Distilling Reinforcement Learning into Single-Batch Datasets

Connor Wilhelm[a,*] **and Dan Ventura**[a]

[a]Computer Science Department
Brigham Young University
ORCID (Connor Wilhelm): https://orcid.org/0009-0002-3028-7711, ORCID (Dan Ventura): https://orcid.org/0000-0002-3111-2238

Abstract. Dataset distillation compresses a large dataset into a small synthetic dataset such that learning on the synthetic dataset approximates learning on the original. Training on the distilled dataset can be performed in as little as one step of gradient descent. We demonstrate that distillation is generalizable to different tasks by distilling reinforcement learning environments into one-batch supervised learning datasets. This demonstrates not only distillation's ability to compress a reinforcement learning task but also its ability to transform one learning modality (reinforcement learning) into another (supervised learning). We present a novel extension of proximal policy optimization for meta-learning and use it to distill a multi-dimensional extension of the classic cart-pole problem, all MuJoCo environments, and several Atari games. We demonstrate distillation's ability to compress complex RL environments into one-step supervised learning, explore distillation's generalizability across agent architectures, and demonstrate distilling an environment into the smallest possible synthetic dataset.

1 Introduction

As deep learning increases in popularity and models are trained on larger tasks, training imposes greater costs to the deep learning community and the planet. High training costs provide a high barrier to entry to many, and can make hyperparameter searches such as neural architecture search prohibitively expensive. Training's high energy usage creates high economic and ecological costs. As deep learning advances, methods to reduce training costs become more necessary. Reinforcement learning in particular suffers from high training costs imposed by repeated simulations of the environment, inefficient experience gathering, reliance on auxiliary networks, and many potential failure states. We propose an extension of dataset distillation to reduce reinforcement learning costs to near-negligible levels: a single step of gradient descent, while avoiding expensive environment simulations.

Dataset distillation[1] is a technique in which a large dataset is compressed into a synthetic dataset, learnable in as few as a single step of gradient descent [19]. The performance of a model when trained on the synthetic dataset should approximate the performance of that model when trained on the original dataset. By compressing the original dataset, distillation greatly reduces the cost of training models; in this work, we demonstrate one-step learning made possible with distillation. The compression provided by distillation can be used for low-resource training, interpretability, and data anonymization [5]. The reduction of training costs resulting from this compression can accelerate searches such as hyperparameter searches, neural architecture searches, and high-performance initialization searches, and can produce inexpensive ensembles.

We propose a generalization of dataset distillation, which we call *task distillation*. Using techniques similar to dataset distillation, we posit that any learning task can be compressed into a synthetic task. Task distillation allows for transmodal distillations, reducing a more complex task, such as a reinforcement learning environment, to a simpler task, such as a supervised classification dataset.

In order to advance task distillation beyond supervised-to-supervised distillation, we explore distilling reinforcement learning environments into supervised learning datasets, which we call *RL-to-SL distillation*. We choose to produce synthetic supervised datasets because supervised learning is the bread-and-butter of machine learning, with decades of research going into improving SL. We consider transforming an arbitrary learning task into supervised learning to be an important step to simplify machine learning. We select RL environments to distill because reinforcement learning has the added complexity of requiring exploration of the environment. Exploration can cause a host of difficulties in learning, such as poor policy spaces that an agent cannot escape and a general increase in training complexity. Learning a successful policy often requires a great deal of exploration, which may involve learning on many repetitive observations that hold little useful information. By producing a compressed synthetic dataset, agents can learn an RL task quickly without exploration costs, as exploration has already occurred during distillation.

The paper begins by exploring distillation on the cart-pole problem [3] as a pedagogical exercise; then demonstrates distillation of more difficult RL environments [MuJoCo [18], *Centipede* [2], *Ms. Pac-Man* [6], *Pong* [1], and *Space Invaders* [17]]; and makes the following contributions:

1. Proposes a new formulation of RL task distillation using proximal policy optimization.
2. Proposes an N-dimensional extension of the cart-pole environment to allow for scaling the difficulty of cart-pole.
3. Demonstrates k-shot learning on single-batch datasets distilled from ND cart-pole; demonstrating generalization to unseen ini-

[*] Corresponding Author. Email: wilhelmconnorr@gmail.com.
[1] In this work, "distillation" refers only to dataset distillation and its generalization, task distillation; not to knowledge/policy distillation.

tialization distributions and architectures.

4. Empirically validates the theoretical work of Sucholutsky and Schonlau [15] by demonstrating the minimum distillation sizes of environments with different numbers of action classes.
5. Provides a method that can be used to scale the difficulty of distillation for complex tasks.
6. Demonstrates distillation of Atari and MuJoCo environments.

We provide additional information, including experiment details, ND cart-pole details, and additional figures and results in the appendices of the arXiv preprint of this paper [20]. Experiment code and distilled datasets are available on GitHub and Zenodo [21].

2 Background

2.1 Dataset Distillation

Dataset distillation was originally proposed by Wang et al. [19]. This method utilizes meta-gradients to train a synthetic dataset, based on how well a predictive model trained on the synthetic data performs on the true dataset. This process is split into two nested learning loops: in inner learning, a newly-initialized predictive model is trained on the synthetic dataset. In outer learning, the trained predictive model is tested against the true dataset, and the loss is backpropagated through the inner-learning process to the synthetic dataset. The synthetic dataset is updated with gradient descent and is used to train a new predictive model in the next iteration.

