A Appendix

A.1 AlphaStar cost estimation

We estimate the cost of AlphaStar [85] based on the following description in the paper: "In StarCraft, each player chooses one of three races — Terran, Protoss or Zerg — each with distinct mechanics. We trained the league using three main agents (one for each StarCraft race), three main exploiter agents (one for each race), and six league exploiter agents (two for each race). Each agent was trained using 32 third-generation tensor processing units (TPUv3) over 44 days."

This corresponds to a total of 12 agents (= 3 main + 3 exploiter + 6 league) trained for a total of 1056 TPU hours = 44 days * 24 hours/day on 32 TPU v3 chips. As per current pricing at https://cloud.google.com/tpu/pricing, TPUv3 (v3-8) cost $8 per hour. Based on this, we estimate the cost of replicating AlphaStar results for an independent researcher would be at $1056 * 12 * 32 * \$8 = \$3,244,032$. Please note that we are only considering the cost of replication and not accounting for other costs such as hyperparameter tuning and evaluation.

A.2 Compute resources for PVRL experiments

For experiments in section 4, we used a P100 GPU. For obtaining the teacher policy, the cost of running the tabula rasa DQN for 400M frames for 10 games on 3 seeds each roughly amounts to 7 days x 30 = 210 days of compute on a P100 GPU. For each of the PVRL methods, we trained on 10 games with 3 runs each for 10M frames, which roughly translates to 4-5 hours. For offline pretraining, we train methods for 1 million gradient steps with a batch size of 32, which roughly amounts to 6-7 hours (this could be further sped up by using large-batch sizes). We list the number of configurations we evaluated for each method below.

- Rehearsal: We tried 5 values of teacher data ratio $\rho \times 4$ n-step values, amounting to a total of 20 configurations.
- JSRL: We tried 4 values of teacher roll-in steps $\alpha \times 2$ values of decay parameter $\beta \times 2$ values of n-step, amounting to 16 configurations.
- RL Pretraining: We tried 2 values of $\lambda \times 4$ values of n-step, amounting to 8 configurations.
- Kickstarting: We report results for 4 values of n-step for a specific temperature and distillation loss coefficient. For hyperparameter tuning, we evaluated 2 temperature values and 2 loss coefficients with a specific n-step. Overall, this corresponds to a total of 8 configurations.
- DQfD: We report results for 4 values of n-step $\times$ 2 values of margin loss coefficients $\times$ 2 values of margin parameters, amounting to 16 configurations.
- QDagger: We report results for 4 n-step values for a specific temperature and distillation loss coefficient. For hyperparameter tuning, we evaluated 2 different temperature and loss coefficients with a specific n-step, akin to kickstarting. Overall, this amounts to 8 configurations.

Based on the above, we evaluated a total of 60 (=$20 + 16 + 16 + 8$) configurations without pretraining while 32 configurations with pretraining. Each of these configurations was evaluated for 30 seeds. This amounts to a total compute time of 300-375 days for runs without pretraining on offline data while 400-480 days of GPU compute for runs involving pretraining, resulting in a total compute time of around 700 - 855 days on a P100 GPU.

A.3 PVRL: Experimental details

Atari 2600 Games: The subset of games in the paper includes games from the original Atari training set used by Bellemare et al. [10] (Asterix, Beam Rider, Seaquest and Space Invaders) as well as validation games used by Mnih et al. [60] (Breakout, Enduro, River Raid), except the games which are nearly solved by DQN, such as Freeway and Pong. We do not use any hard exploration games as a DQN teacher does not provide a meaningful teacher policy for such games. The remaining three games were chosen to test the student agent in environments with challenging characteristics such as requiring planning as opposed to being reactive (*e.g.,* Ms Pacman, Q*Bert), and sparse-reward games that require long-term predictions (Bowling). Furthermore, most of these games can be significantly improved over the teacher DQN performance, which is sub-human on half of the games (Ms Pacman, Q*Bert, Bowling, Seaquest and Beam Rider). Refer to Table A.2 for per-game teacher scores.

