Proximal Policy Distillation

Anonymous authors
Paper under double-blind review

Abstract

We introduce Proximal Policy Distillation (PPD), a novel policy distillation method that integrates student-driven distillation and Proximal Policy Optimization (PPO) to increase sample efficiency and to leverage the additional rewards that the student policy collects during distillation. To assess the efficacy of our method, we compare PPD with two common alternatives, student-distill and teacher-distill, over a wide range of reinforcement learning environments that include discrete actions and continuous control (ATARI, Mujoco, and Procgen). For each environment and method, we perform distillation to a set of target student neural networks that are smaller, identical (self-distillation), or larger than the teacher network. Our findings indicate that PPD improves sample efficiency and produces better student policies compared to typical policy distillation approaches. Moreover, PPD demonstrates greater robustness than alternative methods when distilling policies from imperfect demonstrations. The code for the paper is released as part of a new Python library built on top of stable-baselines3 to facilitate policy distillation: `<AnonymizedGitHubRepository>`.

1 Introduction

Deep Reinforcement Learning (DRL) has been used to autonomously learn advanced behaviors on a wide range of challenging tasks, spanning from difficult games (Silver et al., 2017; Vinyals et al., 2019; Berner et al., 2019) to the control of complex robot bodies (Akkaya et al., 2019; Liu et al., 2022b; Haarnoja et al., 2024), and advanced generalist agents (Reed et al., 2022; Abramson et al., 2020; Octo Model Team et al., 2024).

However, optimizing the performance of DRL agents typically requires extensive trial-and-error, requiring many iterations to identify effective reward functions and appropriate neural network architectures. As such, considerable research effort has been made to improve the sample efficiency of reinforcement learning. A particularly effective approach is to leverage prior knowledge -such as that obtained from previous training runs (Agarwal et al., 2022)- through policy distillation (PD) (Rusu et al., 2016). Policy distillation, inspired by knowledge distillation (KD) in supervised learning (Hinton et al., 2015), focuses on transferring knowledge from a 'teacher' neural network to a 'student' network that usually differs in architecture or hyperparameters.

Despite the conceptual similarity between KD and PD, their use and implementation are very different. KD is mainly used for model compression, whereby functions learned by a high-capacity neural network are transferred to a smaller network. This helps to reduce inference time, which is crucial to run the final model on embedded hardware. This setting is not common in DRL, as the neural networks are typically kept small to shorten the long training times.

In contrast, PD is utilized differently in DRL. For example, a smaller model that is cheap to train can be 'reincarnated' into a larger one that is more expressive, possibly with different hyperparameters (Agarwal et al., 2022), to achieve better performance while limiting the computational resources required. Another effective use of PD in DRL is to use teacher model(s) as 'skills priors', e.g., useful component behaviors that solve sub-tasks of a problem, to make it easier for a student agent to learn to solve more complex tasks (Merel et al., 2019; Liu et al., 2022b; Zhuang et al., 2023). In this case, PD can be used as a regularizing term to bias towards behaviors that are expected to be useful for the larger, more complex task, or to simply combine a number of single-task teachers into a single multitask agent (Rusu et al., 2016). Finally, policy distillation can be used to re-map the types of inputs that a neural network receives. This is useful for

example in robotics, where policies trained on state-based (or even privileged) observations can be distilled into vision-based policies (Akkaya et al., 2019; Liu et al., 2022a).

Regarding implementation, KD typically involves adding a distillation loss to the problem-specific (dataset) loss function, to train a student network to reproduce the same output values as the teacher model. On the other hand, most policy distillation methods only use a distillation loss, training the student through supervised learning rather than in a reinforcement learning setting. However, this overlooks the valuable information that can be gained from the rewards that the student collects during distillation. Exploiting these rewards could accelerate the distillation process, potentially enable the student to outperform the teacher, and reduce the risk of overfitting to imperfect teachers.

We introduce Proximal Policy Distillation (PPD), a novel policy distillation method that combines student-driven distillation and Proximal Policy Optimization (PPO) (Schulman et al., 2017). PPD enhances PPO by incorporating a distillation loss to either perform traditional distillation, or to act as skills prior. Specifically, PPD enables the student to accelerate learning through distillation from a teacher, while potentially surpassing the teacher's performance.

The main **contributions** of this paper are:

1. We introduce Proximal Policy Distillation (PPD), a novel policy distillation method that improves sample efficiency and final performance by combining student-driven distillation with policy updates by PPO.
2. We perform a thorough evaluation of PPD against two common methods for policy distillation, student-distill (on-policy distillation) and teacher-distill (Czarnecki et al., 2019), over a wide range of RL environments spanning discrete and continuous action spaces, and out-of-distribution generalization. We assess the performance of the three methods with smaller, identical (self-distillation), and larger student network sizes, compared to the teacher networks.
3. We analyze the robustness of PPD compared to the two baselines in a scenario with 'imperfect teachers', whose parameters are artificially corrupted to decrease their performance.
4. We release a new Python library, *sb3-distill* [1], which implements the three methods within the stable-baselines3 framework (Raffin et al., 2019) to improve access to useful policy distillation methods.

2 Proximal Policy Distillation

Problem setting. We consider a reinforcement learning setting based on the Markov Decision Process $(\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$ (Sutton & Barto, 2018). An agent interacts with the environment in discrete timesteps $t = 0, \ldots, T-1$ that together make up episodes. On the first timestep of each episode, the initial state of the environment is sampled from the initial state distribution $s_0 \sim \rho_0(s)$. Then, at each timestep the agent selects an action $a_t \in \mathcal{A}$ using the policy function $\pi_\theta : \mathcal{S} \to \mathcal{A}$, represented by a neural network with parameters θ that takes states $s_t \in \mathcal{S}$ as input. The environment dynamics are determined by the transition function $p : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ (i.e., $s_{t+1} \sim p(s_t, a_t)$). The agent receives rewards at each timestep depending on the previous state transition according to a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The objective of the reinforcement learning agent is to find an optimal policy $\pi^\star$ that maximizes the expected sum of discounted rewards

$$\pi^\star = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \gamma^t r_t | a_t \sim \pi(s_t), s_0 \sim \rho_0(s), s_{t+1} \sim p(s_t, a_t)\right]$$

In the context of policy distillation, we have access to a teacher agent that has been trained to solve a reinforcement learning task within the MDP framework. We particularly focus on teachers trained via actor-critic methods. Policy distillation then starts with a new student policy that is randomly initialized.