The goal of dataset distillation is to produce a synthetic dataset $\{X_d, Y_d\}_\theta$ such that predictive models sampled from a given distribution $\lambda_\phi \in \Lambda$ can train on the synthetic dataset to reach high performance on a targeted training task T_0. The original formulation defines a fixed synthetic label vector Y_d; however, in this work we use the soft labels of Sucholutsky and Schonlau [16], in which Y_d is a parameterized vector of real numbers trained along with the synthetic data instances. The sampled learner is trained on the synthetic dataset using an appropriate loss measure; we utilize mean-squared error between prediction logits and soft labels. This inner optimization must be differentiable, as outer learning requires backpropagating through the inner optimization. The trained model is tested against a sample of T_0 with an appropriate loss metric (generally one that would be used to train on T_0 directly) to produce the outer meta-loss.

It is expected that the distillation's performance is on average less than or equal to the performance of the average model trained directly on the task. This is an expected trade-off for high compression and low training time, as the compressed dataset cannot contain all the information found in the full task. However, closely approaching this performance is vital for distillation's use cases.

We visualize a task-agnostic version of this formulation of dataset distillation in Figure 1.

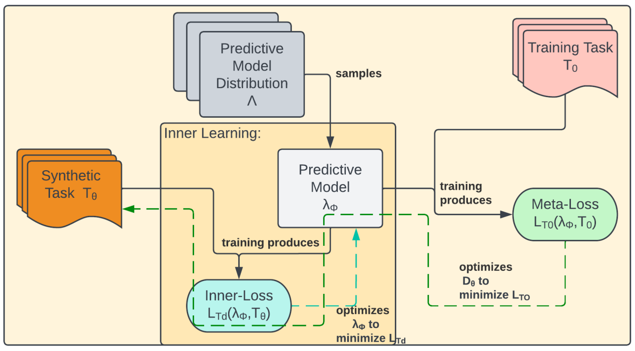


Figure 1: Training process for task-agnostic distillation based on Wang et al. [19]

2.2 Related Works in Dataset Distillation

A similar method to dataset distillation was proposed by Such et al. [14]. Their method uses meta-gradients to train a generator model to produce synthetic datasets, rather than directly optimizing one dataset. This method, the generative teaching network, was tested on supervised learning tasks and on the simple RL cart-pole environment using A2C. This inspired our experiments, though we base our method on that of Wang et al. [19] as our preliminary experiments indicated that parameterizing the dataset improved results.

Other distillation methods exist, such as gradient matching [25], trajectory matching [4], distribution matching [24], differentiable Siamese augmentation [23], and kernel ridge-regression [11]. We base our approach of that of Wang et al. [19] because of its simplicity and its independence from expert models and trajectories, though we posit that our method could be adapted to these approaches.

Non-distillation methods for simplifying learning, such as imitation learning, often require expert examples. Unlike distillation, imitation learning trains a single student model per run [22], while distillation produces a dataset that can produce many trained models inexpensively. Thus, imitation learning and distillation have very different use cases.

Light et al. [8] distills offline RL environment dataset. Our work distinguishes itself by focusing on online RL learning—sampling data by interacting with the environment (semi-supervised learning) rather than using a presampled offline dataset (supervised learning).

Lupu et al. [9] has similar motivations to our work and also tackles distilling reinforcement learning environments. Their approach uses evolutionary strategies to optimize the dataset, while we utilize meta-gradients calculated with PPO loss. Our work distinguishes itself by focusing on minimum-sized datasets and single-step learning, which is feasible given the direction provided by the meta-gradients.

2.3 Proximal Policy Optimization

PPO is a reinforcement learning algorithm that combines two policy gradient methods: actor-critic and trust region optimization. PPO uses two networks: the actor π_ϕ to learn the optimal policy, and the critic V_ψ to approximate the value function given policy π_ϕ. The actor determines the actions taken at each timestep at training and evaluation, while the critic evaluates the actions taken during training only; it performs no function at evaluation time. PPO actor loss over a batch of a given trajectory of observations is as follows [13]:

$$L_\pi = -\frac{1}{b}\sum_1^b \min\left[\frac{\pi_\phi^i(a_t|s_t)}{\pi_\phi^0(a_t|s_t)}A_t,\right. \\ \left.\text{clip}\left(\frac{\pi_\phi^i(a_t|s_t)}{\pi_\phi^0(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A_t\right] \tag{1}$$

where b is the batch size, π_ϕ^0 is the policy network with the parameters used to gather the environment transitions, π_ϕ^i is the policy network with its current parameters, a_t is the action taken and s_t is the state at time t, A_t is the advantage (which we calculate with generalized advantage estimation [12]), and ϵ is the trust region width.

PPO requires an auxiliary critic network, trained alongside the actor, to determine A_t for L_π. The critic is trained to approximate the return R_t expected from timestep t, given the agent is at state s_t and is following the policy π_ϕ. Batched critic loss is as follows [13]:

$$L_V = \frac{1}{b}\sum_{1}^{b}(V_\psi(s_t) - R_t)^2 \tag{2}$$

where b is batch size, V_ψ is the critic/value network, s_t is the state at time t, and R_t is the return at time t.

3 RL-to-SL Distillation

The learner λ—distillation-trained predictive model—must be capable of performing on both the true task T_0 (i.e. act as an RL agent) and the distilled task T_d (i.e. act as a classifier). In RL-to-SL distillation, λ must be capable of both producing a policy $\hat{p}$ for acting in the outer RL environment E_0 based on a state observation: $\hat{p} = \lambda(s)$ for all state observations s in E_0, and label prediction in the inner synthetic SL task based on a data instance: $\hat{y} = \lambda(x)$, for all $x \in X_d$. Once distillation is complete, a new learner $\lambda \in \Lambda$ trained on $\{X_d, Y_d\}_\theta$ should solve the environment E_0, or achieve an acceptable episodic reward on E_0 without training on E_0.