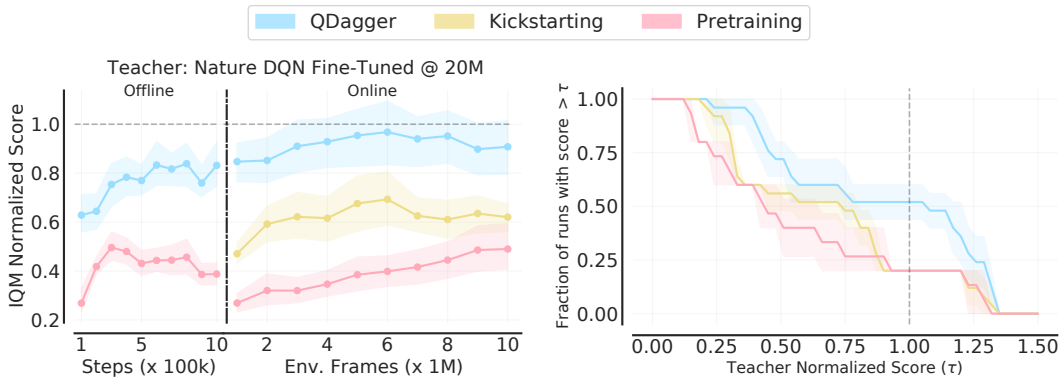

Figure A.11: **Comparison of best-performing PVRL algorithms** for reincarnating a student DQN agent given a teacher policy and replay buffer from a Nature DQN agent trained for 200M frames followed by fine-tuning with Adam for 20M frames (Panel 2 in Figure 1). While the performance of the reincarnated agents depends on the teacher, the ranking of PVRL algorithms remain consistent wrt Figure 2, that is, QDagger > Kickstarting > offline RL pretraining. Shaded regions show 95% bootstrap CIs. **Left**. Sample efficiency curves based on IQM normalized scores, aggregated across 10 games and 3 runs. **Right**. Performance profiles.

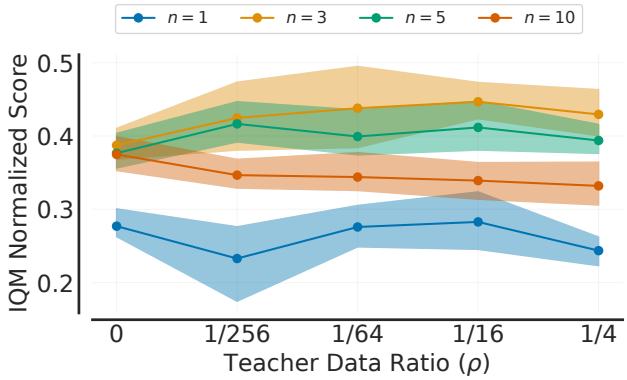

Figure A.12: **Rehearsal** for PVRL. The plots show IQM teacher normalized scores after training for 10M frames, aggregated across 10 Atari games with 3 seeds each. Each point in the above plots correspond to a distinct experiment setting evaluated using 30 seeds. Shaded regions show 95% CIs [2].

Common hyperparameters. We list the hyperparameters shared by all PVRL methods in Table A.1. For all methods, we swept over n-step returns in $\{1, 3, 5, 10\}$ except for DQfD [36], which originally used 10-step returns, where we only tried 3-step and 10-step returns and JSRL for which we tried only 1-step and 3-step returns. For PVRL methods without any pretraining phase, we use a learning rate of $6.25e^{-5}$ (JSRL, Rehearsal), following the hyperparameter configuration for Dopamine [15]. For methods that pretrain on offline data, we sweep over $\{6.25e^{-5}, 1e^{-5}\}$ and found the learning rate of $1e^{-5}$ to perform better in our early experimentation and use it for our main results. We discuss the method specific hyperparameters below. For obtaining the teacher policy using value-based agent, we use the $\text{softmax}(Q_T(s, \cdot)/\tau)$ over the teacher's Q-function Q_T and use the same temperature coefficient τ for both the student and teacher policy.

- **Rehearsal**: We tried 5 different values of the teacher data ratio (ρ) in $\{0, 1/256, 1/64, 1/16, 1/4\}$. The loss, $\mathcal{L}_{Rehearsal}$, can be written as: $\mathcal{L}_{Rehearsal} = \rho\mathcal{L}_{TD}(\mathcal{D}_T) + (1-\rho)\mathcal{L}_{TD}(\mathcal{D}_S)$. As can be seen from the results in Figure A.12, we find a small value of teacher data ratio ($\rho = 1/16$) with 3-step returns to be the best performing configuration.

- **JSRL**. As shown in Figure 4 (left), we swept over the maximum number of teacher roll in steps of α in $\{0, 100, 1000, 5000\}$, and the decay parameter β, which governs how fast we decay the roll-in steps, in $\{0.8, 1.0\}$. Note that $\beta = 1.0$ corresponds to JSRL-Random, which was found to be competitive in performance to JSRL [83].

