VRL3: A Data-Driven Framework for Visual Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose VRL3, a powerful data-driven framework with a minimalist design for solving challenging visual deep reinforcement learning (DRL) tasks. We analyze a number of major obstacles in taking a data-driven approach, and present a suite of design principles, novel findings, and critical insights about data-driven visual DRL. Our framework has three stages: in stage 1, we leverage non-RL datasets (e.g. ImageNet) to learn task-agnostic visual representations; in stage 2, we use offline RL data (e.g. a limited number of expert demonstrations) to convert the task-agnostic representations into more powerful task-specific representations; in stage 3, we fine-tune the agent with online RL. On a set of challenging hand manipulation tasks with sparse reward and realistic visual inputs, compared to the previous SOTA, VRL3 achieves an average of 780% better sample efficiency. And on the hardest task, VRL3 is 1220% more sample efficient and solves the task with only 10% of the computation. These significant results clearly demonstrate the great potential of data-driven deep reinforcement learning.

1 Introduction

Over the past few years, the sample efficiency of Deep Reinforcement Learning (DRL) has significantly improved in popular benchmarks such as Gym [4, 80] and DeepMind Control Suite (DMC) [78]. However, the environments in these benchmarks often look different from the real world (see discussion in section 5.8, page 8 in [71]). It remains unclear whether the sample-efficient methodologies developed for these benchmarks can be successfully extended to more practical tasks that rely on realistic visual inputs from cameras and sensors, such as in robotic control and autonomous driving.

A promising direction is to take a data-driven approach, that is, try to use all the data available that might contribute to the learning process. Indeed, in the past few decades, the most important advances in deep learning have been facilitated by the use of large amounts of data and computation [76, 45]. Such a data-driven approach is natural and mature for supervised learning, but not as much so for RL, where learning is most commonly performed in an online fashion.

Although recent advances in representation learning for RL and offline RL reveal the potential of such an approach, most existing offline RL methods focus on tasks with proprioceptive ("raw-state") input and not the challenging visual control tasks. There is yet to be a method that can fully utilize data from multiple different sources to boost learning efficiency. In this work, the goal is to develop a data-driven framework that fills this role. A minimalist approach is taken to make this framework as simple as possible while being sample efficient. As discussed in a number of critique papers [25, 26, 17], simplicity in algorithm design is preferred because the performance of RL algorithms can be heavily affected by hidden details in the code, and a simpler design often leads to cleaner analysis and better reproducibility.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

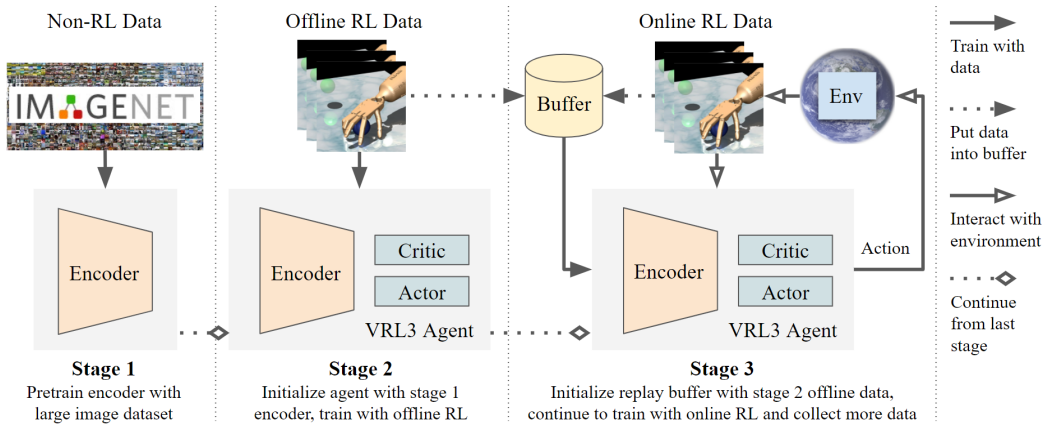


Figure 1: The design of VRL3. In stage 1, we pretrain an encoder using large image datasets to obtain task-agnostic visual representation. In stage 2, we initialize an actor-critic RL agent with the pretrained encoder, and then use offline RL techniques to train on offline RL data. Here we finetune the encoder and also learn the actor and critic. In stage 3, we initialize a buffer with the offline data from stage 2, and further train the entire agent with online RL. The idea is simple: to address challenging visual control tasks, the goal is to fully exploit all the available data.

In later sections, we show that our design of the framework is simple, intuitive, and effective. We also make an effort to maximize robustness and reproducibility, by comparing to stronger baselines that we re-implemented and improved, presenting extensive ablations and a hyperparameter robustness study, listing full technical details, providing open source code, and more.

Figure 1 shows the proposed framework, Visual Deep Reinforcement Learning in 3 Stages (VRL3). VRL3 is designed with a data-driven mindset, and in a minimalist fashion, Each stage is designed to utilize a particular type of data source.

