity removed, the performance seems to still suffer from

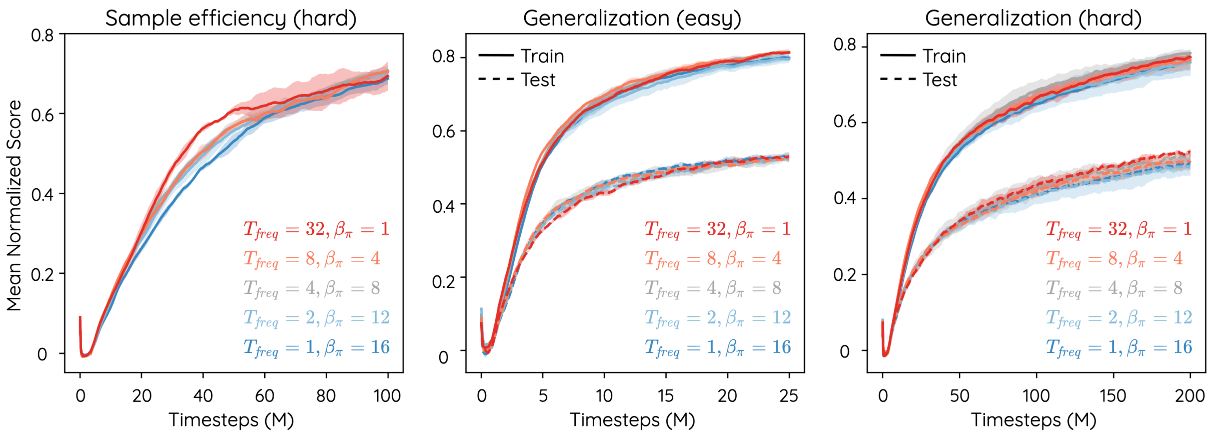


Figure 3. Performance with varying distillation interval T_{freq} and regularization strength β_π, normalized over all Procgen games.

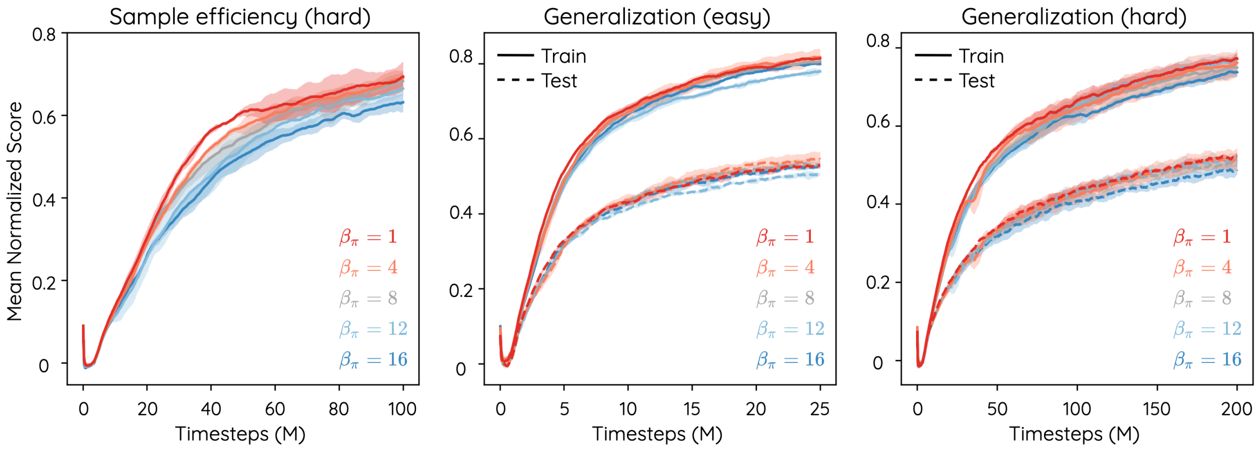


Figure 4. Performance with varying regularization strength β_π and infrequent distillation ($T_{freq} = 32$), normalized over all Procgen games.

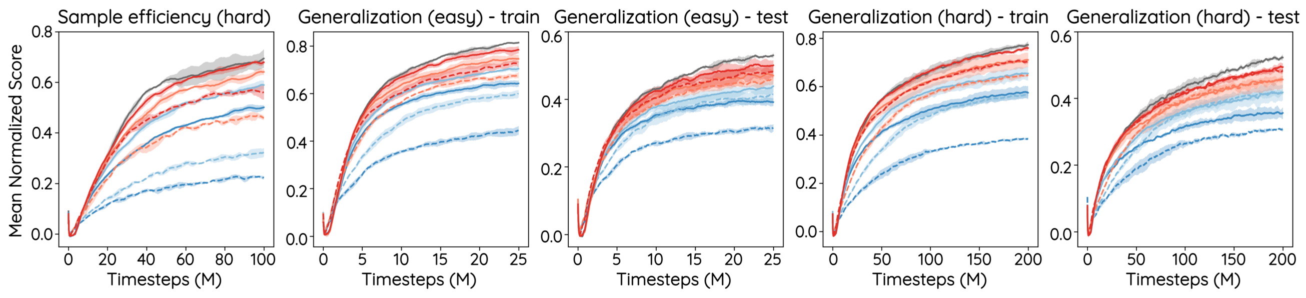
more frequent distillation. However, as Figure 3 shows, if we increase the policy regularization strength as the distillation phase becomes more frequent, we can easily reach a similar performance. In particular, we can perform the policy training phase and distillation phase at the same rate (*i.e.*, $T_{freq} = 1$) with almost no performance drop. Thus, we can remove Line 11 in Algorithm 1 and obtain a simplified PPG. In each iteration, the agent is first trained with policy loss and value regularization and then trained with value loss and policy regularization. We believe removing the infrequency requirement makes PPG easier to analyze since we do not need to consider policy updates across multiple rounds of data collection. On the practical side, this simplification also removes one tunable hyperparameter T_{freq}.

Finally, we complete our investigation with experiments that vary the regularization strength under frequent distillation ($T_{freq} = 32$). As shown in Figure 4, using higher β_π leads to similar or even worse performance. The results indicate that high regularization strength works not by improving performance on its own, but rather by fixing the increased policy distortions caused by frequent distillation.

3.3. Data Diversity

As mentioned earlier, in previous work (Cobbe et al., 2021), one factor that is entangled with the distillation frequency is the size of the off-policy data buffer. In the above section, we investigate the interplay between distillation frequency and policy regularization while fixing the buffer size to its default value ($T_{off} = 32$). We now turn to this factor and conduct experiments with varying T_{off}. A lower T_{off} means less diverse data in the buffer. In the extreme case ($T_{off} = 1$), only on-policy data is used for the distillation phase.

First, we fix the policy regularization strength ($\beta_\pi = 1$) and study how much the performance will be impacted if we reduce T_{off} in addition to reducing T_{freq}. This experiment helps elucidate if, and how much, the change in data diversity contributes to performance degradation. The results are shown in Figure 5. Comparing the solid and dashed lines of the same color and the gray line, we can see that reducing



legend	—	—	- - -	—	- - -	—	- - -	—	- - -
T_{off}	32	32	8	32	4	32	2	32	1
T_{freq}	32	8	8	4	4	2	2	1	1

Figure 5. Performance with varying buffer size T_{off} at different distillation frequencies, normalized over all Procgen games. Here β_π is fixed at 1.

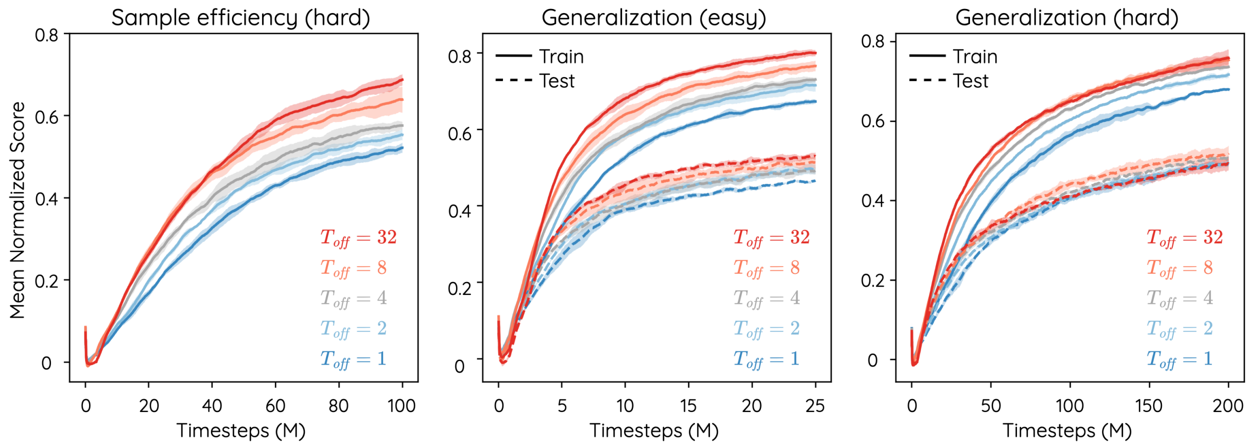


Figure 6. Performance with varying T_{off} under frequent distillation ($T_{freq} = 1$ and $\beta_\pi = 16$), normalized over all Procgen games.