We define the synthetic data instances and labels to match the dimensionality of the states and actions of the RL environment, respectively. Thus, training the learner, the RL policy network, on the synthetic task is equivalent to learning to classify synthetic states into an action or regressing a state-dependent policy. However, the synthetic data instances are not guaranteed to be in or near the state space. The only constraint on the distilled dataset is how well the learners trained on it perform on E_0, as measured by policy loss.

The distillation meta-learning process only distills information pertinent to performing on the task. Thus, when we discuss distilling an environment, we mean that the learning trajectories and policies are distilled, not aspects such as transition and reward functions.

3.1 Proximal Policy Meta-Optimization

We consider PPO to be a great candidate for the distillation outer-learning objective due to the benefits of its trust region protections. However, a direct adaptation of dataset distillation using PPO as the outer objective is not sufficient to utilize these trust region protections. PPO's trust region is created by the clip function, which sets the gradient to 0 when the change in policy throughout learning on a given RL trajectory exceeds a threshold set by ϵ. This is because once clipping is applied, infinitesimal changes will not affect the result. This requires that the policy network π_ϕ be altered throughout training on the RL trajectory. However, only the synthetic dataset is changed by the policy loss during distillation. Thus, we alter dataset distillation such that the agents' initializations do not change throughout training on a given trajectory (i.e. over a single PPO epoch) to ensure that the only changes reflected in the change in policies is due to optimization on PPO loss, rather than differences in parameter initialization or model architecture.

We propose a new formulation for task distillation using PPO as the outer objective, and introduce it as *Proximal Policy Meta-Optimization* (PPMO) for RL-to-SL distillation. This incorporates the policy trust region protections of PPO.

The process to train the distilled dataset is detailed in Algorithm 1. Training consists of 3 nested loops: meta-epochs, policy epochs, and batched iterations. At each meta-epoch, a new agent/policy network

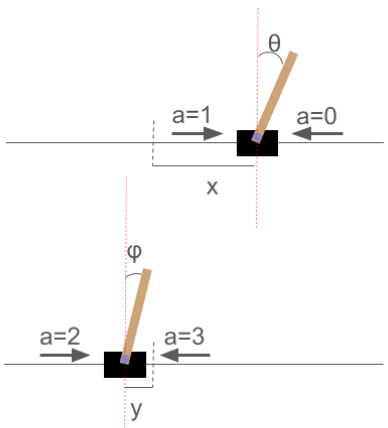
Algorithm 1: PPMO for RL-to-SL Distillation($\{X_d, Y_d\}_\theta$, Λ, V_ψ, E_0, e, n, b)

```
input : initialized synthetic dataset $\{X_d, Y_d\}_\theta$, learner
        distribution $\Lambda$, value network $V_\psi$, RL environment
        $E_0$, number of episodes per iteration $e$, number of
        policy epochs $n$, RL batch size $b$
output: learned synthetic dataset $\{X_d, Y_d\}_\theta$ distilled from
        $E_0$
1  while $\{X_d, Y_d\}_\theta$ has not converged do
2      $\lambda_{\phi_{init}}$ := Sample($\Lambda$);
3      $\pi^0_\phi$, $\nabla_\phi$ := Train($\lambda_{\phi_{init}}$, $\{X_d, Y_d\}_\theta$);
4      $\tau$ = PerformEpisodes($E_0$, $\pi^0_\phi$, $e$);
5      $i := 0$;
6      for policy epoch from $1$ to $n$ do
7          for $(s, a, r) \in \tau$ of batch size $b$ do
8              $L_\pi$, $L_V$ := PPOLoss($\pi^i_\phi$, $\pi^0_\phi$, $V_\psi$, $s$, $a$, $r$);
9              $\nabla\theta$ := BackpropagateMetaGradients($L_\pi$, $\theta$, $\nabla_\phi$);
10             $\nabla\psi$ := Backpropagate($L_V$, $\psi$);
11             Optimize $\theta$ and $\psi$ w.r.t. $\nabla\theta$ and $\nabla\psi$;
12             $\pi^{i+1}_\phi$, $\nabla_\phi$ := Train($\lambda_{\phi_{init}}$, $\{X_d, Y_d\}_\theta$);
13             $i := i + 1$;
14         end
15     end
16 end
17 return $\{X_d, Y_d\}_\theta$;
```

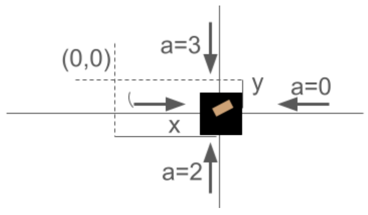
initialization $\lambda_{\phi_{init}}$ is sampled and trained on the synthetic dataset (lines 2-3). This training uses mean-squared error loss on the model's prediction of synthetic data X against the synthetic label Y, and we perform this learning in a single step of gradient descent. The model resulting from the first inner-training process of each meta-epoch, π^0_ϕ, defines the policy executed on the RL environment E_0, creating an RL trajectory τ (line 4). Just like in PPO, this trajectory is used for learning over multiple "policy epochs". For each batch sampled from the shuffled trajectory, PPO policy and value loss is calculated using the formulas defined in Equations 1 and 2, taking in the current parameters of the agent π^i_ϕ, the parameters of the agent used to gather the experiences π^0_ϕ, the value network V_ψ, and the batch of experience data (line 8). Our implementation calculates advantage A_t using GAE. The policy loss is our meta-loss: it is backpropagated through the inner-learning process to update the synthetic dataset $\{X_d, Y_d\}_\theta$ (lines 9 & 11). By updating the distilled dataset with the policy loss, the distilled dataset is optimized to maximize the reward achieved by the policy produced by inner learning. The critic is updated with standard optimization, i.e. not using meta-learning, as in standard PPO (lines 10-11). A new agent π^{i+1}_ϕ is trained from the same initialization $\lambda_{\phi_{init}}$ as the previous agents in this meta-epoch (line 12), and is used to calculate loss in the next sampled batch. Once all trajectory batches have been iterated through n times, the next meta-epoch iteration begins by sampling a new agent initialization (line 2). By the end of this process, the synthetic dataset can be used to train any agent initialization in Λ to perform on E_0.