In stage 1, we learn from large, existing non-RL datasets such as the ImageNet dataset. These datasets might not be directly relevant to the RL task, but they provide task-agnostic visual representations that might be helpful, especially if the RL task also has realistic visuals. A major challenge here is that the non-RL data can come from a different domain with different input shapes. Special care is required to make sure the representations obtained here can be smoothly transferred to the RL task.

In stage 2, we learn from offline RL data, such as a small number of expert demonstrations, or data collected by a previous learning agent. This offline data is related to the RL task and contains important information about solving the task. We initialize an actor-critic RL agent with the pretrained encoder and then employ offline RL techniques which not only train the actor-critic RL agent but also fine-tune the encoder, thereby creating a task-specific representation.

In stage 3, we learn from online RL data. Our main goal is sample efficiency (achieve high performance with as little online data as possible). We, therefore, use off-policy training, which uses and re-uses all available data during training. Note that naively transitioning into standard online RL can lead to training instability and destroy what we learned from the previous stages. We avoid this instability by constraining the maximum Q target value during off-policy updates.

Although each stage brings a number of unique challenges, we show that VRL3 can address all of them and take full advantage of the different data sources. Our contributions are as follows:

1. Novel framework: we present a data-driven framework that can fully exploit non-RL, offline RL, and online RL data to solve challenging tasks with sparse reward and realistic visual input. Most existing visual methods do not utilize offline data, and most offline RL methods do not work with difficult visual tasks, making our framework a novel contribution. We provide source code (in supplementary materials) and a full set of technical details to maximize reproducibility.
2. Novel technical contributions: we propose convolutional channel expansion (CCE) to enable an ImageNet-pretrained encoder to work with RL tasks that take in multiple input images. We also propose the Safe Q target technique for minimal-effort offline to online transition.

3. Novel insights: we show that offline RL data can provide a benefit that cannot be replaced by other stages, and that conservative RL might be a superior option when exploiting offline data, compared to contrastive learning. We also show that a reduced encoder learning rate helps learning in a multi-stage setting. And interestingly, representation pretrained with non-RL data, despite the large domain gap, can give better performance compared to representation learned from limited RL data, but is outperformed when RL data is ample. We also find stage 1 pretraining helps more in more difficult tasks.
4. Novel results: We present extensive ablations that reveal the effect of each component of VRL3, and conduct a large hyperparameter study on tasks with different difficulties, providing useful guidelines on how to deploy and tune our framework to new tasks.
5. Significant improvement over the previous SOTA: VRL3 achieves a new level of SOTA sample efficiency (780% better), parameter efficiency (3 times better), and computation efficiency (10 times faster to solve hardest task) on the challenging Adroit benchmark while also being competitive on DMC.

2 Related Work

VRL3 follows a minimalist design and effectively combines representation learning, offline RL, and online RL. We now discuss related work and how our contribution is novel and different from prior works, with a focus on model-free methods. Additional related work is discussed in Appendix D.

2.1 Learning visual tasks with online RL

With the help of data augmentation, many visual RL tasks can be learned in an online fashion. Typically, agents trained with such methods use convolutional encoders to extract lower-dimensional features from the visual input, and train in an end-to-end fashion [33, 91, 41, 92]. Another class of methods uses contrastive representation learning to solve visual tasks [42, 75, 97]. Contrastive learning can be used to learn useful representations without gradients from the RL objectives. These methods work well in environments with simple visuals, such as in the DMC environments, but can fail entirely if naively applied to challenging control tasks with sparse rewards and more realistic visuals such as Adroit, due to the difficulty in both exploration [64] and learning representations for realistic visual inputs [71]. Different from these works, VRL3 can fully exploit offline RL data and even non-RL data and can achieve SOTA performance in both Adroit and DMC.

2.2 RL with ImageNet pretraining

A number of previous works have used a pretrained encoder, such as a ResNet trained on ImageNet to help convert visual input into low-dimensional features, the encoder can be further finetuned or kept frozen during training [71, 16, 29, 21, 46, 62, 70]. However, such pretraining alone might not be sufficient to achieve the best performance or computation efficiency. Naively initializing the agent with an ImageNet-pretrained encoder might not provide a significant advantage over training from scratch [29]. Most of these works do not incorporate offline RL data. RRL [71] uses offline RL data but does not exploit them to the full extent. Different from these works, we show how ImageNet pretraining can be best combined with offline and online RL and provide a series of novel analyses, interesting results, and important insights. Other pretraining settings are also explored, [88] show masked image pretraining can lead to strong performance on a new suite of robotic tasks. [60] and [57] show that pretraining on different datasets and with more sophisticated methods can be combined with imitation learning. Compared to them, we study RL, which is a different subject, and we also achieve better success rate on Adroit (95%, while they have 85% and <70%, respectively).