T_{off} indeed leads to a significant performance drop. Please also refer to Figure 19 to Figure 23 in the appendix. In addition, this drop loosely correlates to the reduction in T_{off}, with changing T_{off} from 32 to 1 giving the largest decline.

In Section 3.2, we show that it is possible to recover comparable performance under frequent distillation by using a stronger policy regularization (see Figure 3). A natural question is: how does the data diversity affect the results in that case? To answer this question, we run experiments with varying T_{off} while setting $T_{freq} = 1$ and $\beta_\pi = 16$. As the results in Figure 6 show, using more diverse data (*i.e.*, a higher T_{off}) generally yields better performance. Only using the on-policy data (*i.e.*, $T_{off} = 1$) incurs a considerable performance drop.

In the above results for generalization setups, we observe that the influence of data diversity on the testing performance is less significant compared to that on the training performance. A possible explanation is that, whether T_{off} is low or high, the data in the buffer comes from a limited number of training levels. Thus, changes in the data diversity have a smaller influence on the testing performance.

Finally, we end this subsection with an experiment that varies the data diversity in an alternative way. Specifically, we fix $T_{off} = 1$ (*i.e.*, only using on-policy data) but scale the number of parallel environments. To keep the number of samples in each iteration unchanged, we sample a subset of the original size ($N_{env} \times M$) from the collected data and use it instead of all collected data as $\mathcal{D}$ for subsequent training. Building upon the results in the previous subsection, we run the experiments under frequent distillation ($T_{freq} = 1$ and $\beta_\pi = 16$). As the results in Figure 7 show, despite that only on-policy data is used, increasing the number of environments achieves a similar performance boost as using an off-policy buffer. This indicates that data diversity is indeed one of the factors that matter. We observe that the improvements of the final testing performance in the generalization (hard) setup are relatively small, probably because the default configuration $N_{env} = 256$ is already large enough to provide diverse data.

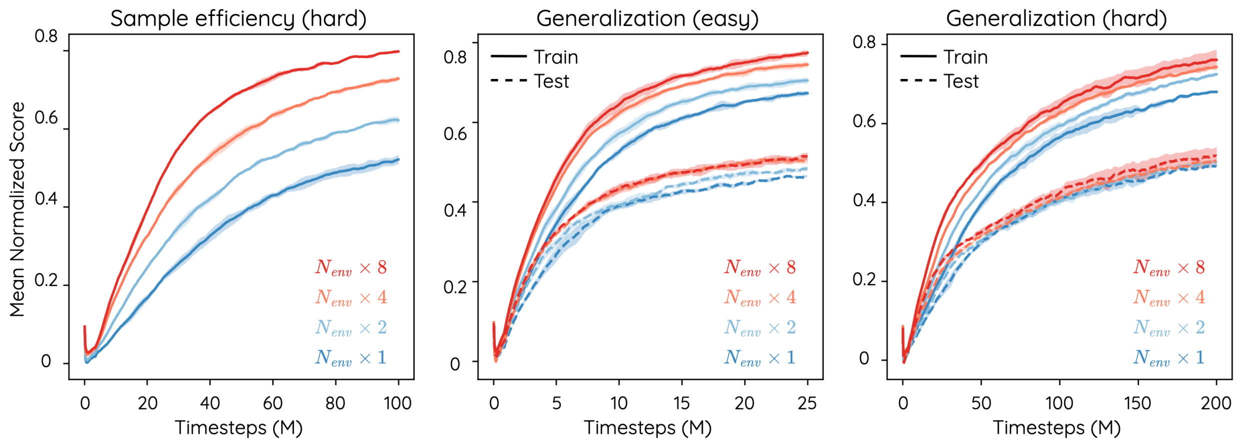


Figure 7. Performance with varying data diversity by scaling the number of parallel environments, normalized over all Procgen games.

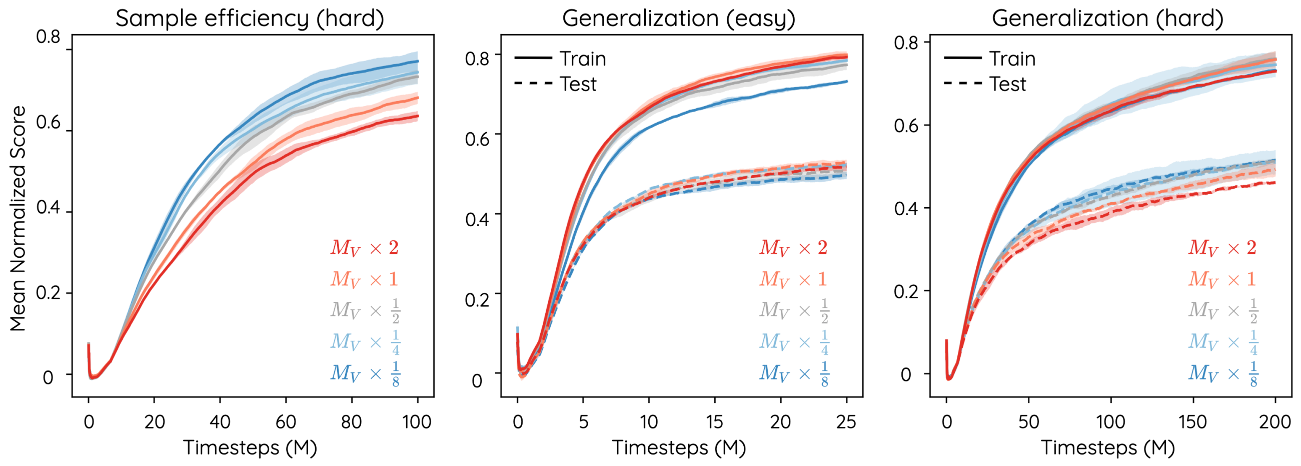


Figure 8. Performance with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), normalized over all Procgen games.

3.4. Sample Reuse

In this subsection, we look at the last aspect: sample reuse. In the prior work (Cobbe et al., 2021), high sample reuse in the distillation phase is attributed as one of the key contributing factors to PPG's good performance. Given a dataset (*e.g.*, the off-policy buffer $\mathcal{B}$), two parameters together determine the level of sample reuse: the number of updates (K_V) and the minibatch size (M_V). Previously, increasing sample reuse is done by increasing the number of updates while keeping the minibatch size fixed. It is possible that the performance boost is the result of more gradient updates rather than higher sample reuse.

To test this hypothesis, we conduct experiments by fixing the number of updates K_V while varying the minibatch size M_V. As the results in Figure 8 show, reducing the minibatch size does not cause a drop in the performance. The only noticeable drop occurs in the generalization (easy) setup, when the minibatch size is too small ($M_V \times {}^1\!/\!_8 = 128$). In some cases, especially the sample efficiency setup, a smaller minibatch size even leads to better results. The results suggest high sample reuse is not a key factor in PPG. We can achieve similar or even better performance with low sample reuse. We also run experiments in the infrequent distillation case (*i.e.*, $T_{freq} = 1$ and $\beta_\pi = 16$), and observe similar results (Figure 9 in the appendix).

One practical benefit of smaller minibatch size is shorter training time. Although the original PPG outperforms PPO, it also introduces additional training costs (due to more updates in the distillation phase). Our findings suggest that we can achieve the same high performance without sacrificing too much in training speed. Table 2 shows the average wall clock training time of each iteration under our hardware. We can see that using $M_V \times {}^1\!/\!_8$ halves the training time, resulting in similar time costs as PPO. While the numbers in Table 2 are certainly influenced by various things, we believe the relative gain in training speed is clear.

Table 2. Comparison of the wall clock training time for each iteration (in seconds), averaged over all runs and games.

	PPG with varying M_V					PPO
	$\times 2$	$\times 1$	$\times 1/2$	$\times 1/4$	$\times 1/8$	
EFF (HARD)	52.7	31.2	20.4	15.5	13.0	12.5
GEN (EASY)	13.3	8.5	6.2	5.0	4.6	3.9
GEN (HARD)	51.6	31.5	20.7	15.3	13.4	12.7

4. Discussions

Value regularization. In Section 3.2, we show that policy regularization is one of the key factors in PPG. Since Algorithm 1 has a pretty symmetric structure under frequent distillation (*i.e.*, removing Line 11), we are interested in how the counterpart of policy regularization, value regularization, affects the performance. Figure 10 in the appendix shows the results of varying value regularization strength β_V. As expected, using too strong regularization hampers policy learning and degrades performance.

Interestingly, applying no value regularization (*i.e.*, $\beta_V = 0$) only causes little performance drop or even improves testing performance in the generalization (hard) setup. In comparison, the performance will degrade greatly with no value regularization in the infrequent distillation case (Figure 11). The results indicate that we need stronger value regularization under infrequent distillation, which is opposite to our observations for policy regularization. Despite the symmetric structure, policy and value regularization seem to play quite different roles in the training process.