4 ND Cart-Pole Experiments

To control the difficulty of cart-pole and the size of its state and action spaces, we expand the cart-pole system into multiple dimensions. Classic cart-pole has 1 degree of freedom in which the cart and pole can move. We extend this to N degrees of freedom, though the cart



(a) Two perpendicular side views of 2D cart-pole. The solid lines represent the degrees of freedom of the cart.



(b) Top-down view of 2D cart-pole. Both dimensions representing the 2 degrees of freedom are visible.

Figure 2: Two views of 2D cart-pole.

is limited to accelerating along one axis at each timestep, creating an action space of size $2N$. The state space is likewise expanded to $4N$: with one quadruple of cart position, cart velocity, pole angle, and pole angular velocity per degree of freedom. Just as in classic cart-pole, if the pole angle or cart position reach a threshold in any of the N dimensions, the experiment terminates. We provide two visualizations of 2D cart-pole in Figure 2. We utilize this extension of cart-pole to examine k-shot learning, generalization, and minimum distillation size in RL-to-SL distillation.

Our cart-pole experiments utilize a standard set of hyperparameters unless otherwise stated. We define the learner distribution Λ to contain a single architecture, α_c, with random initialization. The architecture, initializations, and PPO hyperparameters used in this work are in line with the standard of Huang et al. [7], except that we did not use learning rate annealing or value loss clipping.

4.1 k-Shot Learning

Once a synthetic dataset has been fully distilled, it can be used for k-shot learning. New agent models, including those not used in training the distilled dataset, can be trained on the synthetic task using k learning instances. In our experiments, these k instances are learned as a single batch in one gradient descent step. As the distilled dataset was trained to teach agents sampled from Λ, these agents are expected to outperform other agents on the task after k-shot learning—outliers within Λ and out-of-distribution agents may perform significantly worse on T_0 than the average performance of learners from Λ. However, generalization to unseen architectures is necessary for neural architecture search—a key use-case for distillation.

We verify the success of our distilled datasets by performing k-shot learning on randomly sampled agent architectures and initializations in Λ on the cart-pole environment (see summary of results in Table 1). Note that, because the agents in Λ used in our experiments share the same architecture but are produced by randomly initializing the parameters, we can assume that no agent used in verification was used in training. We also explore deviations from Λ to test k-shot learning on out-of-distribution learners. For each distribution tested, we sample 100 model architecture/initialization values, and train each model on the distilled dataset. The model is tested against the RL environment for 100 episodes, and the means and standard deviations of rewards over all the tested models is reported.

Our experiments showed little difference in k-shot learning based on the size of k: learning on the minimum-sized distillation (see Section 4.2) generally performed better than 512-shot with agents from Λ, but performed slightly worse than 512-shot with the other tested distributions. This implies that larger synthetic datasets are less likely to overfit to Λ. Our experiments show that k-shot learning on T_d appears to be more sensitive to the parameter initialization function (see Ortho and Xe experiments) than to architecture changes (see Random h and L experiments). This strengthens the claim that distillation can be used for neural architecture search. However, these distillations may still be biased to architectures close to those in Λ; using a wider array of architectures during training would ensure a fairer evaluation of varied neural architectures. We leave a thorough examination of generalization to novel architectures for future work.

4.2 Minimum Distillation Size

Distillation compresses a learning task, transforming it from many-shot learning to finite k-shot learning. k, the number of distilled instances to produce, is an important hyperparameter in distillation. Our experiments have determined that varying k has only a small effect on the quality of the distillation, past a certain threshold. Increasing k increases the computation costs of the distillation training non-negligibly, as all k instances are updated in each outer learning step and a synthetic dataset with more instances requires more epochs on average to converge. However, the end-learning costs are negligibly increased, so long as k fits into a single batch.

There is, however, a minimum threshold k_{min}, such that distillation fails when $k < k_{min}$. This number is determined by the geometry of the label space [15]. The distilled instances and labels must distinguish the classes or actions from each other. This can be done in fewer instances than the number of classes using soft labels. Using soft labels, the minimum number of instances to distinguish the classes of a label space with c discrete values is:

$$k_{min} = \lceil c/2 \rceil + 1 \tag{3}$$

This allows for what Sucholutsky and Schonlau [15] call "less than one"-shot learning,[2] or learning on less than one instance per class.

Equation 3 was one of multiple demonstrated by Sucholutsky and Schonlau [15]. We validate this equation experimentally on ND cart-pole and Atari. Distilling to $k \geq k_{min}$ instances results in a successful distillation—the expected reward for $\forall\lambda \in \Lambda$ trained on the distillation is approximately equal to the reward achieved by λ after direct RL training. As an example, 2D cart-pole distillation achieves a mean reward of 350 when $k = 512$ and 365 mean reward when $k = k_{min} = 3$. When $k < k_{min}$, the distillation is unsuccessful in teaching learners to solve the task, resulting in low average rewards consistent with random performance in cart-pole. 2D cart-pole distilled to $k = k_{min} - 1 = 2$ achieves an average of 40.6 reward.

[2] The term k-shot learning is not standardized. In this work, we define k as the total number of training instances, not training instances per class.