2.3 Offline RL and data-driven RL

In the past few decades, some of the most important advances in AI research have been based on data-driven methods, so it is natural to ask whether we can also utilize a similar mindset for DRL [45, 76]. Although off-policy algorithms do not naively work in offline settings [18, 47], many specialized algorithms are proposed to allow effective offline training [18, 35, 8, 61, 86, 34, 17, 85, 32, 90, 2, 20, 87]. Recent works also study the transition from offline to online [56, 44, 100, 44, 32, 47]. However, most of these works are tested on tasks with proprioceptive (positions & velocities) input, and they

do not consider pretraining from non-RL data. Our work is different and is focused on challenging control tasks with visual input, sparse reward, and limited demonstrations.

Note that Adroit has often been studied with proprioceptive and not the more challenging visual input. RRL [71] and FERM [96] are the most relevant to our work. RRL uses an ImageNet-pretrained encoder, performs behavioral cloning (BC) on offline RL data, and then performs on-policy learning. Prior to our work, RRL is the SOTA on visual Adroit and is the only method that achieves non-trivial performance on the hardest Relocate task. FERM performs contrastive learning on offline RL data and then performs off-policy online RL with data augmentation. FERM can learn simple robotic tasks quickly but has brittle performance in Adroit [71].

In most prior works, the different training stages have been **studied separately**. What is missing is a framework that can **combine all 3 stages effectively and efficiently**. A major novel contribution of this work is such a framework and a comprehensive study on how its components interact and build up towards superior performance.

3 VRL3: Visual DRL in 3 stages

We now present the design of our framework VRL3. We focus on the high-level ideas and provide additional technical details in Appendix A and pseudocode in appendix C. Our agent has a convolutional encoder, two Q networks, two target Q networks and a policy network. Note the encoder can be trained separately from the policy and Q networks.

Stage 1 We follow the standard supervised learning routine and train a convolutional encoder f_ξ on a 1000-class ImageNet classification task. We follow the standard training procedure of a typical ResNet model. During training, we shrink the images to the size of 84x84 so that they match the input dimension of our downstream RL task. In our main results, we also use a lightweight encoder with 5 convolutional layers, with batch norm after each layer. In appendix A.7 we show additional results with deeper ResNet models.

Transition to multi-frame RL input After stage 1, we expand the first convolutional layer of the encoder so the input channel size matches the RL task which can take in multiple frames, and the weight values in this layer are divided accordingly to maintain features to later layers, we refer to this as **convolutional channel expansion (CCE)** (Further discussed in Section 5).

Stage 2 We first initialize an RL agent with the pretrained encoder and put the offline RL data in an empty replay buffer $\mathcal{D}$. Note that for Adroit, we have a standard 25 expert demonstrations per task (collected by human users with VR) [64]. We build our implementation on top of DrQv2 because it is a simple off-policy actor-critic algorithm with SOTA performance and a clean and efficient codebase [91]. Here we also try to follow the notations in DrQv2 for better consistency.

Input and image augmentation Let aug denote random shift image augmentation, and cat for concatenation. Let $\boldsymbol{x}$ be the visual input from the camera (or cameras). In the Adroit environment, in addition to the visual observation, we are also given sensor values from the robotic hand, which is a vector of real numbers, denoted by $\boldsymbol{z}$. The augmented visual input (image frames) is put into the encoder, giving a lower-dimensional representation $\boldsymbol{h} = f_\xi(\mathrm{aug}(\boldsymbol{x}))$. We then concatenate it with the sensor values (this is skipped in DMC). For simplicity, let $\boldsymbol{s} = \mathrm{cat}(\boldsymbol{h}, \boldsymbol{z})$ denote the concatenation; $\boldsymbol{s}$ is the input to the Q and policy networks, which are multi-layer perceptrons.

Offline RL update For each update, we sample a mini-batch of transitions $(\boldsymbol{x}_t, \boldsymbol{z}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{x}_{t+n}, \boldsymbol{z}_{t+n})$ from the replay buffer $\mathcal{D}$. Note we have $t+n$ here to compute n-step returns [91]. Let $\boldsymbol{s}_t = \mathrm{cat}(\boldsymbol{h}_t, \boldsymbol{z}_t)$. Let $\tilde{\boldsymbol{a}}_t \sim \pi_\phi(\boldsymbol{s}_t)$ be actions sampled from the current policy. The Q target value y is: $y = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \min_{k=1,2} Q_{\bar{\theta}_k}(\boldsymbol{s}_t, \tilde{\boldsymbol{a}}_{t+n})$.

The standard Q loss $\mathcal{L}_Q$ is (for $k = 1, 2$): $\mathcal{L}_Q(\mathcal{D}) = \mathbb{E}_{\tau\sim\mathcal{D}}\big[(Q_{\theta_k}(\boldsymbol{h}_t, \boldsymbol{a}_t) - y)^2\big]$.