Stiffness and infrequent value update. Stiffness is a measure of how a small gradient step in the network's parameters on one example affects the loss on another example (Fort et al., 2019). It has been shown that a value network with lower stiffness is prone to memorizing the training data (Bengio et al., 2020). In the previous work, Moon et al. (2022) show that delayed value update (*i.e.*, infrequent distillation) leads to higher stiffness and hence helps mitigate overfitting. Since our results indicate that infrequent distillation is not necessary, we are interested in the stiffness measure in such cases. As shown in Figure 12 (see Appendix B.3), less frequent distillation in general correlates with higher stiffness. However, compared to Figure 13, it seems that the difference in stiffness is not closely correlated with the performance. We believe more works are needed to reach a better understanding of the interactions among stiffness, value update frequency, and overfitting.

5. Related Works

PPG is a reinforcement learning framework proposed in (Cobbe et al., 2021), aiming to resolve the optimization interference between policy and value function while preserving the benefits of sharing features between them. Following PPG, some works (Raileanu & Fergus, 2021; Moon et al., 2022) adapt this framework to improve the agent's zero-shot generalization ability. Raileanu & Fergus (2021) use the auxiliary head to approximate the advantage instead of the value, in order to reduce the spurious correlation between the observation and the value. Moon et al. (2022) introduce explicit value regularization to replace the value network training (with detached gradient) during the policy phase.

While these follow-up works focus on making modifications to PPG, our paper studies what are the key contributing factors in PPG. To the best of our knowledge, the only work that studies different design choices in PPG with comprehensive experiments is the original PPG paper. However, as we show in our paper, the two key factors identified before are actually not that critical. In addition, the previous study only focuses on the sample efficiency setup while our work covers both sample efficiency and generalization setups.

Related to our findings, Aitchison & Sweetser (2022) use a distillation loss very similar to the one in PPG and they observe that a smaller batch size works better when optimizing this loss. Apart from Procgen, PPG has also been successfully applied in the very challenging Minecraft environment (Baker et al., 2022). Thus, we believe our findings might be useful beyond Procgen games.

6. Conclusion and Future Works

In this paper, we investigated what factors matter in PPG, a high-performing actor-critic method. Through comprehensive experiments on the Procgen benchmark, we show that the two factors that were believed to be critical, high level of sample reuse and low frequency of feature distillation, turn out to be not the deciding factors. Instead, we unveil that policy regularization and data diversity are what actually matter in PPG. In addition, our findings suggest that we can preserve PPG's high performance while reducing the computation cost (by more than half) to a similar level as PPO.

We have shown that varying policy regularization strength has a significant influence on the training process (*e.g.*, distillation frequency). One interesting topic for future works is to study how policy regularization and value loss influence each other, and how the learned representation evolves. Moreover, what if we use another auxiliary objective other than the value loss? Besides, it is also worthwhile to explore the use of offline data for the distillation phase. Finally, we note that data diversity can be measured and improved consciously, opening up the opportunity for further improvements in the learning process.

Acknowledgement

This work was partially funded by the Israel Science Foundation under Contract 2199/20.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Aitchison, M. and Sweetser, P. DNA: Proximal policy optimization with a dual network architecture. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=WHFgQLRdKf9.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 507–517. PMLR, 2020. URL http://proceedings.mlr.press/v119/badia20a.html.

Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. Video pretraining (VPT): learning to act by watching unlabeled online videos. *CoRR*, abs/2206.11795, 2022. doi: 10.48550/arXiv.2206.11795. URL https://doi.org/10.48550/arXiv.2206.11795.

Bengio, E., Pineau, J., and Precup, D. Interference and generalization in temporal difference learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 767–777. PMLR, 2020. URL http://proceedings.mlr.press/v119/bengio20a.html.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR, 2020. URL http://proceedings.mlr.press/v119/cobbe20a.html.

Cobbe, K., Hilton, J., Klimov, O., and Schulman, J. Phasic policy gradient. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2020–2027. PMLR, 2021. URL http://proceedings.mlr.press/v139/cobbe21a.html.

Dangel, F., Kunstner, F., and Hennig, P. BackPACK: Packing more into backprop. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJlrF24twB.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1406–1415. PMLR, 2018. URL http://proceedings.mlr.press/v80/espeholt18a.html.

Fort, S., Nowak, P. K., and Narayanan, S. Stiffness: A new perspective on generalization in neural networks. *CoRR*, abs/1901.09491, 2019. URL http://arxiv.org/abs/1901.09491.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 1008–1014. The MIT Press, 1999. URL http://papers.nips.cc/paper/1786-actor-critic-algorithms.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Balcan, M. and Weinberger, K. Q. (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html.

Moon, S., Lee, J., and Song, H. O. Rethinking value function learning for generalization in reinforcement learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=JkEz1fqN3hX.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Raileanu, R. and Fergus, R. Decoupling value and policy for generalization in reinforcement learning. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8787–8798. PMLR, 2021. URL http://proceedings.mlr.press/v139/raileanu21a.html.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

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

A. Training Details

For all experiments, we use the residual convolutional networks in IMPALA (Espeholt et al., 2018) and the Adam optimizer (Kingma & Ba, 2015), following previous practices (Cobbe et al., 2020; 2021). We use PyTorch (Paszke et al., 2019) as a deep learning framework. All experiments are conducted using Intel® Xeon® Platinum 8260 CPU and NVIDIA V100 GPU. The table below lists the default values of the parameters in our experiments. Note that when we decrease T_{freq}, we also reduce K_V accordingly such that the total number of gradient updates remains the same. For example, when $T_{\mathit{freq}} = 1$, $K_V = 3072/32 = 96$.

	Sample Efficiency (hard)	Generalization (easy)	Generalization (hard)
T_{off}	32	32	32
T_{freq}	32	32	32
N_{env}	256	64	256
M	256	256	256
K_π	8	8	8
K_V	3072	3072	3072
M_π	8192	2048	8192
M_V	4096	1024	4096
β_V	1	1	1
β_π	1	1	1
Discount factor γ	0.999	0.999	0.999
GAE parameter λ	0.95	0.95	0.95
PPO clip range (ϵ)	0.2	0.2	0.2
Reward normalization?	Yes	Yes	Yes
Entropy bonus coefficient β_H	0.01	0.01	0.01
Learning rate	5e-4	5e-4	5e-4
Maximum gradient norm	0.5	0.5	0.5
Total timesteps	100M	25M	200M
Number of training levels	All	200	500
LSTM?	No	No	No
Frame stack?	No	No	No

B. More Experiment Results

B.1. Scaling minibatch size under infrequent distillation

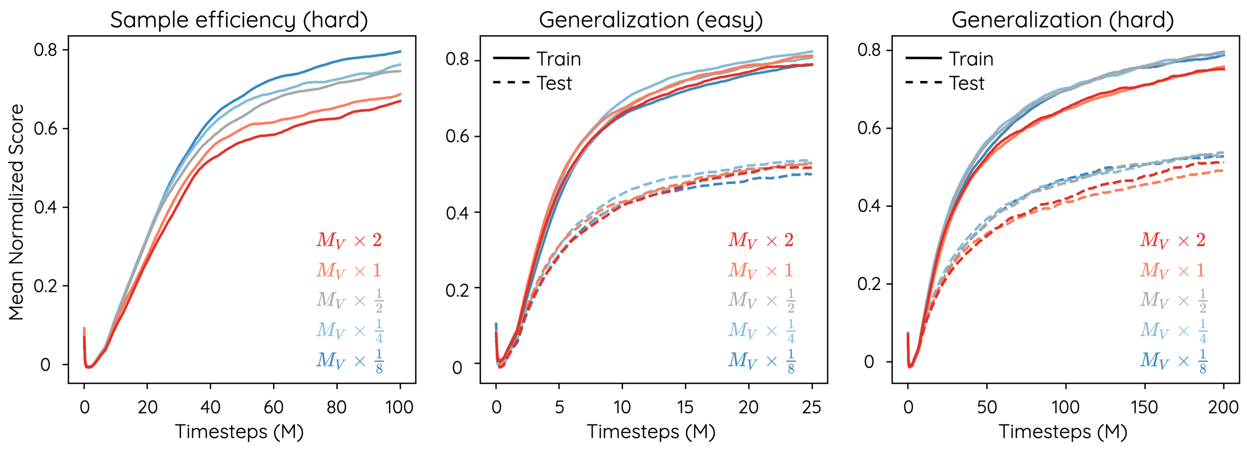


Figure 9. Performance with scaled minibatch sizes under infrequent distillation ($T_{freq} = 32$, $\beta_\pi = 1$), normalized over all Procgen games.