Table A.1: **Common hyperparameters** used by PVRL experiments using a DQN student agent on ALE in Section 4. These hyperparameters are based on the ones used by the Jax implementation of DQN in Dopamine [15].

Hyperparameter	Setting
Sticky actions	Yes
Sticky action probability	0.25
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Frame skip (Action repetitions)	4
Reward clipping	[-1, 1]
Terminal condition	Game Over
Max frames per episode	108K
Discount factor	0.99
Mini-batch size	32
Target network update period	every 2000 updates
Min replay history	20000 steps
Environment steps per training iteration	250K
Update period every	4 environment steps
Training ϵ	0.01
Training ϵ-decay steps	50K
Evaluation ϵ	0.001
Evaluation steps per iteration	125K
Q-network: channels	32, 64, 64
Q-network: filter size	8×8, 4×4, 3×3
Q-network: stride	4, 2, 1
Q-network: hidden units	512
Hardware	P100 GPU
Offline gradient steps per iteration	100K
Offline training iterations	10
Offline learning rate	0.0001

- **RL Pretraining**: We use CQL, which optimizes the following loss:

$$\mathcal{L}_{Pretrain} = \mathcal{L}_{TD}(\mathcal{D}_T) + \lambda \mathbb{E}_{s,a\sim\mathcal{D}_T}\left[\log\Big(\sum_{a'} Q(s,a')\Big) - Q(s,a)\right] \tag{A.3}$$

  The choice of CQL is motivated by its simplicity as well as recent findings that offline RL methods that do not estimate the behavior policy are more suited for online fine-tuning [63].

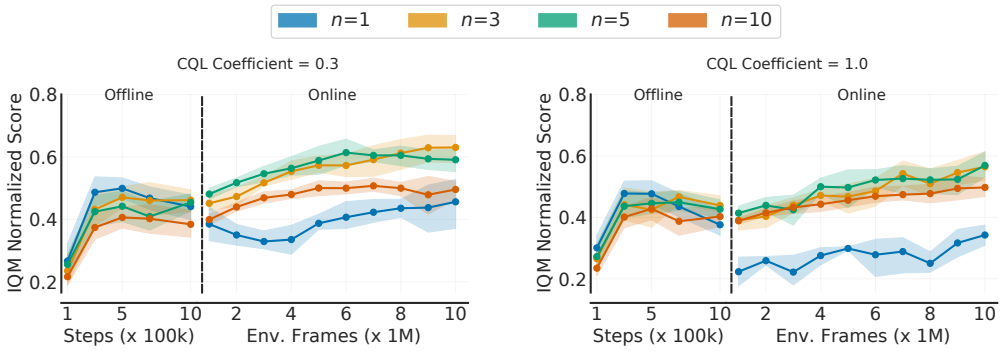

Figure A.13: **RL pretraining** using CQL [46], followed by fine-tuning with **Left**. CQL coefficient 0.3, and **Right**. CQL coefficient 1.0. The plots shows IQM teacher normalized scores over the course of training, computed across 30 seeds, aggregated across 10 Atari games. Online fine-tuning degrades performance with 1-step returns, which is more pronounced with higher CQL loss coefficient.

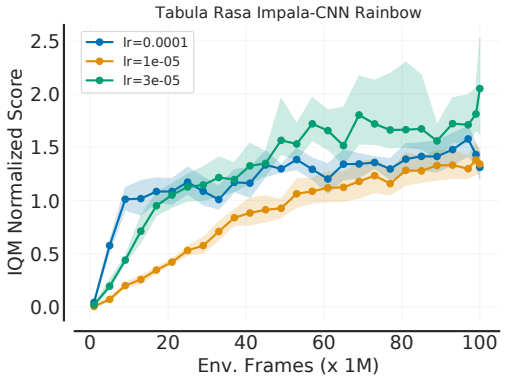

Figure A.14: **Tabula rasa Impala-CNN Rainbow**. Results for different learning rates for the tabula rasa Impala-CNN Rainbow agent. Learning rate of $3e-5$ performs the best. The plots show IQM normalized scores aggregated scores 10 Atari games with 3 seeds, while the shaded regions show 95% bootstrap CIs.