[1] `<AnonymizedGitHubRepository>`

Our objective is to transfer the knowledge from the teacher policy to the student while simultaneously training the student on a task using reinforcement learning. The new task can be the same as the teacher's (e.g., to implement reincarnating RL (Agarwal et al., 2022), or to perform model compression), or a new one (e.g., using the teacher as skills prior (Merel et al., 2019; Liu et al., 2022b; Tirumala et al., 2022)). The student should learn as efficiently as possible, and ideally be robust to imperfect teachers, so that the performance of the student can in principle be higher than the teacher's.

Approach. We build on the framework developed by Czarnecki et al. (2019) since in its general form it already allows for the inclusion of environment rewards during distillation. The general form of the distillation gradient in this framework is given by

$$\nabla_\theta L(\theta) = \mathbb{E}_{q_\theta}\left[\sum_{t=1}^{|\tau|} -\nabla_\theta \log \pi_\theta(a_t|\tau_t)\widehat{R_t} + \nabla_\theta \ell(\pi_\theta, V_{\pi_\theta}, \tau_t)\right] \tag{1}$$

We focus in particular on the case where the *student* is used as the exploration policy q_θ to collect trajectories $\tau = \{s_t, a_t, r_t, s_{t+1}, a_{t+1}, \ldots, r_{t+N}\}$ by interacting with the environment. The use of student policies to collect trajectories during distillation generally results in superior policies compared to using the teacher. This is because using the teacher policy tends to overfit to the states it visits, neglecting a significant portion of the state space that the well-performing teacher overlooks (Czarnecki et al., 2019).

We then extend the formulation by incorporating elements from Proximal Policy Distillation (PPO) (Schulman et al., 2017). Specifically, we use the PPO loss to integrate environment rewards into the objective, we enable the reuse of samples from a rollout buffer to increase sample efficiency, via importance sampling, and we introduce proximality constraints to improve stability.

Proximal Policy Distillation works as follows: a student agent is trained similarly to standard PPO, by alternating between filling up a rollout buffer of trajectories through interaction with the task environment, using the current policy π_{θ_k}, and performing policy updates. Policy updates are calculated by mini-batch gradient descent using samples from the rollout buffer for a fixed number of epochs, to solve the following (implicitly) constrained optimization problem at each iteration:

$$\begin{aligned}
\theta_{k+1} &= \arg\max_\theta \mathbb{E}_{s,a\sim\pi_{\theta_k}}\left[L_{\text{PPO}}(s,a,\theta) - \lambda\,\text{KL}(\pi_{\text{teacher}}(s)\|\pi_\theta(s))\max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon\right)\right] \\
L_{\text{PPO}}(s,a,\theta) &= \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\hat{A}(s,a), g(\epsilon,\hat{A}(s,a))\right), \qquad g(\epsilon,A) = \begin{cases}(1+\epsilon)A, & A \geq 0\\ (1-\epsilon)A, & A < 0\end{cases}
\end{aligned} \tag{2}$$

where the constraint is enforced by the hyperparameter ϵ. L_{PPO} is the PPO-clip loss, λ is a hyperparameter that balances the relative strength of the PPO and distillation losses, π_{θ_k} is the policy from the previous PPO iteration that was used to collect the rollout buffer (hence, $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is the importance sampling ratio), $\pi_{\theta_{k+1}}$ is the resulting policy at the end of the iteration, which will be used to collect the next rollout, and π_{teacher} is the teacher policy that we wish to use to guide learning (e.g., for distillation or as skills prior). Note that we clip the distillation objective in the same way as the PPO loss; however, since KL-divergence is always non-negative, we only need to clip the positive branch of the loss. The critic is trained from scratch using a regression loss, as in PPO. The full algorithm is reported in Appendix A (algorithm 1).

3 Empirical Evaluation

We compared PPD against two policy distillation methods: 'student-distill' and 'teacher-distill'. Student-distill (SD) is similar to 'on-policy distill' from Czarnecki et al. (2019), where policy distillation is treated as a supervised learning problem over trajectories collected using the student as sampling (control) policy. Teacher-distill (TD) is performed in the same way as student-distill, but the teacher policy is used to sample

environment trajectories instead of the student's. Pseudocode of the two baseline methods is included in Appendix A algorithms 2 and 3).

Evaluation was performed over a wide range of challenging reinforcement learning environments involving discrete and continuous action spaces, continuous control, and out-of-distribution generalization. Namely, we tested all distillation methods on three suites of environments:

- A subset of 11 games from the **Atari** suite (`Atlantis`, `BeamRider`, `CrazyClimber`, `DemonAttack`, `Enduro`, `Freeway`, `MsPacman`, `Pong`, `Qbert`, `Seaquest`, and `Zaxxon`).
- 4 **Mujoco** environments (`Ant`, `HalfCheetah`, `Hopper`, and `Humanoid`).
- 4 environments from the **Procgen** bechmark (`coinrun`, `jumper`, `miner`, and `ninja`).

We first trained teacher models for each environment using PPO, repeating the process over five random seeds. Full details of the training procedure, hyperparameters, and network architectures are provided in Appendix A.2.

We then trained student models using the three policy distillation methods, PPD, SD, and TD, applied to three different student network architectures. These include a *smaller* network with approximately 25% of the teacher network's parameters, an *identical* network in a self-distillation scenario (Furlanello et al., 2018), and a *larger* network containing about 3-7 times more parameters than the corresponding teacher. We repeated all the experiments for each random seed. Exact network sizes and architectures are included in Appendix A.3.

Sample efficiency. We first examined the training curves during distillation to evaluate the sample efficiency of the different methods. Figure 1 shows the case of distillation to *larger* student networks. Similar figures for the other student network sizes are included in Appendix B.2.

We found that PPD typically learns faster and reaches better performance levels, except in the Mujoco environments, where student-distill showed a faster learning pace, although final performance was comparable in both methods.