B.2. Varying value regularization strength β_V

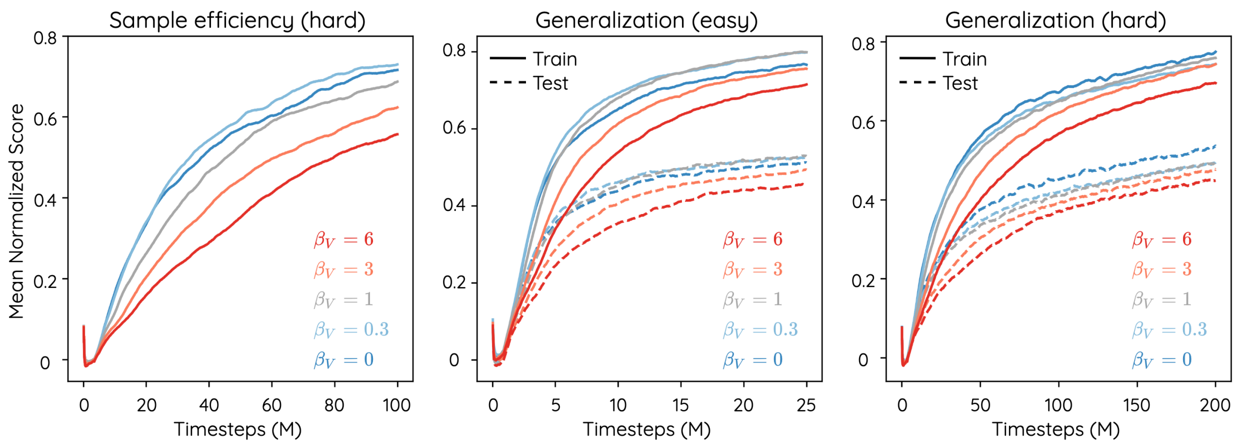


Figure 10. Performance with varying β_V under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), normalized over all Procgen games.

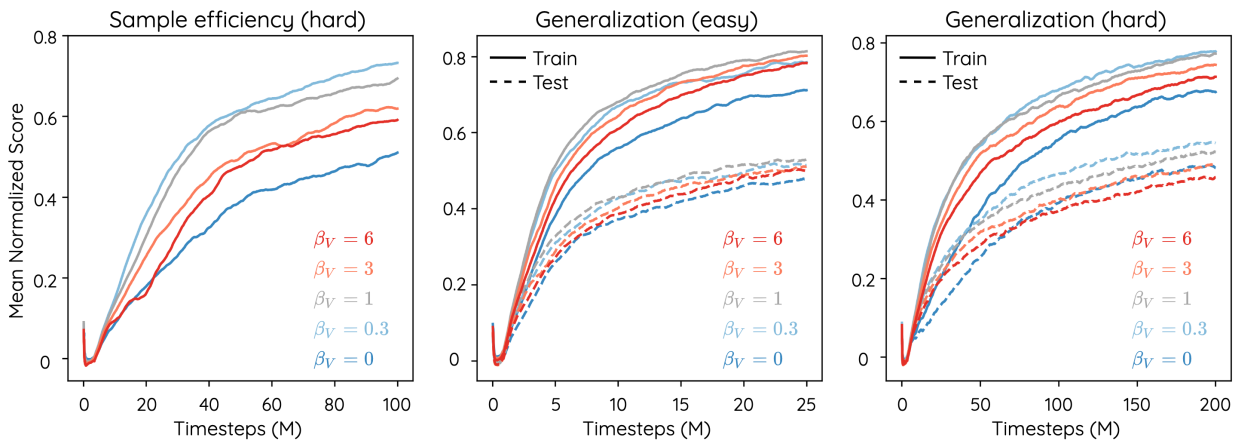


Figure 11. Performance with varying β_V under infrequent distillation ($T_{freq} = 32$, $\beta_\pi = 1$), normalized over all Procgen games.

B.3. Stiffness Analysis

Following (Moon et al., 2022), we measure the stiffness of the value objective gradients between states. Specifically, the stiffness of the value objective gradients between states (s, s') is defined by

$$\rho(s, s') = \frac{\nabla_\theta L_V(s)^\top \nabla_\theta L_V(s')}{\|\nabla_\theta L_V(s)\|_2 \|\nabla_\theta L_V(s')\|_2}$$

where $\theta = (\theta_\phi, \theta_V)$ refers to the parameters of the value function, and $\|\cdot\|_2$ denotes L2 norm. We run the experiments in the generalization (easy) setup. At each iteration, we first compute the individual value objective gradient for each state in the batch using BackPACK (Dangel et al., 2020), then calculate the mean stiffness of the gradients across all state pairs. Finally, the results averaged from 3 runs are shown in the figure below.

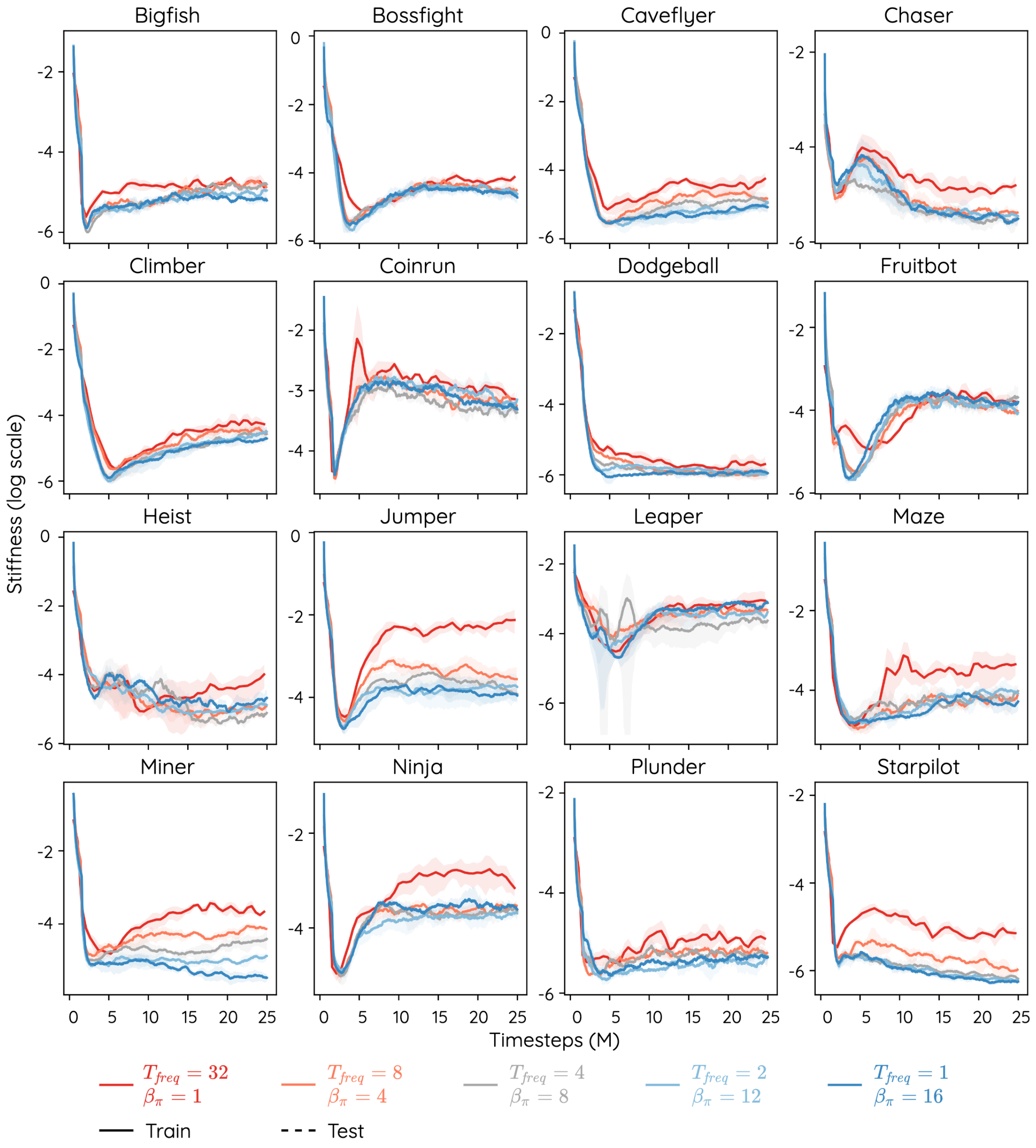


Figure 12. Stiffness measure under different distillation interval T_{freq}.