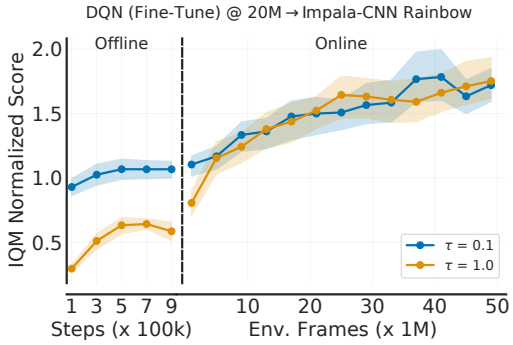

Figure A.15: **Effect of QDagger temperature** τ on the performance of reincarnated Impala CNN-Rainbow in Figure 1. A lower temperature coefficient (0.1) results in better performance in the offline pretraining phase but converges to similar performance in the online phase.

We tried two different values of CQL coefficient λ (0.3 and 1.0), as shown in Figure A.13, and report the results for the better performing coefficient ($\lambda = 0.3$) in the main paper.

- **Kickstarting**. Kickstarting [74] uses the same loss as QDagger (Equation 2), but as discussed in the main paper, kickstarting does not have any pretraining phase on offline data. For the temperature hyperparameter τ for obtaining the policy $\pi(\cdot|s) = \text{softmax}(Q(s,\cdot)/\tau)$, we tried 0.1 and 1.0. Similarly, we swept over two initial values for distillation loss coefficient (λ_0), namely 1.0 and 3.0. For experiments with the DQN student, we found the coefficient 3.0 and temperature 0.1 to perform the best.
- **DQfD**. Following Hester et al. [36], DQfD uses the following loss for training:

$$\mathcal{L}_{DQfD}(\mathcal{D}) = \mathcal{L}_{TD}(\mathcal{D}) + \eta_t \mathbb{E}_{s\sim\mathcal{D}}\left[\max_a \left(Q(s,a) + f(a_T(s),a)\right) - Q(s,a_T(s))\right] \quad \text{(A.4)}$$

where η_t corresponds to the margin loss coefficient, $a_T(s) = \text{argmax}_a \pi_T(a|s)$ and $f(a_T, a)$ is a margin function that is 0 when $a = a_T$ and a positive margin m otherwise. We swept over the values $\{1.0, 3.0\}$ for both the margin parameter m and initial values η_0 of the margin loss coefficient η_t. For the DQN student, we report the results for the better performing margin coefficients in the main paper.

- **QDagger**. Akin to kickstarting, we swept over the values of temperature τ in $\{0.1, 1.0\}$ and distillation coefficient λ_0 in $\{1.0, 3.0\}$. For ALE experiments, we decay the distillation loss coefficient every training iteration (1M environment frames) using the fraction of expected returns obtained by the student policy π compared to the teacher policy π_T, that is, $\lambda_t = \mathbf{1}_{t<t_0} \max(1 - G^\pi/G^\pi_T, 0)$. For fair comparisons with other methods, we use the same strategy for decaying the distillation loss coefficient λ_t for Kickstarting and the margin loss coefficient η_t for DQfD.

A.4 Reincarnating RL as a workflow: Additional details

Revisting ALE. We used the final model checkpoints of Nature DQN [60] from Agarwal et al. [1], which was trained for 200M frames using the hyperparameters in Dopamine [15]. The fine-tuned DQN (Adam) in panel 2 in Figure 1 uses 3-step returns. For the tabula rasa Impala-CNN Rainbow, we use similar hyperparameters to Dopamine Rainbow except for learning rate (lr), for which we ran a sweep over $\{1e-4, 1e-5, 3e-5\}$, shown in Figure A.14, and use the best performing lr of $3e-5$. For the reincarnated Impala-CNN Rainbow in Panel 3, we use a QDagger distillation coefficient of 1.0 and sweep over temperature parameter τ in $\{0.1, 1.0\}$, as shown in Figure A.15. Consistent with our other fine-tuning results on ALE, in Panel 3, using a reduced lr of $3e-6$ for fine-tuning the already fine-tuned DQN agent results in better performance, compared to using an lr of $1e-5$, as used by fine-tuned DQN.