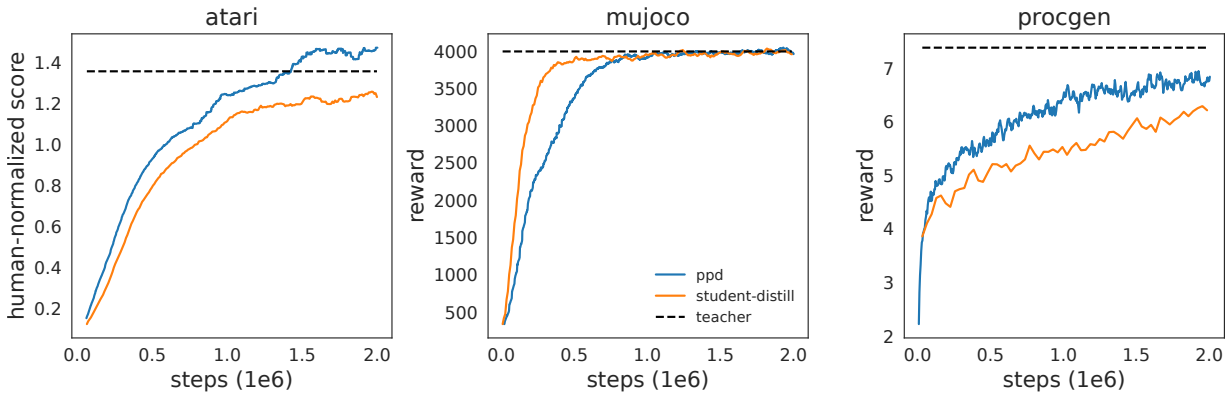

Figure 1: Training curves of PPD and student-distill in the setting of distillation onto a larger student network, showing the average episodic returns collected *during* distillation, averaged over five random seeds and all environments in each of three suites 'atari', 'mujoco', and 'procgen'. The final mean training performance of the teacher models is shown as a dashed line. Results for procgen are calculated over training levels. We observe that PPD students achieve higher rewards than student-distill ones, and are generally more sample efficient.

Distillation performance. We then assessed the performance of student models trained with the three distillation methods. Evaluation was executed in a test setting (which in the case of 'procgen' corresponds to using a different set of levels), where the distilled students were used to interact with the environment. Actions were chosen deterministically, instead of the stochastic action selection used during training, except for 'procgen', where we observed that deterministic policies were prone to getting stuck, leading to lower performance for all agents. Results for Atari environments are reported as human-normalized scores, using base values from Badia et al. (2020).

Table 1 shows the results with respect to the corresponding teacher's performance. We found that PPD outperforms both student-distill and teacher-distill across all environments. Additionally, we observed that, like in Figure 1, the models perform close to each other on Mujoco environments.

Furthermore, we noted that distilling into larger student networks generally results in better student performance after distillation, often surpassing that of the teacher models, especially for PPD.

Table 1: Performance of student models of three sizes (smaller, same, larger), trained using the three distillation methods (PPD, student-distill, and teacher-distill). Evaluation is performed in a test setting (which in the case of 'procgen' corresponds to using a different set of levels). Results are calculated as fraction of teacher score for each environment and random seed, and then averaged by geometric mean. We omit the 'x' symbols in all but the last row to reduce clutter.

env	teacher	smaller			same-size			larger		
	score	td	sd	ppd	td	sd	ppd	td	sd	ppd
atari	2.19	0.89	1.05	**1.06**	1.11	1.16	**1.19**	1.15	1.14	**1.25**
mujoco	4322.42	0.94	**0.97**	**0.97**	0.96	1.0	**1.01**	0.94	**1.0**	**1.0**
procgen	5.16	0.59	0.66	**0.75**	0.75	0.82	**0.98**	0.83	0.93	**1.02**
all		0.83x	0.94x	**0.97x**	0.99x	1.05x	**1.11x**	1.03x	1.07x	**1.14x**

Distillation from imperfect teachers. Since the goal of policy distillation is to reproduce the behavior of a teacher policy into a student network, the performance of the student is generally limited by the performance of the teacher. However, we found that PPD often achieves performance superior to its teacher. We investigated this aspect of the method further with a simple experiment where we deliberately corrupted the performance of the teacher models by perturbing their parameters with Gaussian noise

$$\theta_i \leftarrow \theta_i + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

with $\sigma = 0.05$. We found that it can be challenging to obtain teachers with degraded performance that still manage to perform effectively in their target environments. For this reason, we focused on a subset of the environments (four Atari environments {`BeamRider`, `CrazyClimber`, `MsPacman`, `Qbert`} and four Procgen environments {`miner`, `jumper`, `coinrun`, `ninja`}) for which reasonable imperfect teachers could be generated.

We then performed policy distillation using the three methods, and compared their performance relative to that of the original non-corrupted teachers. Evaluation was limited to students with larger networks. For Procgen, we evaluated the trained agents using test-levels (results for training levels are available in the Appendix, Table 8). Each experiment was repeated five times with different random seeds.

As shown in Table 2, PPD was found to consistently achieve better performance than both student-distill and teacher-distill, which instead fared only marginally better than their respective corrupted teachers.

Effect of hyperparameter λ. We finally explored the impact of the hyperparameter λ, which balances the PPO and distillation losses, on distillation performance. The analysis was limited to the same subset of Atari environments as in the study of imperfect teachers, and focused on distillation onto the larger network architecture. The value of the hyperparameter was varied over four values $\lambda \in \{0.5, 1, 2, 5\}$.

Table 2: Performance of student models, trained using the three distillation methods (PPD, student-distill, and teacher-distill) using 'imperfect teachers' that are artificially corrupted to decrease in performance. Results are calculated as a fraction of the original teacher score for each environment and random seed, and then averaged by geometric mean. Distillation is performed on four Atari and four Procgen environments, and onto larger student networks.

env	corrupted teacher score	larger td	larger sd	larger ppd
atari	0.41x	0.45x	0.46x	**0.59x**
procgen	0.67x	0.57x	0.63x	**0.71x**

Figure 2 shows that increasing the value of λ speeds up distillation, due to the higher weight given to the distillation loss. Nevertheless, the impact of this hyperparameter is relatively small.

Test-time evaluation of the PPD students with different λ showed that lower values allow students to learn better policies than the corresponding teacher, as expected since high λ prioritizes the distillation loss against the PPO loss, leaving less margin for the student to deviate from the teacher's actions. The relative final test performance of students compared to the teacher are shown in parentheses in the legend of Figure 2.

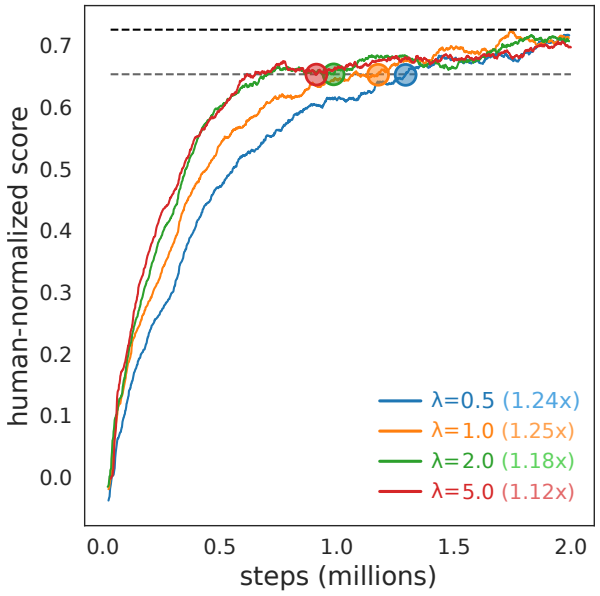

Figure 2: Performance of student models trained using PPD with different values of $\lambda \in \{0.5, 1, 2, 5\}$. Results are averaged over 5 random seeds. Distillation is performed on four Atari environments and onto larger student networks. Dashed lines indicate 100% (black) and 90% (gray) of the average performance of the teacher models. A circle on each curve marks the last training step where the student's performance is below 90% of the teacher's. All curves were collected during training, as in Figure 1. Values in parentheses indicate the relative final test performance of the students with respect to the teacher.