Following [35], and build on top of our framework, we use an additional conservative Q loss $\mathcal{L}_{QC}$ term: $\mathcal{L}_{QC}(\mathcal{D}) = \mathbb{E}_{\tau\sim\mathcal{D}}\big[\log \sum_{\tilde{\boldsymbol{a}}_t} \exp(Q(\boldsymbol{s}_t, \tilde{\boldsymbol{a}}_t)) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t)\big]$. On a high-level, this loss essentially reduces the Q value for actions proposed by the current policy, and increase the Q values for actions in the replay buffer [35].

The standard policy (actor) loss $\mathcal{L}_\pi$ is: $\mathcal{L}_\pi(\mathcal{D}) = -\mathbb{E}_{\tau\sim\mathcal{D}}\big[\min_{k=1,2} Q_{\theta_k}(\boldsymbol{s}_t, \boldsymbol{a}_t)\big]$.

Let α be the learning rate for the policy network and the Q networks. Let β_{enc} be the encoder learning rate scale, so that the encoder learning rate is $\alpha_{\text{enc}} = \beta_{\text{enc}}\alpha$. For each offline RL update in stage 2, we perform gradient descent on $\mathcal{L}_Q(\mathcal{D}) + \mathcal{L}_{QC}(\mathcal{D})$ to update the encoder f_ξ and the Q networks $Q_{\theta_1}, Q_{\theta_2}$, and gradient descent on $\mathcal{L}_\pi$ to update the policy π_ϕ. Note that we do not use the gradient from the policy to update the encoder, following previous work [92, 91]. We also use Polyak averaging hyperparameter τ to update target networks, as is typically done.

Safe Q target technique When computing the Q target in stages 2 and 3, we propose a simple yet effective solution to prevent potential Q divergence due to training instability and distribution shift. For all tasks, we set a maximum Q target value. To decide this threshold value, we reshape the maximum per-step reward $r_{\max}$ to be 1 and estimate what the maximum Q value should be for an optimal policy. To be concrete, we can compute the sum of this infinite geometric series, $Q_{\max} = \sum_{t=0}^{\infty} \gamma^t r_{\max} = 100$ when $\gamma = 0.99$ (γ is discount factor). Empirical results show that this computed threshold is robust and does not require tuning (as shown in Figure 1e in the appendix A, slightly lower or higher threshold values such as 50 and 200 also give similar performance). Now if during any Q update, a Q target value y exceeds this threshold value, then we reduce it to be closer to the threshold value: if $y > Q_{max} + 1$, then $y \leftarrow Q_{\max} + (y - Q_{\max})^\eta$, where $0 \leq \eta \leq 1$ is the safe Q factor. Notice the $Q_{\max} + 1$ here is a technical detail to address edge cases.

Reduced encoder learning rate Due to the domain gap between ImageNet and the RL task, we apply a reduced encoder learning rate to avoid the disturbance of pretrained features in stages 2 and 3.

Stage 3 We continue to use the replay buffer $\mathcal{D}$, which already contains the offline RL data. We now perform standard online RL, and newly collected data are added to $\mathcal{D}$. The update is the same as in stage 2, except we remove the conservative Q loss $\mathcal{L}_{QC}(D)$.

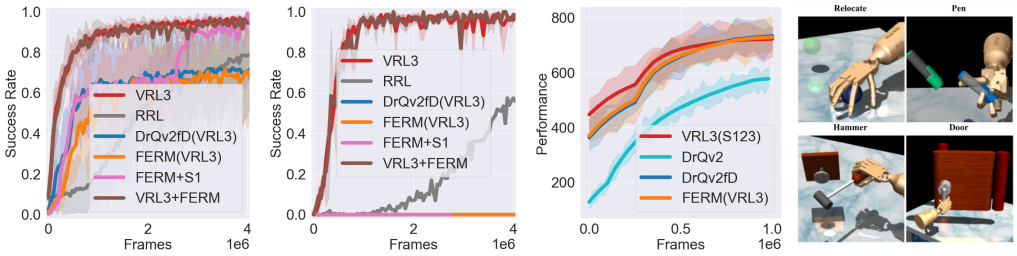
4 Results

Figure 2a shows success rate comparison averaged over all 4 Adroit tasks, Figure 2b shows results on the most challenging Relocate task, and Figure 2c shows average performance comparison on all 24 DMC tasks. We use the following methods as our baselines. DrQv2 is an off-policy online RL method that has the SOTA performance on DMC. RRL is an on-policy method that uses pretrained ImageNet features and utilizes demonstrations with behavioral cloning and has SOTA performance on Adroit. FERM is a 2-stage method that first trains an encoder with contrastive learning on offline RL data and then perform online RL. Note that currently there are not many successful multi-stage visual RL frameworks like VRL3, and as discussed in section 2, most existing offline-online methods are not designed to work with the more challenging visual inputs, so they are not included.