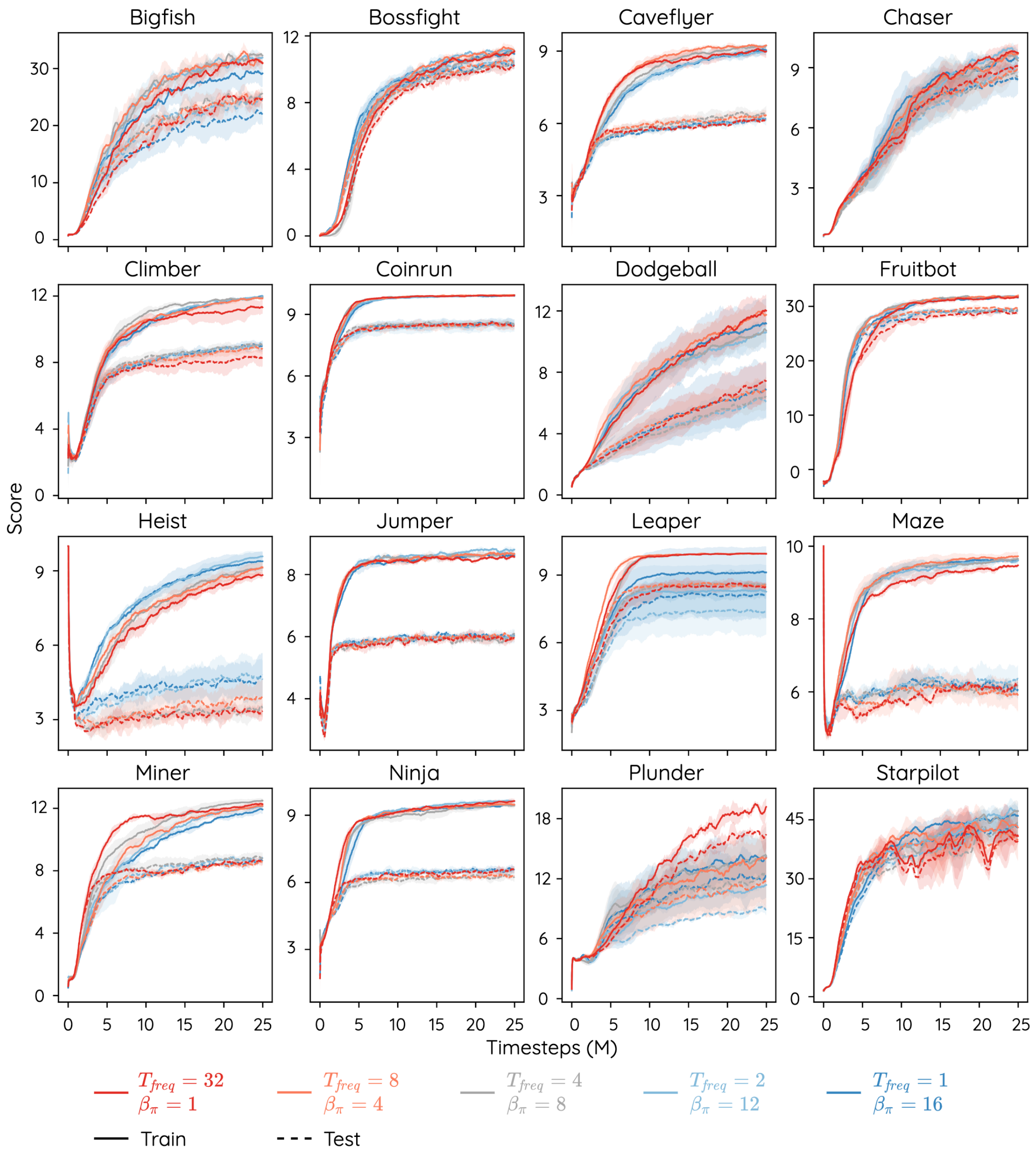


Figure 13. Per-game performance in the generalization (easy) setup, with varying T_{freq} and β_π. See Figure 3 for the normalized result.

B.4. Evaluation results using the rilable library (Agarwal et al., 2021)

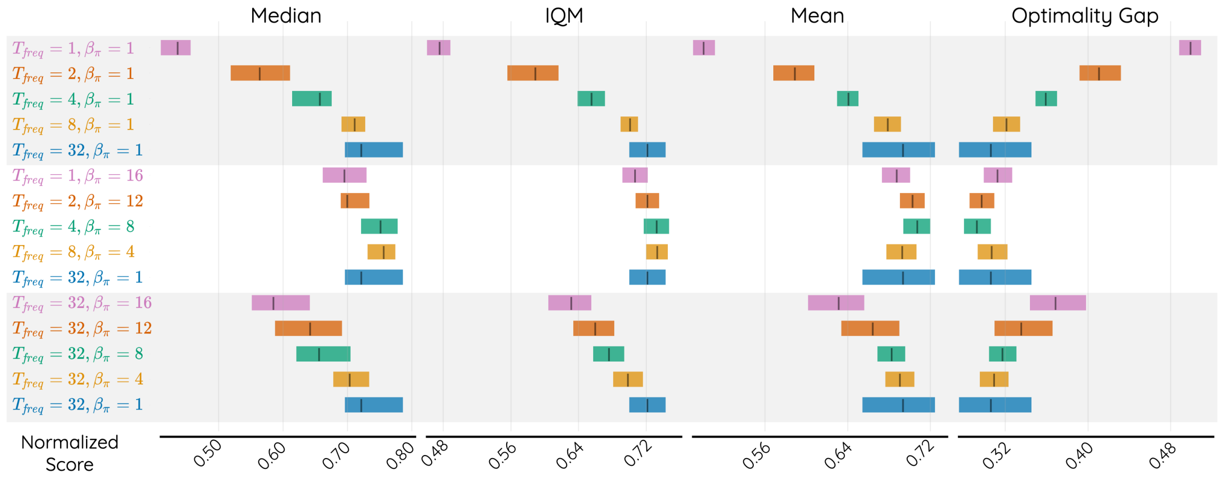


Figure 14. Performance in the sample efficiency (hard) setup with varying T_{freq} and β_π, corresponding to Figure 2, 3 and 4.

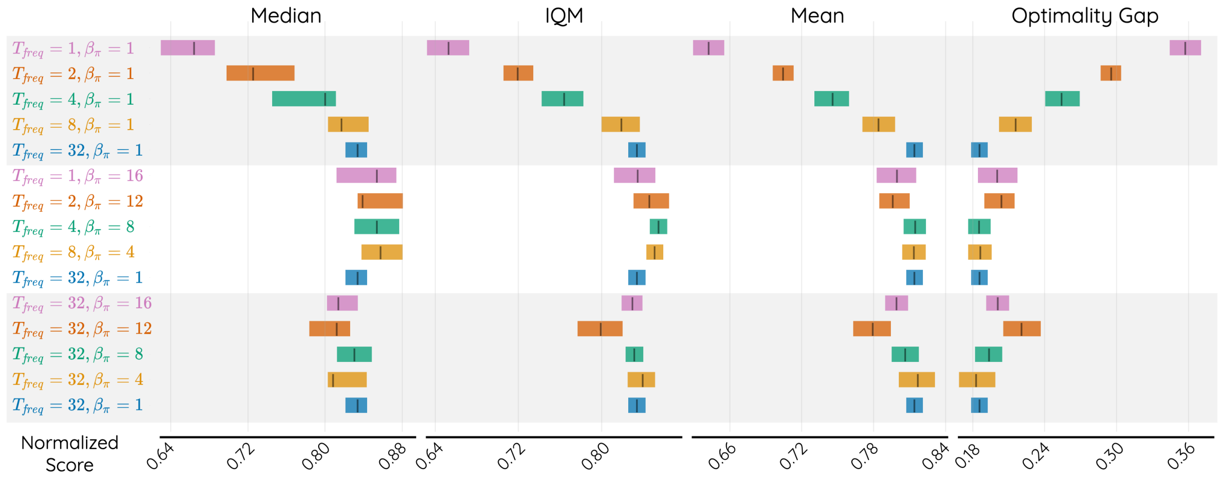


Figure 15. Training performance in the generalization (easy) setup with varying T_{freq} and β_π, corresponding to Figure 2, 3 and 4.

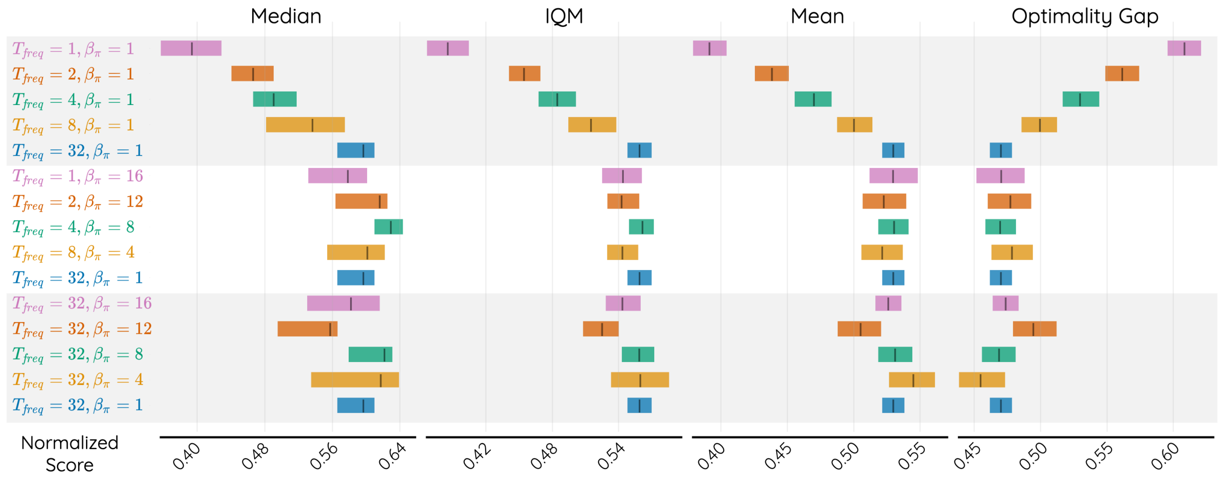


Figure 16. Testing performance in the generalization (easy) setup with varying T_{freq} and β_π, corresponding to Figure 2, 3 and 4.