4 Related Work

Most current policy distillation methods overlook the rewards obtained by the student during the distillation process. Instead, they frame policy distillation solely as a supervised learning task. In this approach, a

dataset of state observations and teacher outputs is collected using either the student or teacher policies. The data is then stored in a distillation dataset, from which training mini-batches are sampled.

While previous work (Czarnecki et al., 2019) has presented a general framework for policy distillation that allows the inclusion of environment rewards, the authors show that many types of current PD methods may have difficulty converging in that case.

PD as a form of supervised learning can be easily adapted to transfer functions between different types of models, such as value-based teachers (e.g., trained with DQN) and policy-based students. This proves beneficial in scenarios like reincarnating reinforcement learning (Agarwal et al., 2022), allowing continued training with alternate RL methods after distillation. For instance, Green et al. (2019) showed that teachers trained using DQN can be distilled into "PPO students," allowing for subsequent fine-tuning with Proximal Policy Optimization (PPO) (Green et al., 2019). However, unlike in our work, PPO was not used during distillation, and the trajectories for the distillation loss were collected in a teacher-driven way, with the teacher interacting with the environment instead of the student. This method closely resembles the 'teacher_distill' strategy in our baseline, which we demonstrated to be less effective than PPD.

An alternative way to perform policy distillation is through imitation learning (IL), where demonstrations from teacher models provide specific actions as hard targets, rather than using the typical soft targets (e.g., output probabilities). For example, Zhuang et al. (2023) use DAgger (Dataset Aggregation) (Ross et al., 2011) to distill five behaviors for a robot quadruped (climb, leap, crawl, tilt, run) into a single control policy, by selecting the appropriate teacher for each part of a training obstacle course. The advantage of DAgger over naive Behavior Cloning stems from its ability to query teacher actions for new states encountered by the student, which is always possible in policy distillation, rather than relying solely on teacher-collected demonstrations.

Finally, closest to our work is the approach by Schmitt et al. (2018), which combines a policy gradient loss with a distillation loss and student-driven distillation. The authors also propose a method to dynamically update the relative weighting λ of the policy gradient loss and distillation losses for multiple teachers, using Population Based Training (PBT) (Jaderberg et al., 2017). The work then applies distillation in a manner similar to reincarnating RL (Agarwal et al., 2022), where a smaller teacher is distilled into a larger student. However, distillation and learning are combined, instead of being performed sequentially. The main difference with Proximal Policy Distillation is that the PPD extends PPO, while the work by Schmitt et al. (2018) is based on IMPALA (Espeholt et al., 2018).

5 Discussion and Conclusions

We introduced Proximal Policy Distillation (PPD), a new method for policy distillation that combines reinforcement learning by PPO with a suitably constrained distillation loss.

Through a thorough evaluation over a wide range of environments, we showed that PPD achieves higher sample efficiency and better final performance after distillation, compared to two popular distillation baselines, student-distill and teacher-distill. We suggest that this is due to the reuse of samples from the rollout buffer, together with the capacity to exploit environment rewards during distillation. We also confirmed that using the student policy to collect trajectories during distillation effectively reduces overfitting to teacher demonstrations (Czarnecki et al., 2019).

In theory, the sample efficiency of teacher-distill can be improved by first collecting a dataset of trajectories using the teacher and then applying offline supervised learning. Instead, our evaluation of teacher-distill was performed online, by immediately using and then discarding teacher trajectories. We note however that reusing demonstrations collected using the teacher policy may exacerbate overfitting to the teacher. This is shown in the Appendix (Figure 8): while teacher-distill can quickly match the teacher performance during training, that is on trajectories generated by the teacher itself, its performance drops significantly when the distilled student is used to interact with the environment (as shown in Table 1).

We also found that using larger student neural networks during distillation correlates with improved performance, even surpassing that of the original teacher. Our results thus validate the advantages of 'reincarnating'

reinforcement learning agents that are first trained on simpler networks to reduce training time into larger and more capable networks (Agarwal et al., 2022; Schmitt et al., 2018). However, if the student is to continue learning using any actor-critic RL method, it becomes necessary to also distill the critic of the teacher together with its policy network (actor). In the case of PPD, this is not necessary, since the student agent can learn its value function from scratch while interacting with the environment during distillation.

We further investigated the robustness of PPD compared to the two baselines in a special case where teachers were corrupted to perform suboptimally. This setting draws from the challenging, unresolved issue of learning from imperfect demonstrations (Gao et al., 2018), which stems from the fact that successful replication of a teacher's behavior would include both its good and bad actions. In our tests, we found that PPD students managed to regain a large fraction, although not all, of the original uncorrupted performance in the environments tested, outperforming both student-distill and teacher-distill. Most notably, PPD students consistently obtained higher rewards than the imperfect teachers they were learning from. Further analysis of the performance of the distilled models on training versus test levels on Procgen further suggests that both student-distill and teacher-distill remain limited by the performance of the (imperfect) teachers.

Finally, we released the 'sb3-distill' [2] library to aid reproducibility of the work and facilitate the application of policy distillation methods, particularly Proximal Policy Distillation. The library implements the three methods PPD, student-distill, and teacher-distill, within the stable-baselines3 framework (Raffin et al., 2019) by introducing a new 'PolicyDistillationAlgorithm' interface to extend 'OnPolicyAlgorithm' classes.

Future work should focus on extending PPD to use teachers as skills priors, especially in the case of multitask policy distillation. To make that practical, and to further help PPD students to achieve higher performance than their teacher, it will be beneficial to implement a dynamic balancing in the trade-off between the PPO and individual teacher distillation losses, as for example done by Schmitt et al. (2018), who used Population Based Training to update the λ_k parameters during training.