Table 1: k-shot learning on ND cart-pole. Each column represents an agent distribution (initialization and architecture), and each row represents an ND cart-pole distillation, trained only on models sampled from Λ. Agents are sampled, trained on the distillation, and tested on ND cart-pole, with rewards reported. The distributions are as follows: Λ is the agent distribution used in training, using a fixed architecture α_c and orthogonal initialization with a standard deviation $\sigma = 0.01$ on the final layer and $\sigma = \sqrt{2}$ on the others. "α_c: Ortho σ=1" is like Λ but with $\sigma = 1$ on all layers. "α_c: Xe" uses architecture α_c but is initialized using Xe initialization with the same σ as Λ. "α_c: Xe σ=1" uses Xe initialization with $\sigma = 1$ for all layers. "Random h" modifies architecture α_c by randomly sampling the hidden layer size from the set $[32, \dots, 256]$ (in α_c the size is fixed at 64). "Random L" modifies architecture α_c by randomly sampling the number of hidden layers (number of layers not counting the output layer) from the set $[1, \dots, 6]$ (in α_c it is fixed at 2). Both Random h and Random L used the same initialization as Λ. The Λ column represents base distillation performance, while the others represent generalization to new initializations and architectures.

Experiment	Λ	α_c: Ortho σ=1	α_c: Xe	α_c: Xe σ=1	Random h	Random L
1D 2-Shot	500 ± 0.0	238 ± 200	258 ± 207	175 ± 209	500 ± 0.0	500 ± 0.0
1D 512-Shot	500 ± 0.0	233 ± 221	246 ± 198	137 ± 178	500 ± 3.8	500 ± 0.0
2D 3-Shot	365 ± 31.9	16.3 ± 11.1	76.8 ± 67.3	13.4 ± 6.3	443 ± 55.7	415 ± 159
2D 512-Shot	350 ± 37.2	15.9 ± 10.5	68.6 ± 50.9	13.9 ± 7.8	457 ± 81.0	439 ± 99.8
3D 4-Shot	264 ± 20.6	12.4 ± 2.3	38.6 ± 21.5	10.9 ± 1.3	379 ± 93.3	305 ± 157
3D 512-Shot	240 ± 26.1	12.7 ± 3.6	41.8 ± 24.0	11.5 ± 2.4	401 ± 132	383 ± 116
4D 5-Shot	127 ± 7.4	12.5 ± 2.1	35.0 ± 12.2	11.0 ± 1.5	165 ± 29.8	156 ± 60.7
4D 512-Shot	129 ± 6.8	12.5 ± 2.0	39.7 ± 13.2	11.3 ± 1.6	203 ± 55.7	188 ± 46.1
5D 6-Shot	87.5 ± 4.3	12.8 ± 1.5	35.7 ± 9.2	11.5 ± 1.5	112 ± 18.3	125 ± 44.6
5D 512-Shot	87.5 ± 4.0	12.8 ± 1.8	41.0 ± 7.1	12.0 ± 1.5	140 ± 40.1	159 ± 30.4

The equation $k_{min} = \lceil c/2 \rceil + 1$ determines how many instances are required to distinguish the classes of the action space, assuming all actions are necessary. In ND cart-pole, all actions are needed to solve the task, and all actions are performed approximately at the same frequency. Thus, the distilled dataset must provide distinction between each action in its soft labels. That is, there must be at least one instance in which the action probability label is high, and one where it is low, relative to the other action probabilities in the label. See Figure 3 for a visualization of 1D and 2D cart-pole distilled to k_{min} instances.

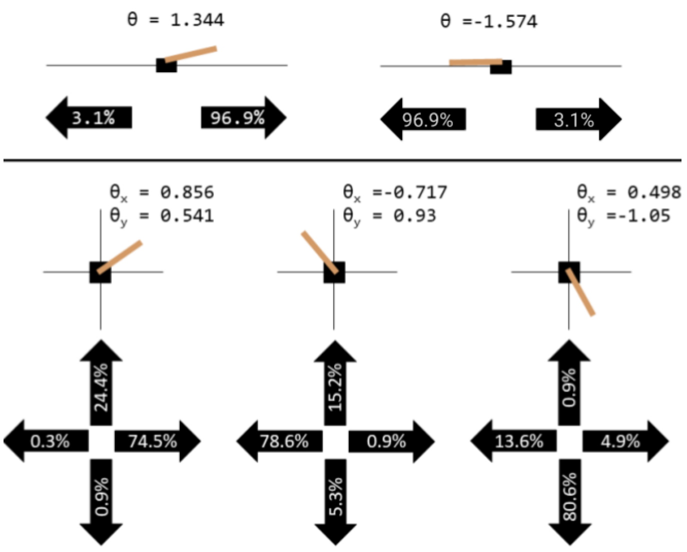


Figure 3: Visualizations of a distillation of 1D (above) and 2D (below) cart-pole with simplified information (only the angle and softmaxed labels). While a full examination of interpretability is outside the scope of this work, this demonstrates that some tasks when distilled provide clear interpretability. In both distillations, the action label demonstrates that the agent should move the cart in the direction in which the pole is leaning. In 2D cart-pole, which was distilled to 3 states despite having 4 actions, the "up" action never has the highest probability mass in any of the labels; instead, the mass is split between two instances. This demonstrates how distillations with fewer than c instances can teach a model to distinguish c classes or actions.

5 Atari Environments

We distill Atari environments to demonstrate RL-to-SL distillation scaling to more complex and difficult reinforcement learning tasks. The Atari environments have a much larger state and action space than cart-pole, as well as more complex reward functions, providing a significant challenge to reinforcement learning and distillation.