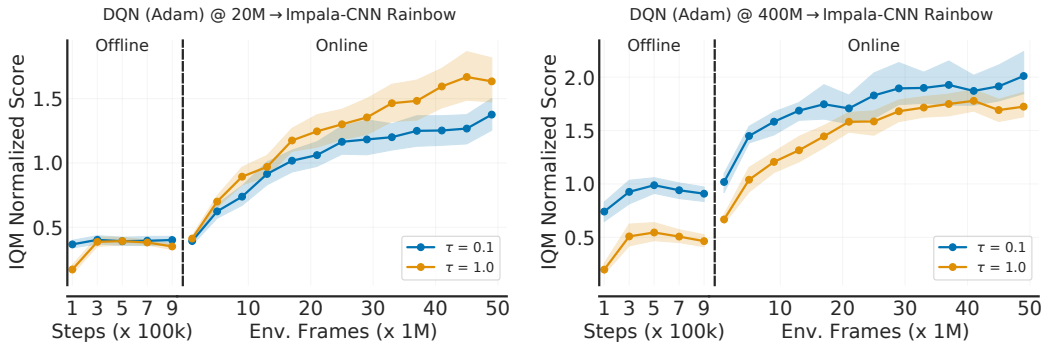

Figure A.16: **Effect of QDagger temperature** τ on the performance of reincarnating Impala CNN-Rainbow from **Left**. DQN (Adam) @ 20M, and **Right**. DQN (Adam) @ 400M. Notably, the better performing temperature τ is dependent on the teacher policy. Lower τ results in cloning a more "spikier" teacher policy. With a reasonably good teacher policy (DQN @ 400M), τ value of 0.1 performs better than 1.0 while with a more suboptimal teacher policy (DQN @ 20M), the higher temperature coefficient of 1.0 performs better than 0.1.

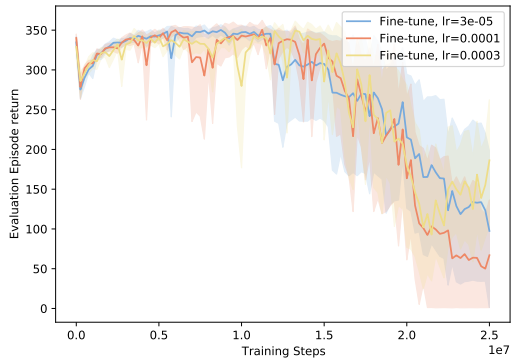

Figure A.17: **Fine-tuning TD3**. Results for fine-tuning a trained Acme TD3 agent with different learning rates. All the different learning rates exhibit similar performance trends including severe degradation after prolonged training.

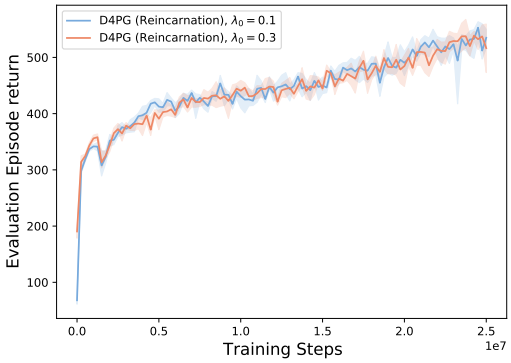

Figure A.18: **Effect of varying initial QDagger distillation coefficient** λ_0 on the performance of reincarnated D4PG. Higher coefficient (0.3) results in faster transfer but converges to similar performance to the runs with lower coefficient (0.1).

A.4.1 Humanoid:Run

The purpose of humanoid:run experiments is to show the utility of reincarnation in a complex continuous control environment given access to a pretrained policy and some data from its final replay buffer. Irrespective of performance from fine-tuning performance TD3, we demonstrate the benefits of using reincarnation over tabula rasa D4PG. For example, reincarnated D4PG achieves performance obtained by SAC in 10M frames[3] in only half the number of samples (*i.e.,* 5M frames).

For obtaining the pretrained policy, TD3 was chosen as (1) it's a well-known method for continuous control, and (2) we could use an off-the-shelf JAX implementation in Acme, which is competitive on humanoid-run. For tabula rasa TD3 [29], we used a learning rate (lr) of $3e-4$ for both the policy and critic, which are represented using a MLP with 2 hidden layers of size (256, 256). For other hyperparameters, we used the default values in Acme's TD3 implementation.

For fine-tuning TD3, we use the last 500K environment steps from the TD3's replay buffer and use the same hyperparameters as tabula rasa TD3 except for lr. Specifically, we swept over the lr in $\{1e-4, 3e-4, 3e-05\}$, shown in Figure A.17, and find that all lrs exhibit performance degradation with prolonged training. We hypothesize that this degradation is likely caused by network capacity loss with prolonged training in value-based RL methods [47, 56]. We believe that this issue does not affect our conclusions about the efficacy of reincarnating RL over a tabula rasa workflow. Further investigation of this performance degradation is outside the scope of the present work.