References

Josh Abramson, Arun Ahuja, Iain Barr, Arthur Brussee, Federico Carnevale, Mary Cassin, Rachita Chhaparia, Stephen Clark, Bogdan Damoc, Andrew Dudzik, et al. Imitating interactive intelligence. *arXiv preprint arXiv:2012.05672*, 2020.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Beyond tabula rasa: Reincarnating reinforcement learning. *arXiv preprint arXiv:2206.01626*, 2022.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Wojciech M Czarnecki, Razvan Pascanu, Simon Osindero, Siddhant Jayakumar, Grzegorz Swirszcz, and Max Jaderberg. Distilling policy distillation. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1331–1340. PMLR, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

[2] `<AnonymizedGitHubRepository>`

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International conference on machine learning*, pp. 1607–1616, 2018.

Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. In *6th International Conference on Learning Representations, ICLR, Workshop Track Proceedings*, 2018.

Sam Green, Craig M Vineyard, and Cetin Kaya Koç. Distillation strategies for proximal policy optimization. *arXiv preprint arXiv:1901.08128*, 2019.

Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H Huang, Dhruva Tirumala, Jan Humplik, Markus Wulfmeier, Saran Tunyasuvunakool, Noah Y Siegel, Roland Hafner, et al. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9(89):eadi8022, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

I-Chun Arthur Liu, Shagun Uppal, Gaurav S Sukhatme, Joseph J Lim, Peter Englert, and Youngwoon Lee. Distilling motion planner augmented policies into visual control policies for robot manipulation. In *Conference on Robot Learning*, pp. 641–650. PMLR, 2022a.

Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Ali Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *Science Robotics*, 7(69):eabo0235, 2022b.

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Antonin Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3, 2019.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *Transactions on Machine Learning Research*, 2022.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *4th International Conference on Learning Representations, ICLR*, 2016.

Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Dhruva Tirumala, Alexandre Galashov, Hyeonwoo Noh, Leonard Hasenclever, Razvan Pascanu, Jonathan Schwarz, Guillaume Desjardins, Wojciech Marian Czarnecki, Arun Ahuja, Yee Whye Teh, et al. Behavior priors for efficient reinforcement learning. *The Journal of Machine Learning Research*, 23(1):9989–10056, 2022.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL `https://zenodo.org/record/8127025`.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher G Atkeson, Sören Schwertfeger, Chelsea Finn, and Hang Zhao. Robot parkour learning. In *Conference on Robot Learning*, 2023.

A Supplementary Methods

A.1 Algorithm listings and baseline methods

We include full algorithm listings for the three distillation methods compared in this work. PPD is shown in Algorithm 1, student-distill in Algorithm 2, and teacher-distill in Algorithm 3.

Algorithm 1 Proximal Policy Distillation

Input: teacher policy π_{teacher}
Initialize student policy π_θ and value function V_ϕ.
for $k = 1, 2, \ldots$ **do**
  Collect trajectories by running the student policy π_θ in the environment to fill a rollout buffer $\mathcal{D}_k = \{(s_i, a_i, r_i, s_i')\}$ with n environment steps.
  Compute returns $\hat{R}_i$ and then advantage estimates, $\hat{A}_i$.
  for epoch $= 1, 2, n_{\text{epochs}} \ldots$ **do**
    Shuffle rollout buffer.
    for m $= 1, 2, n_{\text{minibatches}} \ldots$ **do**
      Extract the m-th mini-batch $\mathcal{D}_k^m$ from $\mathcal{D}_k$.
      Update the policy and value functions by maximizing the combined objectives via gradient descent over mini-batches

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k^m|} \sum_{(s,a)\in\mathcal{D}_k^m} L_{\text{PPO}}(s,a,\theta) + \alpha L_{\text{entropy}}(s,a,\theta) - \lambda L_{\text{PPD}}(s,a,\theta)$$

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k^m|} \sum_{(s,a,\hat{R})\in\mathcal{D}_k^m} \left(V_\phi(s) - \hat{R}\right)^2$$

      where

$$L_{\text{PPO}}(s,a,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\hat{A}(s,a), g(\epsilon, \hat{A}(s,a))\right), \qquad g(\epsilon, A) = \begin{cases}(1+\epsilon)A, & A \geq 0\\ (1-\epsilon)A, & A < 0\end{cases}$$

$$L_{\text{PPD}}(s,a,\theta) = \text{KL}(\pi_{\text{teacher}}(s) \| \pi_\theta(s)) \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon\right)$$

    end for
  end for
end for

Algorithm 2 Student-Distill

1: **Input:** teacher policy π_{teacher} and value function V_{teacher}
2: Initialize student policy π_θ and value function V_ϕ.
3: **for** $k = 1, 2, \ldots$ **do**
4: Collect trajectories by running the **student** policy π_θ in the environment to fill a rollout buffer $\mathcal{D}_k = \{(s_i, a_i, r_i, s'_i)\}$ with n environment steps.
5: Perform a step of gradient descent to obtain new parameters θ_{k+1} and $\phi k + 1$

$$\theta_{k+1} = \theta_k - \eta \frac{1}{|\mathcal{D}_k|} \sum_{(s,a)\in\mathcal{D}_k} \text{KL}(\pi_{\text{teacher}} \| \pi_\theta) - \alpha L_{\text{entropy}}(\theta, s, a)$$

$$\phi_{k+1} = \phi_k - \eta \frac{1}{|\mathcal{D}_k|} \sum_{(s,a)\in\mathcal{D}_k^m} \left(V_\phi(s) - V_{\text{teacher}}(s)\right)^2$$

6: **end for**

Algorithm 3 Teacher-Distill

1: **Input:** teacher policy π_{teacher} and value function V_{teacher}
2: Initialize student policy π_θ and value function V_ϕ.
3: **for** $k = 1, 2, \ldots$ **do**
4: Collect trajectories by running the **teacher** policy π_{teacher} in the environment to fill a rollout buffer $\mathcal{D}_k = \{(s_i, a_i, r_i, s'_i)\}$ with n environment steps.
5: Perform a step of gradient descent to obtain new parameters θ_{k+1} and $\phi k + 1$

$$\theta_{k+1} = \theta_k - \eta \frac{1}{|\mathcal{D}_k|} \sum_{(s,a)\in\mathcal{D}_k} \text{KL}(\pi_{\text{teacher}} \| \pi_\theta) - \alpha L_{\text{entropy}}(\theta, s, a)$$

$$\phi_{k+1} = \phi_k - \eta \frac{1}{|\mathcal{D}_k|} \sum_{(s,a)\in\mathcal{D}_k^m} \left(V_\phi(s) - V_{\text{teacher}}(s)\right)^2$$

6: **end for**

A.2 Training of the teacher models

We train teacher models for each environment over five random seeds $\{100, 200, 300, 400, 500\}$. We used the following environments from Gymnasium (Towers et al., 2023) and Procgen (Cobbe et al., 2019):

- **Atari** (11 environments): `AtlantisNoFrameskip-v4`, `SeaquestNoFrameskip-v4`, `BeamRiderNoFrameskip-v4`, `EnduroNoFrameskip-v4`, `FreewayNoFrameskip-v4`, `MsPacmanNoFrameskip-v4`, `PongNoFrameskip-v4`, `QbertNoFrameskip-v4`, `ZaxxonNoFrameskip-v4`, `DemonAttackNoFrameskip-v4`, `CrazyClimberNoFrameskip-v4`.
- **Mujoco** (5 environments): `Ant-v4`, `HalfCheetah-v4`, `Hopper-v4`, `Swimmer-v4`, `Humanoid-v4`.
- **Procgen** (4 environments): `miner`, `jumper`, `ninja`, `coinrun`.