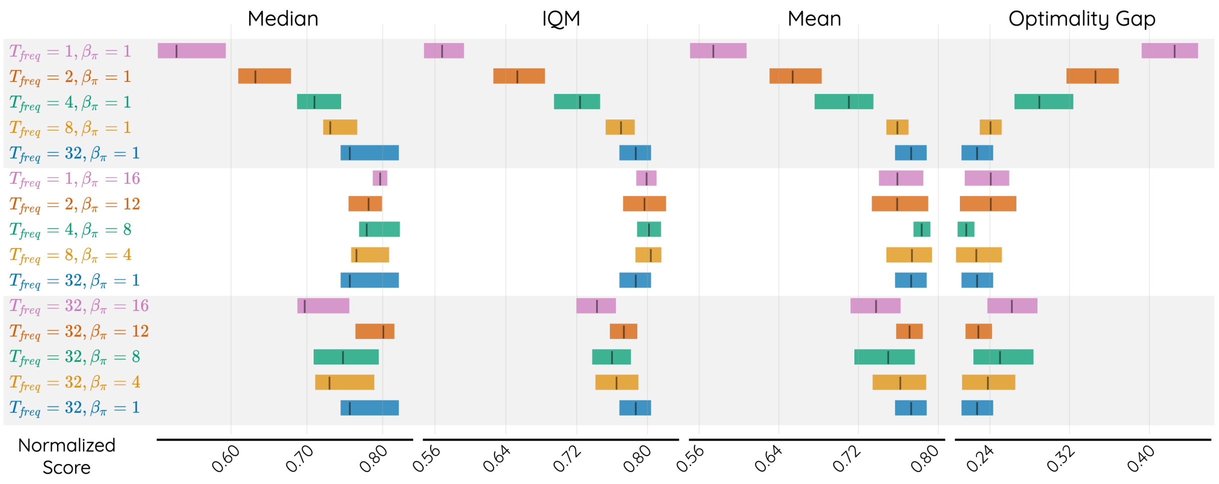


Figure 17. Training performance in the generalization (hard) setup with varying T_{freq} and β_π, corresponding to Figure 2, 3 and 4.

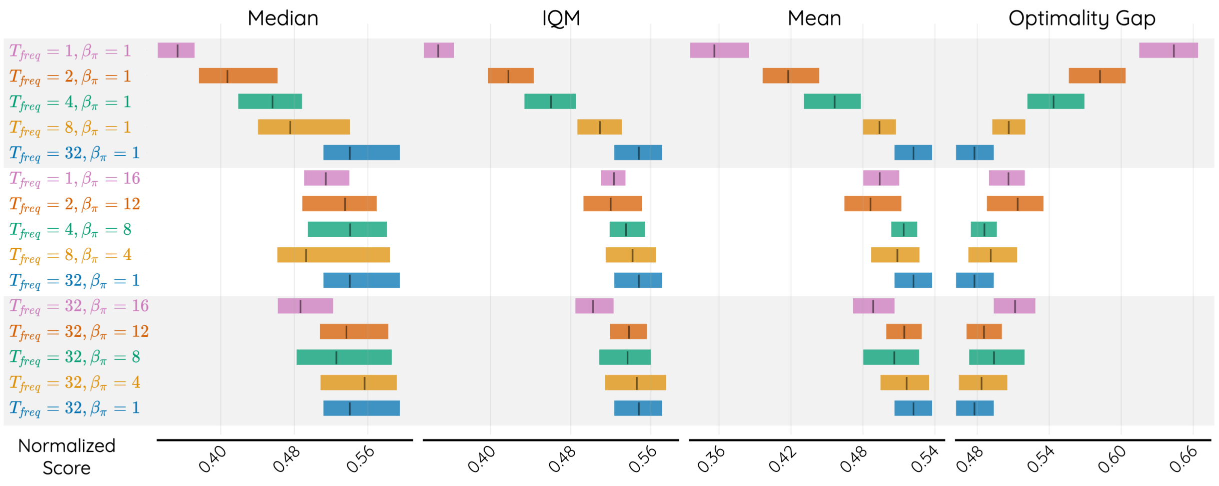


Figure 18. Testing performance in the generalization (hard) setup with varying T_{freq} and β_π, corresponding to Figure 2, 3 and 4.

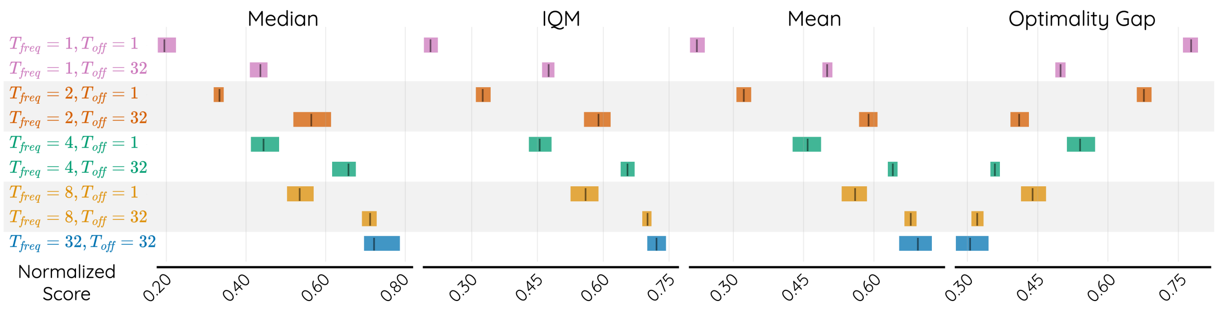


Figure 19. Performance in the sample efficiency (hard) setup with varying T_{off} at different T_{freq}, corresponding to Figure 5.

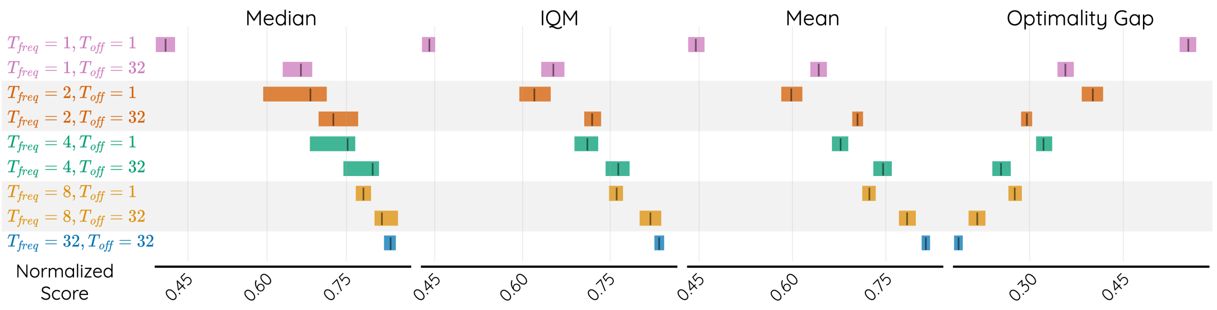


Figure 20. Training performance in the generalization (easy) setup with varying T_{off} at different T_{freq}, corresponding to Figure 5.

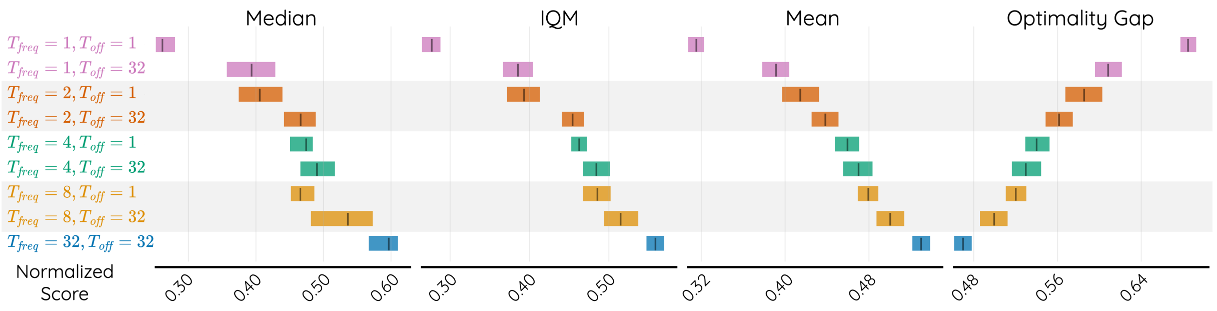


Figure 21. Testing performance in the generalization (easy) setup with varying T_{off} at different T_{freq}, corresponding to Figure 5.