We have chosen a small selection of Atari tasks for brevity. We have avoided selecting tasks that are difficult for a standard PPO agent to learn, as finding tasks that can be learned by distillation but not by direct-task learning is beyond the scope of this work. As Atari tasks are mostly open-ended reward maximization problems, we determine the success of the distillation by comparing the average reward achieved by randomly sampled agent initializations from Λ trained on a fully distilled dataset $\{X_d, Y_d\}_\theta$, against the average reward achieved by randomly sampled learners from Λ trained directly on the environment using PPO. This is in line with the distillation ratio metric proposed by Sucholutsky and Schonlau [16], which also compares performance after direct-task learning versus distilled-task learning.

For the Atari experiments, we define Λ as the distribution of random initializations of a single architecture α_a, derived from the standardized PPO implementation of Huang et al. [7], as with parameter initialization and all PPO hyperparameters. For the sake of feasibility, we limit the amount of training time for the baseline RL experiments. While it may be possible for the agents to achieve a higher reward given more training time, it is difficult prove RL converged due to large performance plateaus during learning; thus, we consider time limits and apparent convergence as reasonable early-stop heuristics. In order to provide a more fair comparison to the converged full distillation results of *Centipede*, we trained the Centipede agent to 8000 epochs, giving it the same approximate number of update steps as in the fully distilled experiment. Those results are reported, yet note that the reward appeared to plateau at 1000 epochs. To provide a baseline for our RL agents themselves, we ensured our baseline PPO agents reached comparable performance to the results of DQN agents in the work of Mnih et al. [10].

Due to the increase in complexity in Atari environments versus cart-pole environments, the cost of distillation increases greatly as well. It may be unreasonable to commit the large amount of required resources for a full distillation of an Atari environment without knowing how long distillation will take or if distillation can

Algorithm 2: Meta-gradient Task Distillation with Encoding($\{X_d, Y_d\}_\theta$, Λ^l, ε^l_ξ, n, T_0)

```
input : distilled dataset $\{X_d, Y_d\}_\theta$, learner distribution $\Lambda^l$,
        encoder $\varepsilon^l_\xi$, number of inner iterations $n$, target
        training task $T_0$ with data instances $X_0$ and outer
        loss function $L$
output: distilled dataset $\{X_d, Y_d\}_\theta$ and trained encoder $\varepsilon^l_\xi$
1  while $\{X_d, Y_d\}_\theta$ and $\varepsilon^l_\xi$ have not converged do
2      $\lambda^l_\phi$ := Sample($\Lambda^l$);
3      $e^l_{X_d}$ := $\varepsilon^l_\xi(X_d)$;
4      $\phi$, $\nabla_\phi$, := Train($\lambda^l_\phi$, $e^l_{X_d}$, $Y_d$);
5      $e^l_{X_0}$ := $\varepsilon^l_\xi(X_0)$;
6      $L_{T_0}$ := $L(\lambda^l_\phi, e^l_{X_0})$;
7      $\nabla_\theta$, $\nabla_\xi$ := Backpropagate($\nabla_\phi$, $L_{T_0}$);
8      Optimize $\theta$ and $\xi$ w.r.t. $\nabla_\theta$ and $\nabla_\xi$;
9  end
10 return $\{X_d, Y_d\}_\theta$, $\varepsilon^l_\xi$;
```

succeed given the experimental setup; thus, we provide a method for controlling the difficulty of distillation by providing intermediate steps between direct-task learning and distillation. We do so by providing an algorithm that generalizes both direct-task learning and distillation. Then, we demonstrate this method on four Atari environments. We perform these experiments by distilling with $k = k_{min}$ for each of the four chosen Atari environments.

5.1 Encoder Rollback for Variable Difficulty

Due to the expense of distilling Atari games, we provide a method to scale the complexity of distillation training, providing evidence for the feasibility of full distillations. We utilize an encoder network to simplify the complex state spaces without reducing the dimensionality of the synthetic dataset. This formulation with the encoder as all but the final layer is defined and explored by Zhou et al. [26] as a combination of dataset distillation and neural feature regression. While they use this formulation to train downstream task learners, we expand this method for difficulty scaling.

An encoder network ε with parameters ξ transforms the learner model's input into a smaller space. We use our standard 5-layer Atari learner architecture α_a and split it along layer l, such that the encoder ε^l_ξ consists of layers $[0, l)$ of α_a and the learner λ^l_ϕ consists of layers $[l, 5)$. The encoder's output is fed directly into the learner, just as if they were still a single network. The difference between learner and encoder lies in training: the learner's parameters are still updated by inner learning on the synthetic dataset, and the learner is resampled from Λ^l each outer iteration. The encoder, however, is only initialized at the start of training and is trained through outer meta-learning on T_0 alongside the distillation, rather than with the learner.

We provide the encoder formulation of generalized task distillation as Algorithm 2. The encoder ε is initialized with the distilled dataset and, unlike the learner, is not reinitialized in the outer loop. ε encodes the data associated with both the synthetic task (line 3) and the real task (line 5), which in RL-to-SL distillation includes the synthetic data instances and the environment's states, respectively. The learner predicts on the encoded data, rather than directly on the tasks' data (lines 4, 6). While the learner's parameters are updated through inner learning, the encoder's parameters are updated through outer learning, alongside the distilled dataset (line 8). We utilize the encoder in inner learning, as that allows the distilled instances to be the same dimensionality throughout our encoder rollback experiments; thus, we can ensure that the distilled dataset is large enough to hold all information needed at any value of l.