Training on Atari and Procgen environments was performed using an IMPALA-CNN architecture with $\{16, 32, 32\}$ convolutional filters, a 256-unit fully connected layer, and ReLU activations (Espeholt et al., 2018). Separate neural networks were used for policy and value networks. Training on Atari was performed using PPO for a total of 10 million environment steps, while Procgen environments were trained for 30 million steps. Atari environments were wrapped to terminate on the first life loss, without frame skipping, and using frame stacking with the 4 most recent environment frames. For Procgen, we used 200 'easy' mode unique levels during training.

Mujoco environments used a MultiLayer Perceptron (MLP) backbone with two hidden layers of 256 units and ReLU activation. Separate networks were used to represent the policy and value functions. Training was performed for 10 million environment steps.

Rewards were always normalized using a running average, and observations were normalized when training Mujoco environments. Final normalization statistics for Mujoco teachers were then fixed and reused during distillation, to guarantee that both student and teacher received the same inputs. This was not strictly required, and it was only chosen for simplicity.

The PPO hyperparameters are shown in Table 3. Simple hyperparameter tuning was initially performed, starting from parameter values suggested from the stable-baselines3 model zoo (Raffin, 2020).

Table 3: PPO hyperparameters used for training the teacher models.

Hyperparameter	**Value**
n_envs	18
n_steps	256 (Atari, swimmer, hopper), 512 (Procgen, Mujoco)
batch_size	512
γ	0.995
λ	0.9
lr	3e-4
n_epochs	4
ent_coef	0.01 (Atari, Procgen, swimmer, hopper), 0.0 (Mujoco)

A.3 Policy distillation experiments

We performed policy distillation of all teachers, across all random seeds, for all PD methods (PPD, student-distill, teacher-distill) onto three different student network sizes: **smaller**, **same** (same architecture as the student, i.e., self-distillation), and **larger**.

The smaller network for Atari and Procgen was a Nature-CNN (Mnih et al., 2015) with convolutional filters $\{32, 32, 32\}$ $(8s4, 4s2, 3s1)$ and a fully connected layer of 128 units, resulting in $\sim 0.25x$ the number of parameters of the teacher. The larger networks for Atari and Procgen were IMPALA-CNNs with $\{32, 64, 64\}$ convolutional filters and 1024 units in the fully connected layer ($\sim 7.5x$ the number of parameters of the base teacher network).

The smaller network for Mujoco was an MLP with two hidden layers of sizes 128 and 64 ($\sim 0.25x$ the number of parameters of the teacher), while the larger network had two hidden layers of size 512 ($\sim 3x$ the number of parameters of the base network)

Table 4 reports the number of parameters for all student networks.

The hyperparameters of PPD related to PPO were the same as for the teacher training, except we used $\gamma = 0.999$ during distillation ($\gamma = 0.995$ for swimmer and hopper), end_coef=0, and shorter rollout trajectories (n_steps=64 for PPD, and n_steps=5 for student-distill and teacher-distill). Distillation was performed for 2 million environment steps for all methods.

Evaluation of distilled students was performed with deterministic actions for Atari and Mujoco, and stochastic actions in Procgen. Evaluation on Procgen environments was always on unseen test levels, rather than the training levels used for training the teacher and for distillation. Results from Atari games are reported with human-normalized scores, using base values from Badia et al. (2020). Rewards were averaged over 50 randomized episodes for each trained or distilled agent and random seed, for Atari and Mujoco, and 200 randomized episodes for Procgen.

Table 4: Number of parameters of the three networks used (smaller, same, larger) for each of the three domains (Atari, Mujoco, Procgen). Values in parentheses denote the relative size of each network with respect to the original teacher.

domain	smaller	same-size	larger
atari	279.3k (0.25x)	1.1M (1x)	8.3M (7.54x)
mujoco	22.7k (0.24x)	95.8k (1x)	322.7k (3.37x)
procgen	142.5k (0.23x)	626k (1x)	4.6M (7.35x)

B Supplementary Results

B.1 Teacher training

Training curves for all teachers, averaged over five random seeds, are shown as follows: Figure 3 (Atari), Figure 4 (Mujoco), and Figure 5 (Procgen).

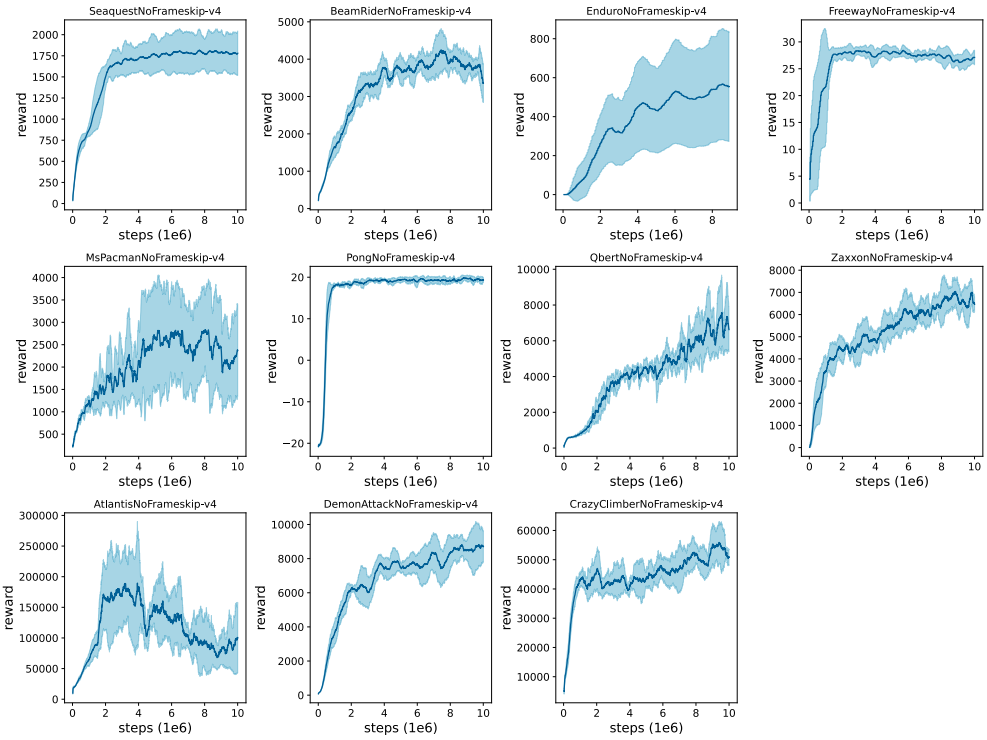

Figure 3: Training curves for all Atari teachers, averaged over 5 random seeds. Shaded areas denote standard deviation.