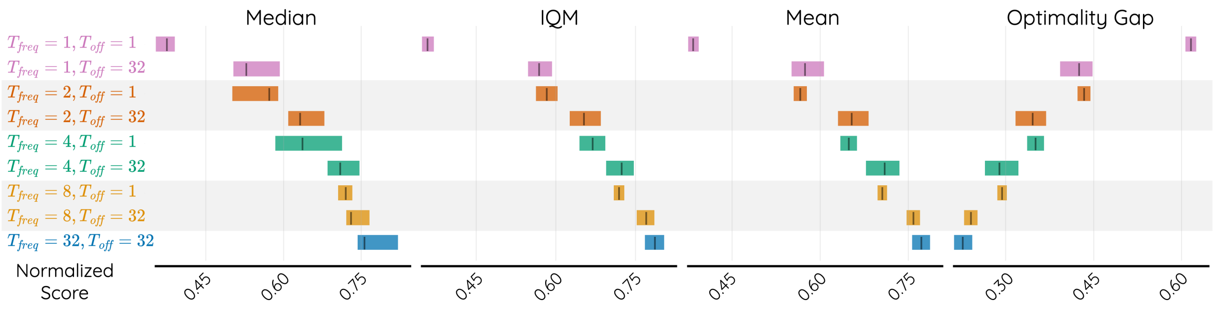


Figure 22. Training performance in the generalization (hard) setup with varying T_{off} at different T_{freq}, corresponding to Figure 5.

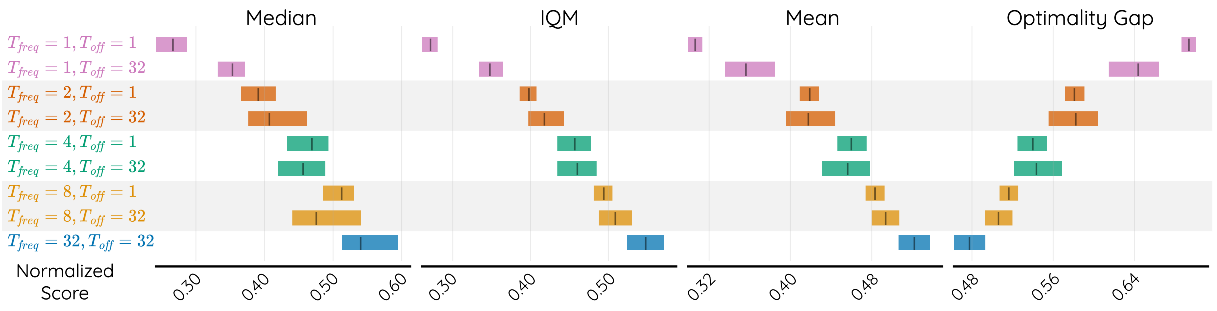


Figure 23. Testing performance in the generalization (hard) setup with varying T_{off} at different T_{freq}, corresponding to Figure 5.

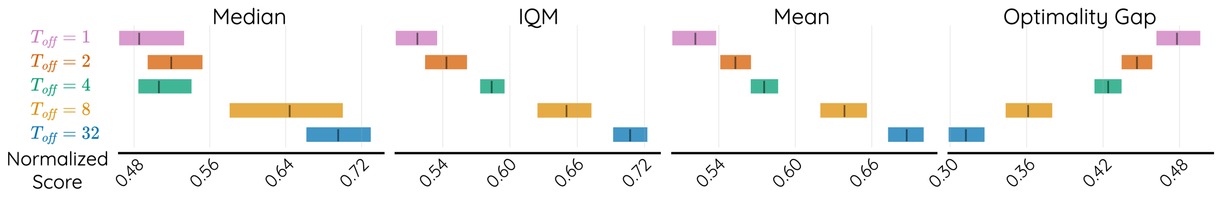


Figure 24. Performance in the sample efficiency (hard) setup with varying T_{off} under $T_{freq} = 1$ and $\beta_\pi = 16$, corresponding to Figure 6.

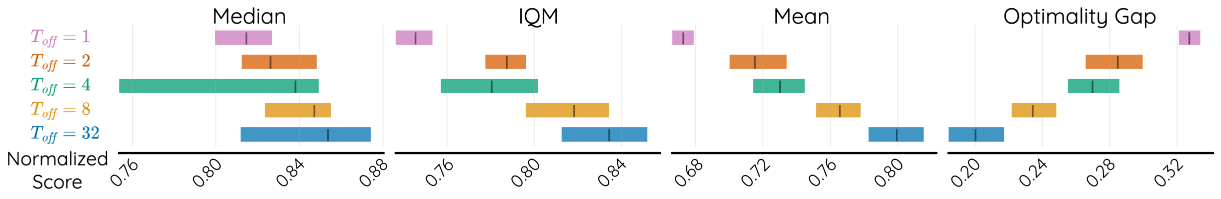


Figure 25. Training performance in the generalization (easy) setup with varying T_{off} under $T_{freq} = 1$ and $\beta_\pi = 16$, corresponding to Figure 6.

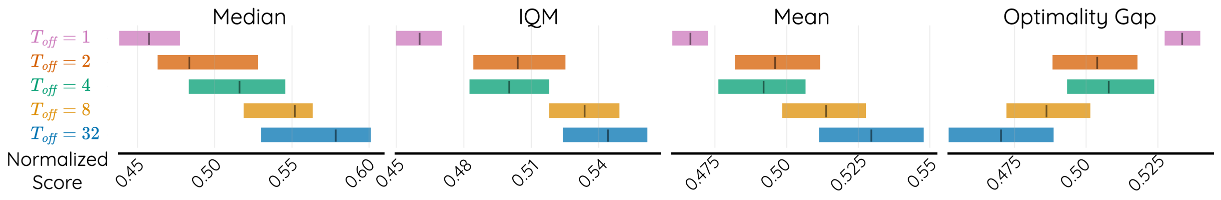


Figure 26. Testing performance in the generalization (easy) setup varying T_{off} under $T_{freq} = 1$ and $\beta_\pi = 16$, corresponding to Figure 6.

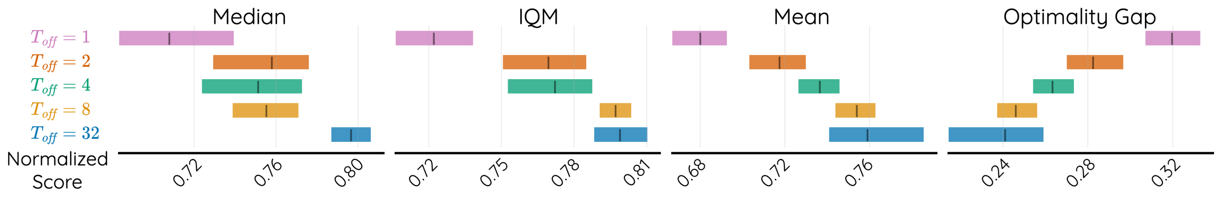


Figure 27. Training performance in the generalization (hard) setup with varying T_{off} under $T_{freq} = 1$ and $\beta_\pi = 16$, corresponding to Figure 6.

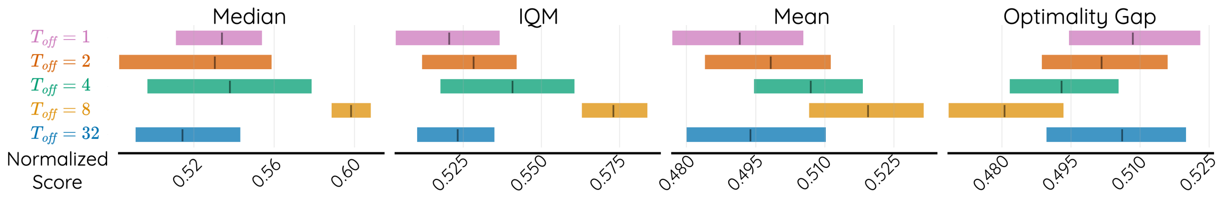


Figure 28. Testing performance in the generalization (hard) setup varying T_{off} under $T_{freq} = 1$ and $\beta_\pi = 16$, corresponding to Figure 6.

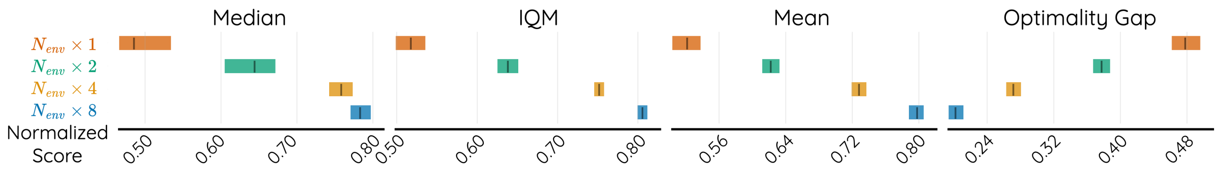


Figure 29. Performance in the sample efficiency (hard) setup with varying data diversity by scaling the number of parallel environments, corresponding to Figure 7.