[3] See https://github.com/denisyarats/pytorch_sac for SAC learning curves.

For the tabula rasa D4PG [8], we use MLP networks with 3 hidden layers of size $(256, 256, 256)$ for the policy and $(512, 512, 256)$ for the critic. We used $3e-4$ as the learning rate and a sigma value of 0.2 that sets the variance of the Gaussian noise to the behavior policy. Other hyperparameters use the default values for D4PG implementation in Acme. For reincarnating D4PG using QDagger, we minimize a distillation loss between the D4PG's actor policy and the teacher policy from TD3 jointly with the actor-critic losses. For QDagger specific hyperparameters, we pretrain using 200K gradient updates as well as decay λ_t to 0 over a period of 200K gradient updates during the online training phase. Additionally, we sweep over distillation coefficient λ_0 in $\{0.1, 0.3\}$, as shown in Figure A.18. Note that both TD3 and D4PG use 5-step returns by default in Acme.

A.4.2 Balloon Learning Environment (BLE)

Details about BLE can be found in [33]. For self-containedness, we include important details below.

Station Keeping. Stratospheric balloons are filled with a buoyant gas that allows them to float for weeks or months at a time in the stratosphere, about twice as high as a passenger plane's cruising altitude. Though there are many potential variations of stratospheric balloons, the kind emulated in the BLE are equipped with solar panels and batteries, which allow them to adjust their altitude by controlling the weight of air in their ballast using an electric pump. However, they have no means to propel themselves laterally, which means that they are subject to wind patterns in the air around them. By changing its altitude, a stratospheric balloon can surf winds moving in different directions.

BLE Evaluation. The goal of a BLE agent is to station-keep, *i.e.,* to control a balloon to stay within 50 km of a fixed ground station by changing its altitude to catch winds that it finds favorable. We measure how successful an agent is at station-keeping by measuring the fraction of time the balloon is within the specified radius, denoted TWR50 (*i.e.,* the time within a radius of 50km).

Environment Details. The observation space in BLE is a 1099 dimensional array of continuous and boolean values. An agent in BLE can choose from three actions to control the balloon: move up, down, or stay in place. The balloon can only move laterally by "surfing" the winds at its altitude; the winds change over time and vary as the balloon changes position and altitude More details can be found at https://balloon-learning-environment.readthedocs.io/en/latest/environment.html.

Perciatelli Teacher and Data. Perciatelli is the name of the RL agent (QR-DQN with a MLP architecture) trained by Bellemare et al. [11] using a distributed RL setup for more than a month in the production-level Loon simulator and released with the BLE environment after fine-tuning for 14M steps in BLE. For the data, we use the final replay buffer (of size 2M transitions) logged by the Perciatelli agent during BLE fine-tuning.

Architectures. The network architecture used by QR-DQN [23] and Perciatelli is a multilayer perceptron (MLP) with 7 hidden layers of size 600 each with ReLU activations and approximate the distribution using 51 fixed quantiles. For the DenseNet architecture [39] employed by IQN [22] and R2D6, we use 7 hidden layers of size 512 each (for TPU-efficiency), which contains significantly more parameters than the Perciatelli MLP. Additionally, R2D6 uses a LSTM layer of size 512 on top of the DenseNet encoder with QR-DQN loss, while IQN samples 128 quantiles for minimizing the implicit quantile regression loss.

Hyperparameters. We set most hyperparameters for our distributed RL agents based on configuration of the BLE Quantile agent, which can be found at agents/configs/quantile.gin. For tabula rasa agents, we swept over the lr in $\{2e-6, 6e-6, 1e-5\}$ and found $1e-5$ to be the best performing lr. However, for fine-tuning Perciatelli, we found that a lower lr of $1e-6$ performs better. For reincarnated R2D6 and IQN, we use a lr of $1e-5$ and $6e-6$ respectively during the online phase while $2 \times lr$ in the offline pretraining phase. We set distillation temperature τ to be equal to 1.0 and linearly decay the QDagger distillation coefficient λ_t over a fixed number of learner steps (1M for R2D6 and 160000 for IQN). Furthermore, we swept over the initial value of λ_t (λ_0) in $\{0.3, 1.0\}$ and found 1.0 to be better for R2D6 while 0.3 for IQN.