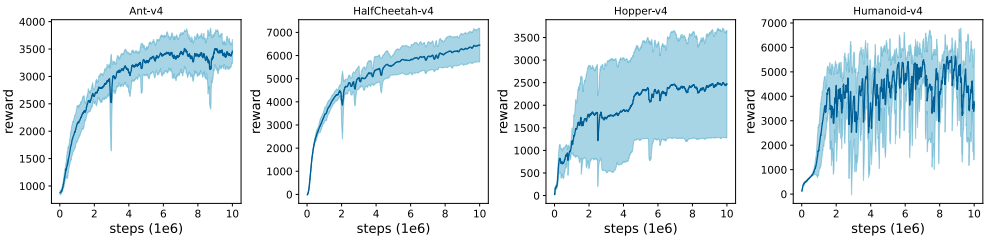

Figure 4: Training curves for all Mujoco teachers, averaged over 5 random seeds. Shaded areas denote standard deviation.

B.2 Policy distillation

We include figures with the training performance during distillation to different sizes of student networks, extending Figure 1 from the main text. Figure 6 shows the training curves while distilling into the **smaller**

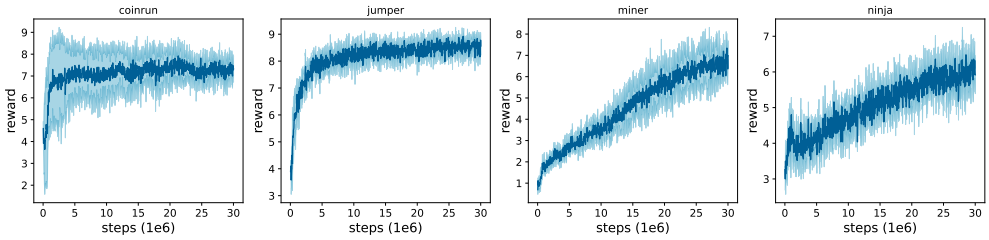

Figure 5: Training curves for all Procgen teachers, averaged over 5 random seeds. Shaded areas denote standard deviation.

student network, Figure 7 for the **same**-sized network, and Figure 8 for the **larger** network. The latter is thus like Fig. 1 from the main text, but with the inclusion of the training curve for teacher-distill.

Note that while teacher-distill approaches the performance of the teacher model quickly, the result is misleading, since performance in teacher-distill is measured over teacher trajectories. Indeed, the final performance of teacher-distill evaluated using the distilled student's trajectories (see Table 1) is worse than both other models, suggesting significant overfitting.

We then report individual scores for all environments and students in Table 5.

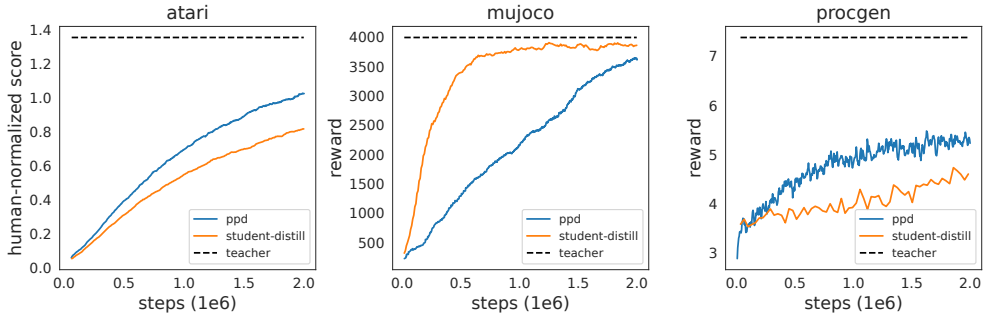

Figure 6: Same as Figure 1, but with distillation to a **smaller** student.

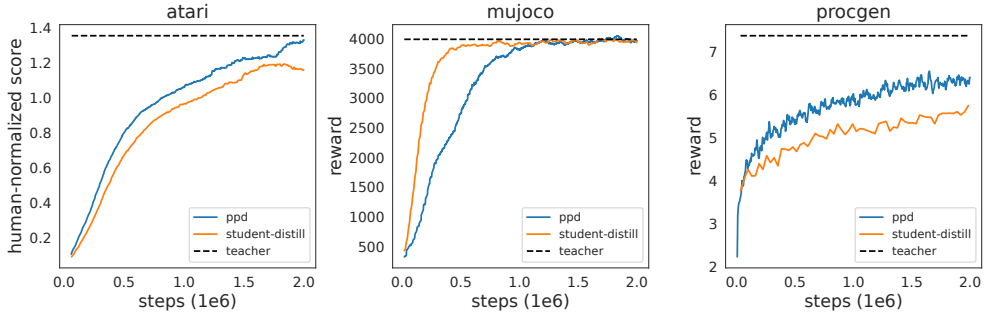

Figure 7: Same as Figure 1, but with distillation to a **same**-sized student.

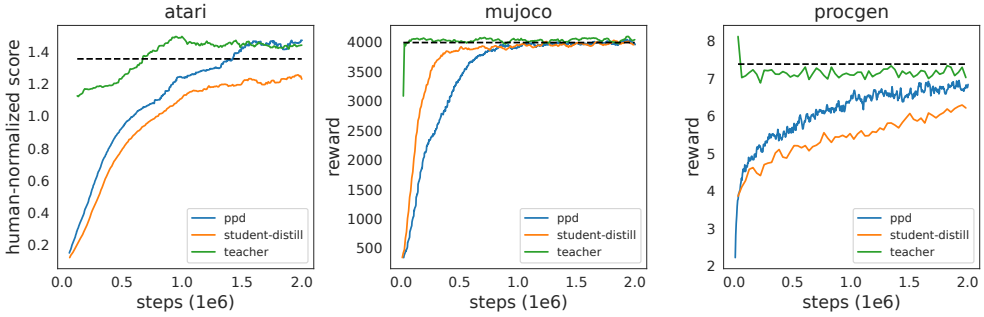

Figure 8: Same as Figure 1, showing distillation to a larger student, but also including teacher-distill. Note that while teacher-distill approaches the performance of the teacher model quickly, the result is misleading, since performance in teacher-distill is measured over teacher trajectories. Indeed, the final performance of teacher-distill evaluated using the distilled student's trajectories (see Table 1) is worse than both other models, suggesting significant overfitting.