Getting stronger baselines Naively, DrQv2 does not use offline data and does not work well in Adroit due to the difficulty in exploration. And FERM has brittle performance on Adroit [71]. To create stronger baselines, we introduce variants of DrQv2 and FERM, which are both built on top of our framework. Specifically, DrQv2fD(VRL3) does not have stage 2 training but starts stage 3 training with demonstrations in the buffer. FERM(VRL3) is our re-implementation based on VRL3 and has much stronger performance compared to the results in [71] (more robust, is about 2 times faster, and reaches a 1.25x average success rate). For Adroit, we also compare with FERM+S1, which is FERM(VRL3) plus S1 pretraining, and VRL3+FERM, which is VRL3 with additional contrastive encoder learning in stage 2. For these improved baselines, we use the hyperparameters we searched for VRL3 for maximum performance. Doing so helps to ensure our comparison actually reflects the differences in algorithmic design and not hidden implementation details [17, 25].

Figure 2a shows that interestingly, for Adroit, both FERM(VRL3) and DrQv2fD(VRL3) outperform RRL in the early stage, showing that with the right implementation, these off-policy agents can learn quickly even without stage 1 representations. Figure 2b shows that on the most challenging Relocate task, DrQv2fD and FERM variants fail to learn entirely because they do not exploit stage 2 data sufficiently. VRL3 greatly outperforms all other methods.

Significantly stronger SOTA sample efficiency on Adroit: Compared to RRL, VRL3 achieves an average of 780% better sample efficiency over all tasks and is 1220% faster to solve the most difficult Relocate task (in terms of the data required to reach 90% success rate). VRL3+FERM has the same performance as VRL3, showing that additional contrastive learning is not necessary. FERM+S1 is stronger than FERM, showing that stage 1 pretraining also works well with contrastive stage 2



(a) 4 Adroit tasks (b) Relocate (c) 24 DMC tasks (d) Adroit Suite [64]

Figure 2: (a) Average success rate over 4 visual Adroit tasks. The performance for VRL3 is averaged over 10 seeds (seeds 0-9). To ensure consistency with prior work, results for RRL (Adroit SOTA) are provided by the authors. VRL3 achieves the strongest performance with a minimalist design. (b) Success rate for the hardest Relocate task. (c) Average performance over all 24 DMC tasks. VRL3 is still the best, though the performance gap is smaller. Results for DrQv2 are provided by the authors.

learning. For parameter and computation efficiency, we also use a much smaller encoder compared to RRL. We are 50 times more parameter efficient in the encoder and 3 times more parameter efficient for all networks in the agent. VRL3 is also more computationally efficient, in fact, VRL3 can solve the hardest task with just 10% of the computation (Detailed comparison in Appendix B).

For DMC, we collect 25K data with fully trained DrQv2 agents to enable stage 2 training. As shown in Figure 2c, VRL3 has the best performance in the early stage, FERM(VRL3) and DrQv2fD are slightly weaker, but achieve the same performance at 1M data. All three of them outperform DrQv2 (results from authors). Note that DMC is typically treated as an online RL benchmark, and DrQv2 is an online method, so we do not claim a new SOTA. These results are only to show that VRL3 can work in this popular benchmark and can utilize offline data to learn faster. Compared to DMC, Adroit is more challenging due to its sparse reward setting and more complex, more realistic visuals with more depth, lighting, and texture information. This might explain the larger performance gap in Adroit. This shows the advantage of VRL3 is more evident in more challenging visual tasks.

5 Challenges, Design Decisions, and Insights

In addition to the strong performance on Adroit (compared to the previous SOTA, an average of 780% better sample efficiency, 10 times faster in wall-clock time to solve the hardest task), another advantage of VRL3 is its simplicity. The minimalist design of VRL3 not only makes it much faster and more lightweight than the previous SOTA but also allows us to easily ablate and study its core components. In this section, we discuss some technical challenges and provide a comprehensive analysis of how the different components of VRL3 work together towards superior performance. We focus on Adroit and all experiment results are averaged over 4 Adroit tasks (per-task plots in Appendix A).

5.1 Stage 1: Pretraining with non-RL data

The first challenge we face in stage 1 is input shape mismatch. In stage 1, the input to the encoder is a single image with RGB channels. For the downstream RL task, however, it can be beneficial for the agent to take in multiple consecutive frames of images as input, or frames from multiple different cameras (sometimes even a combination of both), so that the agent can learn to use temporal and spatial information in these frames for better performance.

A naive solution is to put each frame separately into the pretrained encoder, as is done in [71]. This naive approach has two downsides: 1) it requires more computation, and 2) even if we finetune the encoder, it cannot learn important temporal and spatial features from the interaction of frames.