While distilling with an encoder limits the use of the distillation to training only new learners that utilize that encoder, it does allow scaling the difficulty of distillation for tasks where a full distillation may be too computationally expensive. When $l = 0$, the encoder has 0 layers, and thus training is equivalent to distillation without using an encoder, which we refer to as *full distillation*. When $l = 5$ (all layers of α_a), the learner has 0 layers, and thus the algorithm is equivalent to directly training the encoder network on T_0, i.e. direct-task learning. Since direct-task learning is generally more stable and computationally cheaper, higher l values (larger encoders) represent easier distillation tasks, while lower l values (smaller encoders) represent harder distillation tasks. Thus, if we can successfully distill a task with a given encoder, and show that we can roll the encoder back by reducing l and successfully distill the task, we can show how the cost of distillation increases as l decreases. With this, we provide evidence for the feasibility of performing full distillations for tasks for which the cost of distillation is prohibitive given fixed resources.

5.2 Experiments

In the Atari experiments, a 5-layer agent (with architecture α_a) is used. These agents are standard for PPO learning and are capable of learning the tested environments directly. We demonstrate that the environments can also be learned with an encoder with $l = 4$, where the learner consists of a single linear probe, and for decreasing values of l, at the cost of increasing computational complexity. We test 100 agents on 100 episodes for each distillation experiment.

In general, as l decreases, the number of outer epochs to convergence increases, though the amount by which it increases varies across the environments and network layers. This makes it difficult to accurately predict the cost of a full distillation; however, this method does provide a smoother increase in computation cost per step than jumping from reinforcement learning to distillation. We were able to fully distill *Centipede*, due to its relatively cheap cost of full distillation. The other environments quickly became prohibitively expensive for our resources; however, we assert that full distillations of these environments is possible given sufficient resources (see Table 2).

While these experiments provide evidence for the feasibility of full distillations of these environments, the costs may be prohibitive. However, distillation frontloads computation costs and allows for cheaper training on downstream tasks—as low as one optimization step per model. Given a large enough search space for neural architecture search, distillation is a cost-effective solution. Training Atari agents is expensive, and each agent must explore the environment on its own. Distilling an Atari environment allows for training these agents in a single optimization step without exploration.

Our distillation of Atari *Centipede* provides a clear example of distillation's cost saving. On our hardware, standard reinforcement learning on *Centipede* took approximately 54 minutes while producing the distillation took 8 hours 17 minutes (see Table 3). Though distillation incurs a 9.2x upfront cost to RL, the benefits are quickly seen by the costs of training new agents on the distilled dataset. It takes 0.18 seconds to initialize a new model and train it on the 10-instance distilled dataset for 1 GD step. This cost is negligible compared to RL training. If we produce 10 or more agents, distillation will incur less costs overall, including the initial distillation training, than reinforcement learning. If applied to use cases that require large numbers of models, such as ensembling or neural architecture search,

Table 2: Results of distillation training with $k = k_{min}$, rolling back an encoder from $l = 5$ to $l = 0$. † indicates that the run was performed with parallelization (6 gpus), so the number of epochs is not comparable to non-parallelized runs. $*$ indicates the distillation has not fully converged. Empty cells represent experiments that did not exceed random performance due to high computational costs.

(a) Average End-Episode Reward Achieved at Convergence

	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	8011 ± 1049	8438 ± 758	7860 ± 756	8266 ± 561	8144 ± 614	8042 ± 492
Space Invaders	1508 ± 320	2123 ± 156	$1443^{*} \pm 245$	$768^{*\dagger} \pm 63$		$284^{*} \pm 2.1$
Ms. Pac-Man	4310 ± 521	3351 ± 91	$2950^{\dagger} \pm 269$	$2570^{\dagger} \pm 243$		$415^{*} \pm 119$
Pong	19.7 ± 1.2	21.0 ± 0.01	$5.6^{*\dagger} \pm 6.4$	$-9.1^{*\dagger} \pm 4.1$		

(b) Outer Epochs to Apparent Convergence

	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	1000	3000	4000	5000	40,000	8000
Space Invaders	115,000	90,000	175,000*	670,000*†		205,000*
Ms. Pac-Man	53,000	50,000	140,000†	1,300,000†		190,000*
Pong	5000	17,000	50,000*†	300,000*†		

Table 3: Comparison of the costs of distillation vs direct-task learning on Atari *Centipede* (see Table 2 for experiment results). Training was timed by performing computation on one NVIDIA A100 GPU. Note that while the initial distillation ("Distilling $\{X_d, Y_d\}_\theta$") is more costly than direct RL training, training new agents on the distilled dataset ("10-shot on $\{X_d, Y_d\}_\theta$") is significantly cheaper.

Training	Time per iteration	Total iterations	Total Datapoints
RL on E_0	3.25 s/epoch	1,000 epochs	8,000,000
Distilling $\{X_d, Y_d\}_\theta$	3.73 s/meta-epoch	8,000 epochs	64,000,000
10-shot on $\{X_d, Y_d\}_\theta$	0.18 s/batch	1 batch	10

the cost benefits are clear, as the costs increase by 0.18 seconds per model, rather than 54 minutes per model through standard RL.

6 Continuous Environments

We distill continuous RL environments due to their importance in fields such as robotics. We use the MuJoCo-v4 (Multi-Joint dynamics in Contact) environments [18], which have continuous state and action spaces. Continuous PPO uses the state to predict the mean value of each action and learns a fixed log standard deviation for each action, independent of the state. The distilled dataset is treated similarly to the discrete environments. The distilled instances and labels train the learner's mean action, while a state-agnostic standard deviation is learned. In addition, due to the continuous action space, we opted to distill to $k = 64$ instances, rather than determining k_{min} through trial-and-error. We trained all models for 24 hours on 4 GPUs. Note that some models, such as the Inverted Double Pendulum and Walker2D may not have converged. We report our results in Table 4.