Table 5: The table extends Table 1 from the main text by reporting results for each environment separately, averaged over 5 random seeds. Atari games are reported as human-normalized scores.

env	teacher	smaller			same-size			larger		
	score	td	sd	ppd	td	sd	ppd	td	sd	ppd
atari										
atlantis	12.85	21.1	16.27	23.43	18.07	14.07	20.42	18.07	19.64	28.03
seaquest	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
beamrider	0.2	0.12	0.16	0.2	0.2	0.23	0.26	0.22	0.22	0.25
enduro	0.69	0.57	0.61	0.58	0.79	0.8	0.88	0.85	0.85	0.87
freeway	0.99	0.73	0.69	0.98	0.95	0.97	1.0	0.98	0.96	0.99
mspacman	0.35	0.33	0.34	0.31	0.34	0.38	0.34	0.39	0.36	0.38
pong	1.17	0.98	1.13	1.14	1.16	1.17	1.17	1.17	1.17	1.17
qbert	0.58	0.54	0.58	0.39	0.66	0.7	0.73	0.67	0.76	0.76
zaxxon	0.69	0.51	0.59	0.58	0.57	0.58	0.68	0.59	0.6	0.64
demonattack	4.45	5.62	11.08	11.61	9.64	10.25	8.71	9.37	7.59	9.55
crazyclimber	2.08	1.53	2.83	2.23	2.19	2.92	2.33	2.36	2.35	2.62
mujoco										
ant	3574	3529	3505	3279	3466	3543	3549	3530	3630	3490
half-cheetah	6664	6588	6530	6549	6595	6621	6638	6567	6620	6631
hopper	2510	2451	2521	2501	2489	2496	2534	2454	2540	2527
humanoid	4540	3747	4226	4443	4070	4591	4804	3737	4416	4618
procgen-train										
coinrun	7.29	5.39	5.55	6.76	5.94	6.12	6.99	6.63	6.91	7.25
jumper	8.39	6.98	7.07	7.89	8.22	8.29	8.41	8.29	8.43	8.43
miner	6.64	1.24	1.47	1.76	2.34	2.8	4.39	3.08	4.11	5.61
ninja	6.38	3.44	4.21	4.61	4.55	4.73	5.18	5.23	5.12	5.65
procgen-test										
coinrun	6.28	5.2	5.08	6.17	5.06	5.55	6.36	5.7	5.74	6.45
jumper	5.34	3.89	4.53	5.4	5.57	5.47	5.75	5.6	5.9	5.73
miner	4.57	1.22	1.46	1.49	1.87	2.51	3.66	2.37	3.34	4.43
ninja	4.44	3.34	3.83	4.33	4.1	4.13	4.68	4.29	4.58	4.57

B.3 Distillation: imperfect teachers

We report the results of distillation from imperfect teachers for each environment in Table 6. The results from this table are aggregated and shown in Table 2 from the main text.

Table 6: Full results for each environment, extending Table 2 from the main text. We show the performance of student models, trained using the three distillation methods (PPD, student-distill, and teacher-distill) from 'imperfect teachers' that are artificially corrupted to decrease in performance. Results are calculated as a fraction of the original teacher score for each environment and random seed, and then averaged by geometric mean. Distillation is performed on four Atari and four Procgen environments, and onto larger student networks. Evaluation for Procgen is performed on test levels. We omit the 'x' symbols to reduce clutter.

env	original teacher score	corrupted teacher score	td	larger sd	ppd
BeamRider	3754.15	0.29x	0.33	0.31	**0.39**
CrazyClimber	62912.0	0.39x	0.37	0.44	**0.54**
MsPacman	2640.16	0.49x	0.52	0.55	**0.71**
Qbert	7904.5	0.54x	0.66	0.6	**0.81**
atari		0.41x	0.45x	0.46x	**0.59x**
miner	6.64	**0.69x**	0.36	0.48	**0.69**
jumper	8.39	0.56x	0.6	**0.63**	**0.63**
coinrun	7.29	0.8x	0.71	0.77	**0.83**
ninja	6.38	0.64x	0.69	0.65	**0.72**
procgen		0.67x	0.57x	0.63x	**0.71x**

B.4 Procgen: train vs test levels

We look into the performance of teachers and students on Procgen environments, where evaluation is performed on the original **training** levels instead of the unseen test levels used for evaluation in the main text.

Table 7 extends the main results Table 1, while Table 8 shows results for the case of imperfect teachers.

Table 7: Performance of students models of three sizes (smaller, same, larger), trained using the three distillation methods (PPD, student-distill, and teacher-distill). Evaluation is performed separately on the **train** and test levels of Procgen, contrary to Table 1 that only shows performance evaluated on unseen test levels. Results are calculated as a fraction of teacher score for each environment and random seed, and then averaged by geometric mean.

env	teacher	smaller			same-size			larger		
	score	td	sd	ppd	td	sd	ppd	td	sd	ppd
procgen (train)	7.17	0.5x	0.55x	**0.64x**	0.67x	0.71x	**0.85x**	0.76x	0.83x	**0.93x**
procgen (test)	5.16	0.59x	0.66x	**0.75x**	0.75x	0.82x	**0.98x**	0.83x	0.93x	**1.02x**

Table 8: Performance of student models, trained using the three distillation methods (PPD, student-distill, and teacher-distill) from 'imperfect teachers' that are artificially corrupted to decrease in performance. Results are calculated as a fraction of the original teacher score for each environment and random seed, and then averaged by geometric mean. Distillation is performed on four Atari and four Procgen environments, and onto larger student networks. We omit the 'x' symbols to reduce clutter. Evaluation is performed on **training levels**.

env	original teacher score	corrupted teacher score	larger td	larger sd	larger ppd
miner	6.64	0.82x	0.41	0.56	**0.82**
jumper	8.39	0.89x	0.87	0.88	**0.91**
coinrun	7.29	0.88x	0.86	0.88	**0.92**
ninja	6.38	0.78x	0.78	0.75	**0.83**
procgen		0.84x	0.7x	0.76x	**0.87x**