Advantage of convolutional channel expansion (CCE): We propose the CCE technique to address the above issue. Let m be the number of input frames for the RL task, With CCE, we expand the first convolutional layer of the stage 1 encoder so that it can now take in inputs of $3m$ channels instead of 3. We repeat the weight matrix m times and then scale all the weight values to $1/m$ of their original

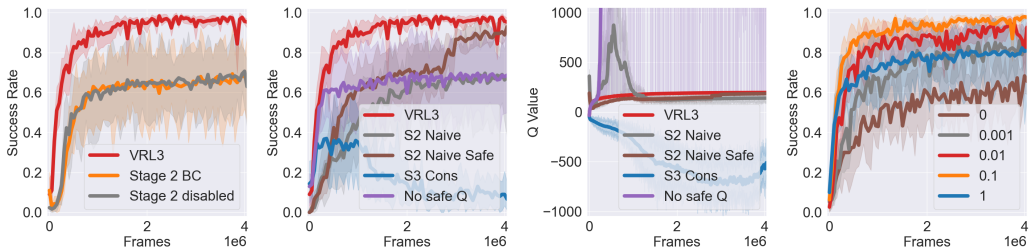
values. This allows us to modify the input shape without disrupting the learned representations. Such a simple method (can be done in one line of code, with no additional computation overhead, ablation study against 4 other variants in Appendix A.9) can effectively solve the mismatch problem when transitioning a visual encoder from the non-RL pretraining stage to the RL stages.

To enable faster experimentation and avoid difficulties in training deep networks in DRL [3, 59], we use a lightweight encoder with 5 convolutional layers, which is 50 times smaller than the ResNet34 used in RRL. Quite surprisingly, VRL3 can achieve superior performance with such a tiny encoder.

5.2 Stage 2 and Stage 3

In stage 2, why use conservative RL updates instead of BC or contrastive representation learning? We use conservative RL for the following reasons: a) we can finetune the encoder to get task-specific representations with RL signal b) comparing with contrastive learning methods which only train the encoder, we can train the encoder as well as the policy and Q networks, giving a jump-start in stage 3. c) With the safe Q technique, this also brings us a minimal-effort design that allows us to transition effectively from stage 2 to stage 3. We can simply remove the conservative term without any elaborate modifications. d) this fits well with an off-policy backbone, allowing maximum data reuse. (BC such as in [64] only works with on-policy methods). Next, we discuss extensive empirical evidence that supports our design.

Figure 3a shows that for VRL3, only using BC on expert demonstrations in stage 2 does not give a good performance. In fact, using BC in stage 2 gives a similar performance to when stage 2 is entirely disabled. Figure 3a shows that for VRL3, only using BC on expert demonstrations in stage 2 leads to the same performance compared to when stage 2 is entirely disabled. Additional ablation can be found in Appendix A.8. This problem is caused by a distribution mismatch of actor and critic at the beginning of stage 3 since BC only trains the actor and not the critic. For contrastive learning, we observe from Figure 2 that FERM has slightly worse performance in DMC and much worse performance in Adroit. This is because Adroit has a more challenging sparse reward setting and FERM fails to fully exploit offline RL data by only updating the encoder. These results show that when tackling challenging tasks, **learning only the representation in stage 2 can be useful, but is not enough to fully exploit offline RL data**. This is a novel and important finding that has never been carefully studied or emphasized in the literature.



(a) Stage 2 BC or disable (b) S2-S3 (c) S2-S3 Q value (d) Encoder lr

Figure 3: a) Effect of BC and offline RL updates in stage 2. Only applying BC or disabling stage 2 training entirely leads to poor performance. b) and c) Taking naive RL updates in stage 2 (S2 Naive) leads to severe overestimation, which can be mitigated by safe Q (S2 Naive Safe). Taking conservative updates in stage 3 (S3 Cons) leads to underestimation. VRL3 gives the best performance with conservative update only in stage 2 and safe Q. Note in our setting, the maximum reasonable Q value should be around 100. d) Effect of different encoder learning rates (e.g., 0.01 means the encoder's learning rate is 100 times slower than that for the policy and Q networks.)

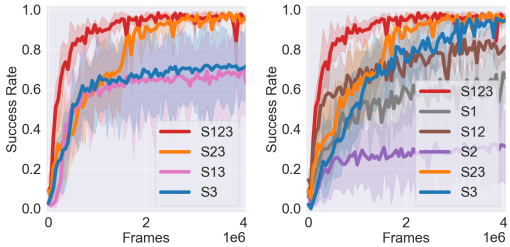
Effect of safe Q target for stage 2-3 transition: Special care is required to effectively perform off-policy learning in stage 2. In VRL3, we perform conservative updates in stage 2 and remove the conservative term in stage 3. When we transition from stage 2 to stage 3, the new online data will cause a distribution shift that can lead to instability in Q values [56], The safe Q target technique is a simple, efficient, and minimalist solution to this issue. Figure 3b and 3c show the effect of these design decisions: updating with naive RL in stage 2 (S2 Naive) or removing the safe Q technique (No safe Q) lead to severe overestimation in stage 3 and cause performance drop. Updating with

conservative RL in stage 3 (S3 Cons) leads to severe underestimation and even worse performance. Naive RL in stage 2 with safe Q in stages 2 and 3 (S2 Naive Safe) can mitigate overestimation but is still weaker than VRL3, which takes conservative updates in stage 2 and use safe Q in stages 2 and 3.