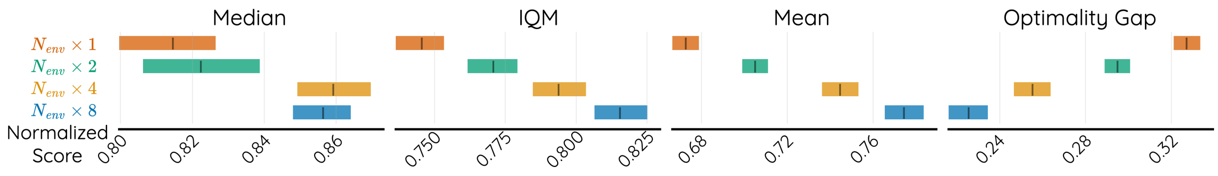


Figure 30. Training performance in the generalization (easy) setup with varying data diversity by scaling the number of parallel environments, corresponding to Figure 7.

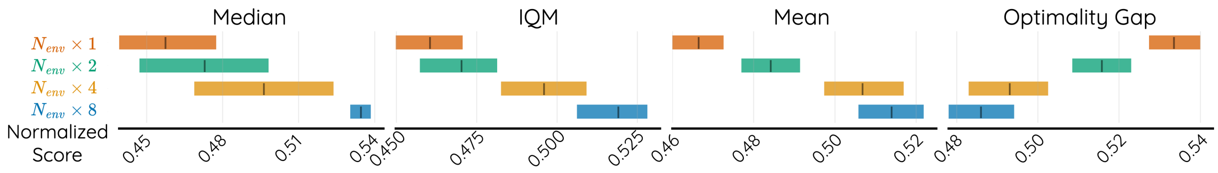


Figure 31. Testing performance in the generalization (easy) setup varying data diversity by scaling the number of parallel environments, corresponding to Figure 7.

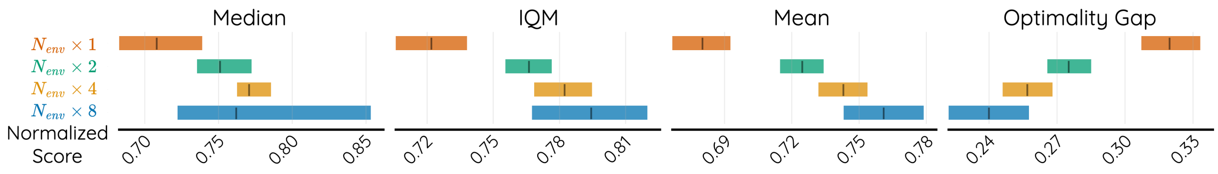


Figure 32. Training performance in the generalization (hard) setup with varying data diversity by scaling the number of parallel environments, corresponding to Figure 7.

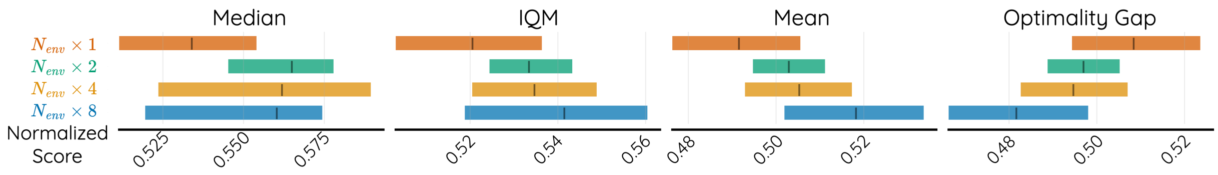


Figure 33. Testing performance in the generalization (hard) setup varying data diversity by scaling the number of parallel environments, corresponding to Figure 7.

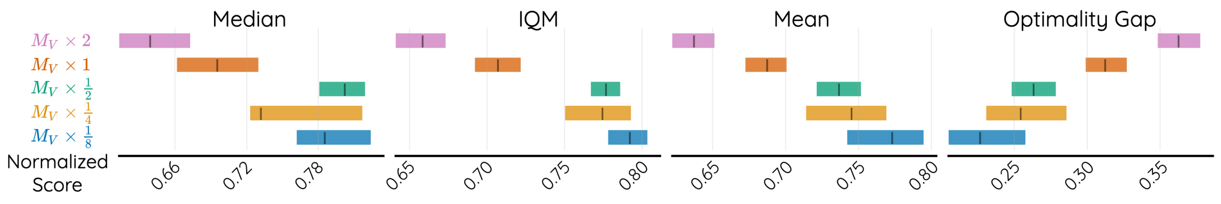


Figure 34. Performance in the sample efficiency (hard) setup with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), corresponding to Figure 8.

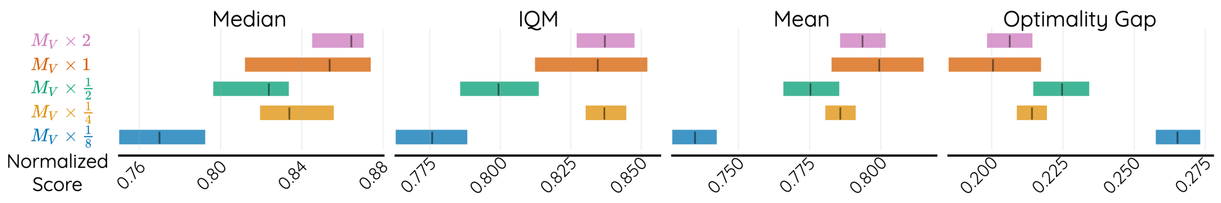


Figure 35. Training performance in the generalization (easy) setup with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), corresponding to Figure 8.

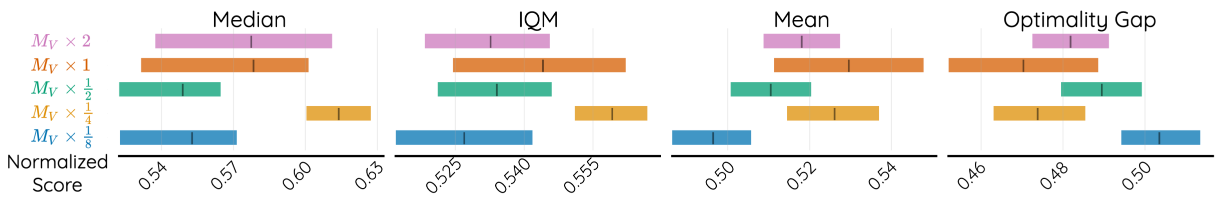


Figure 36. Testing performance in the generalization (easy) setup with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), corresponding to Figure 8.

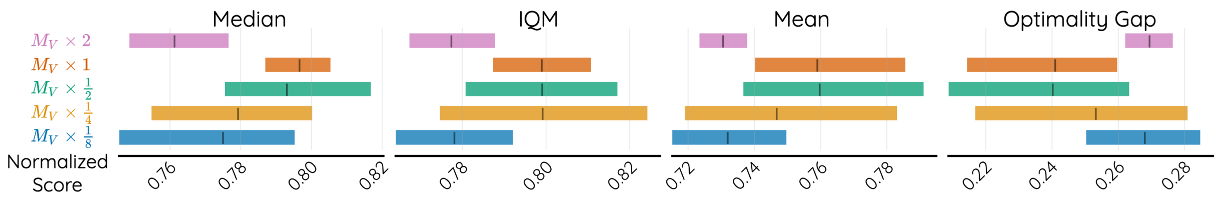


Figure 37. Training performance in the generalization (hard) setup with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), corresponding to Figure 8.

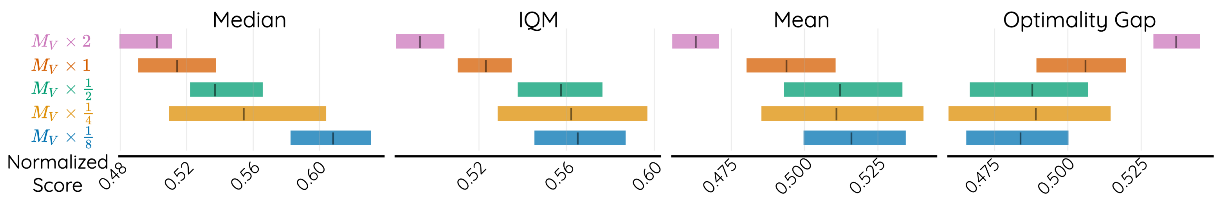


Figure 38. Testing performance in the generalization (hard) setup with scaled minibatch sizes under frequent distillation ($T_{freq} = 1$, $\beta_\pi = 16$), corresponding to Figure 8.