A.5 Additional ablations for QDagger

Dependence on offline replay data. In the PVRL setting on ALE, we assumed access to the last replay buffer (containing 1M transitions) of the teacher DQN (Adam) agent trained for 400M frames. This assumption is aligned with maximally reusing existing computational work and often holds

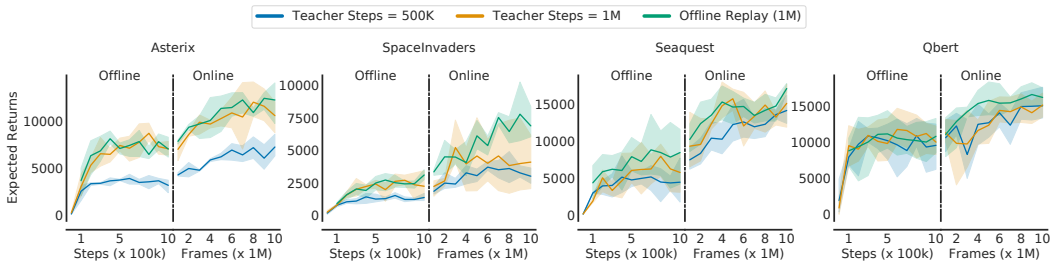

Figure A.19: **Substituting offline replay with teacher collected data** for pretraining phase of QDagger for reincarnating a student DQN given access to the teacher policy obtained from a DQN (Adam) trained for 400M frames. For collecting data using the teacher, we roll out the teacher policy with ϵ-greedy exploration where we decay ϵ from 1 to 0 linearly for the number of teacher steps. Note that 1 transition corresponds to 4 frames in Atari (due to an action repeat of 4).

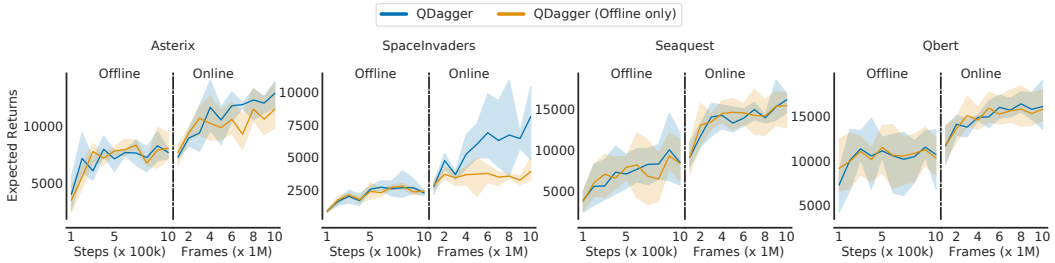

Figure A.20: **Effect of online phase of QDagger** on the performance of a DQN student reincarnated using QDagger, given access to the teacher policy and final replay buffer obtained from a DQN (Adam) trained for 400M frames. While using QDagger only during offline phase performs comparably to QDagger during both online and offline phase on Seaquest and Q*Bert, the online phase leads to slight improvement on Asterix and significant improvements on Space Invaders.

in practice as last replay buffer of RL agents is typically logged on disk in addition to their policy checkpoints. However, it is possible that we only have access to the teacher policy. In this scenario, even if we don't have access to the teacher's replay buffer, we can generate data using the teacher policy during the online RL phase (this would cost environment samples) and use this teacher collected data for pretraining with QDagger loss (analogous to the behavior cloning phase in Dagger).

To verify this empirically, we ran a QDagger ablation on 4 games where we do not assume access to the teacher's replay buffer. As shown in Figure A.19, we found that collecting the same amount of teacher data as the offline replay leads to comparable performance on 3/4 games (Asterix, Qbert, Seaquest). Somewhat surprisingly, collecting only 500K transitions from teacher results in comparable performance to using 1M transitions, except on Asterix where we see substantially lower performance. The slightly better performance with offline replay could be related to its better diversity than teacher collected data. While we used ϵ-greedy exploration with the teacher policy, the offline replay data is collected using possibly diverse policies, as the teacher agent was continually updated during training.

Effect of QDagger loss in online phase. In this work, we use QDagger loss during both the offline pretraining and online RL phase (Equation 2). To evaluate the benefit of QDagger loss in the online phase on ALE, we ran an ablation on 4 games where we use standard Q-learning instead of QDagger loss during the online RL phase. As shown in Figure A.20, we see that the online phase helps improve performance on some games (*e.g.,* Space Invaders, Asterix) while not having much effect on others. The impact of the QDagger loss in the online phase also depends on the offline replay buffer size. For example, on the more complex BLE domain, we only had access to a small amount of replay data relative to the training budget of the Perciatelli teacher. With this teacher, using the QDagger loss only during the offline phase resulted in much lower performance than using QDagger for both offline and online phase.