Reduced encoder learning rate for better finetuning: When we fine-tune the encoder in stages 2 and 3, a reduced encoder learning rate can prevent pretrained representations from getting destroyed by noisy RL signals, especially in the early stage of RL training. This problem has also been observed in other environments [67]. Figure 3d shows the effect of encoder learning rates. These results show that even though there is a large domain gap between ImageNet and RL tasks, a frozen stage 1 encoder can still provide useful features, and finetuning the encoder can further improve performance. Although it is not new that transfer + finetuning can give the best result [93], it has not been adequately studied in the RL setting with large domain gap and noisy signals.

5.3 Contribution of each stage

VRL3 achieves strong performance by fully exploiting data from 3 different stages. We now try to understand how much each stage contributes to the final performance. We perform 2 sets of comparisons, for each VRL3 variant in Figure 4a, we disable one or more stages of training entirely (e.g. S13 means stage 2 training is removed), and for Figure 4b, we disable the *encoder training* in one or more stages, but always maintain the updates to actor and critic networks (e.g. S12 means encoder is frozen in stage 3).



(a) Stage ablation (b) Encoder ablation

Figure 4: (a) Effect of each training stage. The numbers after "S" indicates the stages that are enabled. (b) Effect of enabling encoder training in each stage.

Here we summarize a number of insights we can obtain from these results. In Figure 4a: Stage 1 pretraining improves sample efficiency (S123 > S23); **stage 2 is critical and its contribution cannot be replaced by the other stages** (S123> S13, S23 > S13). Note that since Adroit is challenging with limited offline data, S3 is always enabled to obtain non-trivial performance.

In Figure 4b: when the encoder is only trained in one stage, stage 3 encoder training gives a much stronger performance than the other two, showing the importance of task-specific representations (S3 > S1 and S2). Note that interestingly, **although task-specific features are more useful, when RL data is scarce, non-RL pretraining provides better results despite the large domain gap** (S3> S1 > S2). Training the encoder in more stages always leads to improved performance (S123 > S12 > S1).

An important finding is that our results indicate **offline RL is more beneficial than just contrastive representation learning in stage 2**, note that S3 in Figure 4b (perform offline RL update but freeze encoder in stage 2, encoder trained in stage 3) has much stronger performance than S3 in Figure 4a (no stage 2 training at all). This is consistent with results in Figure 2, the performance gap between VRL3 and FERM+S1 is caused by VRL3 doing offline RL updates in stage 2, and thus learning a full agent, instead of just the encoder features.

With the above insights, Table 1 summarizes fundamental differences between VRL3 and some other popular visual RL methods.

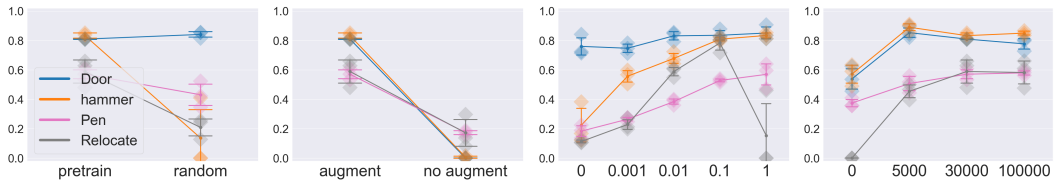
5.4 Additional Studies and Analysis

We also perform an extensive study on hyperparameter robustness and sensitivity on each of the 4 Adroit tasks (with varying difficulty). Due to limited space, we only show a small portion of our

Table 1: Comparison of core design decisions. VRL3 fully utilizes non-RL, offline RL and online RL datasets. DA refers to data augmentation, Con for contrastive learning, Offline for offline RL. Note that offline RL data can be used to learn representation (encoder), or the task (actor, critic), or both.

Characteristics	DrQ/RAD	CURL/ATC	FERM	RRL	VRL3
Stage 1	None	None	None	Pretrain	Pretrain
Stage 2	None	None	Con	BC	Offline
Stage 3	DA	Con	DA	No DA	DA
Leverage large visual datasets	✗	✗	✗	✓	✓
Task-specific representations	✓	✓	✓	✗	✓
Offline RL data for representations	✗	✗	✓	✗	✓
Offline RL data for task learning	✗	✗	✗	✓	✓
High data reuse rate (off-policy)	✓	✓	✓	✗	✓
Prevent Q network divergence	✗	✗	✗	✗	✓

study in Figure 5 and put the rest of the results in Appendix A. This provides additional insights and helps us understand what hyperparameters should be tuned with high priority when applied to a new task, for example, results show that S1 pretraining is useful for harder tasks but not for the simplest Door task; data augmentation is always critical; the encoder learning rate is important and should be tuned in difficult tasks; even a small number of S2 offline updates can greatly help learning. In total, we studied 16 different components and hyperparameters, and identified 9 of them as important to performance, out of these 9, we further show that 6 of them are robust, and 3 of them are sensitive (learning rate, action noise std, encoder learning rate scale). Note that the encoder learning rate scale is **the only important and sensitive hyperparameter introduced by VRL3**, while the other 2 are from the backbone algorithm. These important studies show that our framework is quite robust, we believe these results and insights will provide a better understanding on data-driven DRL methods, and we hope they will make it easier for other researchers to deploy this framework to other tasks.