The distillation-trained models outperform the random agents in each environment, and outperform the RL agents in 3 of the 8 environments. This demonstrates that our distillation algorithm is general-use, as it can succeed on both discrete and continuous action spaces. Success on this task specifically opens up the potential for distillation's use in robotics and other real-world RL tasks. Interestingly, the standard deviation of models trained by separate distillations is generally lower than that of the separate RL-trained models, even when the mean reward of the distillation-trained models is higher. This may imply that RL produces more outlier agents, and that the distillation process may favor different minima to those that the RL training finds. Just like in the *Centipede* example, the distilled dataset can train models quickly, in a single step of gradient descent, allowing for many RL agents to be produced at low cost.

Table 4: Results of 5 distillations on each of the MuJoCo continuous environments. PPO RL performance of 5 agents and uniform random agent performance are also recorded for comparison. For RL, each agent performs 1000 episodes, and for distillation, 1000 episodes are recorded from separate agents trained on the distilled dataset. The means of each RL agent or distilled dataset are taken, and the mean and standard deviation across the 5 agents/datasets are recorded.

	Random	RL	Full Distillation
Ant	-52 ± 88	3980 ± 669	3115 ± 319
Half Cheetah	-282 ± 79	1908 ± 1347	2226 ± 242
Hopper	21 ± 23	1441 ± 532	2168 ± 302
Humanoid	123 ± 37	631 ± 486	544 ± 32
H. Standup	$34,034 \pm 3028$	$104,950 \pm 53,667$	$138,478 \pm 6037$
Inv. Double Pend.	57 ± 16	7553 ± 1036	1235 ± 427
Reacher	-42.95 ± 4.05	-5.63 ± 0.22	-7.78 ± 0.57
Walker2D	2.1 ± 5.1	3537 ± 432	331.7 ± 117

7 Discussion

We have demonstrated distillation of cart-pole, Atari, and MuJoCo environments into compact, single-batch datasets. We have demonstrated the benefits of distillation in reducing training costs. While the up-front expenses of distillation are not insignificant, for applications that require training many models, the per-model speedup is significant. The cost may be reduced by using better reinforcement learning algorithms, better hyperparameters, and parallelization. The performance of distillation-trained models may be improved with a longer distillation process or a robust hyperparameter search.

Distillation has been proposed for use in downstream tasks such as data anonymity [5] and neural architecture search [19], and could be viable for other search problems, such as finding high-performance models or good parameter initializations. In addition, methods that require multiple end-task models, such as ensembling, may benefit from distillation. However, distillation has not yet been widely adopted for these use cases, likely due to applications being an afterthought in distillation research and to the computational expense of distillation. Generalized task distillation, and RL-to-SL distillation in particular, opens avenues for real-world application by expanding the types of problems that distillation can be used on.

We intend for our future work to examine use cases of task distillation to prove its viability for real-world applications, such as neural architecture search. We hope to improve interpretability and understanding of how and why distillation works. We also intend to expand distillation to other real and synthetic tasks, such as more complex RL algorithms.

References

[1] Atari. Pong. Arcade, Atari 2600, 1972.

[2] Atari. Centipede. Arcade, Atari 2600, 1980.

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[4] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu. Dataset distillation by matching training trajectories. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 4749–4758, 2022.

[5] T. Dong, B. Zhao, and L. Lyu. Privacy for free: How does dataset condensation help privacy? In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 5378–5396, 2022.

[6] General Computer Corporation. Ms. Pac-Man. Arcade, Atari 2600, 1982.

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

[8] J. Light, Y. Liu, and Z. Hu. Dataset distillation for offline reinforcement learning. *arXiv*, arXiv:2407.20299, 2024.

[9] A. Lupu, C. Lu, J. Liesen, R. T. Lange, and J. Foerster. Behaviour distillation. In *International Conference on Learning Representations*, 2024.

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

[11] T. Nguyen, Z. Chen, and J. Lee. Dataset meta-learning from kernel ridge-regression. In *International Conference on Learning Representations*, 2021.

[12] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv*, arXiv:1707.06347, 2017.

[14] F. P. Such, A. Rawal, J. Lehman, K. O. Stanley, and J. Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9206–9216, 2020.

[15] I. Sucholutsky and M. Schonlau. "Less than one"-shot learning: Learning N classes from M < N samples. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, volume 35, pages 9739–9746, 2021.

[16] I. Sucholutsky and M. Schonlau. Soft-label dataset distillation and text dataset distillation. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2220–2227, 2021.

[17] Taito. Space Invaders. Arcade, Atari 2600, 1978.

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

[19] T. Wang, J.-Y. Zhu, A. Torralba, and A. Efros. Dataset distillation. *arXiv*, arXiv:1811.10959, 2018.

[20] C. Wilhelm and D. Ventura. "Distilling reinforcement learning into single-batch datasets". *arXiv*, preprint arXiv:2508.09283, 2025. Full version of this paper.

[21] C. Wilhelm and D. Ventura. Code and data for "Distilling reinforcement learning into single-batch datasets". Zenodo, 2025. Available at https://doi.org/10.5281/zenodo.16658503.

[22] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*, 54(12):7173–7186, 2024.

[23] B. Zhao and H. Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, 2021.

[24] B. Zhao and H. Bilen. Dataset condensation with distribution matching. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6503–6512, 2023.

[25] B. Zhao, K. R. Mopuri, and H. Bilen. Dataset condensation with gradient matching. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[26] Y. Zhou, E. Nezhadarya, and J. Ba. Dataset distillation using neural feature regression. In *Advances in Neural Information Processing Systems 35*, 2022.