A.6 Per-game Scores on ALE

Table A.2: Average scores for the random agent, human agent and the DQN (Adam) trained for 400 million frames (based on 5 runs). For teacher normalized scores reported in the paper, the random agent is assigned a score of 0 while the DQN (Adam) agent is assigned a score of 1. Normalization using teacher allow us to compare the performance of student agents relative to the teacher and avoid high performing outliers (such as Breakout and Space Invaders) in terms of human normalized scores.

Game	Human	DQN @400M	Random
Asterix	8503.3	13682.6	210.0
Beam Rider	16926.5	6608.4	363.9
Bowling	160.7	33.9	23.1
Breakout	30.5	234.2	1.7
Enduro	860.5	1142.0	0.0
Ms Pacman	6951.6	4366.2	307.3
Q∗bert	13455.0	11437.3	163.9
River Raid	17118.0	17061.9	1338.5
Seaquest	42054.7	14228.2	68.4
Space Invaders	1668.7	7613.6	148.0

Table A.3: Per-game scores for the top-performing methods in Figure 2. Mean scores along with minimum and maximum scores, shown in brackets, across 3 runs.

Game	QDagger	Kickstarting	Offline Pretraining
Asterix	12301.1 (9802.2, 14118.4)	7872.0 (6979.4, 8691.9)	4132.2 (3802.6, 4533.1)
BeamRider	6428.3 (6103.2, 6925.3)	5330.9 (4964.8, 5661.2)	1854.5 (1794.6, 1942.4)
Bowling	34.1 (30.0, 38.5)	31.3 (30.0, 33.0)	25.5 (16.4, 30.1)
Breakout	277.6 (254.5, 295.8)	237.6 (191.0, 281.7)	126.3 (118.2, 133.1)
Enduro	1765.6 (1415.1, 2161.3)	1167.3 (1031.9, 1268.2)	1167.9 (1124.7, 1222.9)
MsPacman	5110.6 (4742.7, 5325.6)	4196.8 (4038.8, 4485.4)	3687.2 (3608.7, 3764.3)
Qbert	16198.8 (14957.2, 17521.1)	10028.4 (7337.3, 12860.8)	16421.6 (14018.6, 18395.2)
Riverraid	19496.7 (18585.6, 20068.4)	16160.5 (14474.1, 17768.7)	14137.7 (13858.5, 14681.7)
Seaquest	17056.0 (16560.9, 17743.1)	7849.9 (3674.8, 12504.5)	12099.4 (10374.0, 15204.1)
SpaceInvaders	6891.9 (6055.9, 8374.6)	2459.6 (2111.7, 2726.2)	958.7 (841.1, 1033.2)

Table A.4: Per-game scores for worst-performing methods in Figure 2. Mean scores along with minimum and maximum scores, shown in brackets, across 3 runs.

Game	Rehearsal	DQfD	JSRL
Asterix	2329.6 (2172.0, 2503.9)	2370.8 (2140.9, 2681.0)	1294.2 (1018.2, 1720.3)
BeamRider	2740.4 (2262.7, 3123.5)	2482.9 (1596.4, 3246.7)	4642.0 (4272.0, 5324.4)
Bowling	33.5 (30.0, 36.0)	33.0 (30.0, 39.0)	37.8 (30.0, 46.1)
Breakout	70.2 (56.1, 82.2)	31.2 (21.2, 38.4)	49.7 (35.8, 59.2)
Enduro	985.0 (928.6, 1095.0)	736.6 (671.5, 792.4)	839.4 (660.7, 949.9)
MsPacman	2604.3 (2343.2, 2997.0)	2886.2 (2576.5, 3184.8)	2140.4 (1871.8, 2585.4)
Qbert	10649.0 (7697.8, 13882.9)	13386.7 (11800.7, 15295.5)	5828.7 (2871.6, 7753.4)
Riverraid	8640.5 (7789.1, 9532.1)	10290.7 (9722.8, 11323.5)	7498.6 (6916.7, 7916.8)
Seaquest	2859.2 (2300.9, 3969.2)	5325.5 (4454.8, 6128.3)	1140.8 (760.3, 1590.0)
SpaceInvaders	657.3 (638.7, 689.4)	533.3 (520.1, 557.5)	570.1 (549.9, 600.9)