(a) S1 pretrain (R) (b) S2,3 augmentation (R) (c) S2,3 encoder lr scale (S) (d) S2 num of update (R)

Figure 5: Part of the hyperparameter sensitivity study. For each figure, X-axis shows different hyperparameter values, Y-axis shows the average success rate in the first 1M training frames, over three seeds. Each dot is the average success rate of one seed, the error bar shows one standard deviation. (S) means critical and relatively sensitive. (R) means critical but is robust or easy to tune.

6 Discussion, Limitations and Future Work

In this paper, we propose a 3-stage data-driven framework for solving challenging visual control tasks. We perform extensive experimentation to ablate and analyze our framework to identify the most important technical components. We arrive at a minimalist design that is simple, intuitive, and much more sample efficient and computationally efficient than the previous SOTA on Adroit. Although this is a significant result, we do have some limitations: for example, Adroit has more realistic visuals compared to DMC, but it is still a simulated benchmark. Experiments on robotics hardware are required to understand how VRL3 should be deployed in the real world. Other limitations include: it remains unclear whether stage 1 training is still beneficial if the domain gap between non-RL data and RL data is too great (RL task visuals different from the real world). We also did not experiment extensively on model-based method. These can be exciting future research topics. Despite these limitations, VRL3 brings a number of novel and interesting insights, in addition to the strong empirical results. We believe VRL3 is an important step toward practical data-driven DRL for real world visual control tasks.

References

[1] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

[2] Chenjia Bai, Lingxiao Wang, Zhuoran Yang, Zhihong Deng, Animesh Garg, Peng Liu, and Zhaoran Wang. Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning. *arXiv preprint arXiv:2202.11566*, 2022.

[3] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning. *arXiv preprint arXiv:2106.01151*, 2021.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[7] Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. *International Conference on Learning Representations*, 2021.

[8] Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 18353–18363, 2020.

[9] Hyesong Choi, Hunsang Lee, Wonil Song, Sangryul Jeon, Kwanghoon Sohn, and Dongbo Min. Self-supervised structured representations for deep reinforcement learning. 2021.

[10] Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. *arXiv preprint arXiv:2005.08068*, 2020.

[11] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR, 2021.

[12] Yuchen Cui, Scott Niekum, Abhinav Gupta, Vikash Kumar, and Aravind Rajeswaran. Can foundation models perform zero-shot task specification for robot manipulation? In *Learning for Dynamics and Control Conference*, pages 893–905. PMLR, 2022.

[13] Kefan Dong, Yuping Luo, Tianhe Yu, Chelsea Finn, and Tengyu Ma. On the expressivity of neural networks for deep reinforcement learning. *International Conference on Machine Learning (ICML)*, 2020.

[14] Danny Driess, Ingmar Schubert, Pete Florence, Yunzhu Li, and Marc Toussaint. Reinforcement learning with neural radiance fields. *arXiv preprint arXiv:2206.01634*, 2022.

[15] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[16] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[17] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[18] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[19] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[20] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[21] Abhinav Gupta, Adithyavairavan Murali, Dhiraj Prakashchand Gandhi, and Lerrel Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. *Advances in neural information processing systems*, 31, 2018.

[22] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[23] Siddhant Haldar, Vaibhav Mathur, Denis Yarats, and Lerrel Pinto. Watch and match: Supercharging imitation with regularized optimal transport. *arXiv preprint arXiv:2206.15469*, 2022.

[24] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[25] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[26] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[27] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12519–12530, 2019.

[28] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[29] Ryan Julian, Benjamin Swanson, Gaurav S Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. *arXiv preprint arXiv:2004.10190*, 2020.

[30] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[33] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[34] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[35] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[36] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[37] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. *arXiv preprint arXiv:2005.04269*, 2020.

[38] Hang Lai, Jian Shen, Weinan Zhang, and Yong Yu. Bidirectional model-based policy optimization. *arXiv preprint arXiv:2007.01995*, 2020.

[39] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

[40] Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[41] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[42] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[43] Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. *arXiv preprint arXiv:2007.04938*, 2020.

[44] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022.

[45] Sergey Levine. Understanding the world through action. In *Conference on Robot Learning*, pages 1752–1757. PMLR, 2022.

[46] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[47] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[48] Xiang Li, Jinghuan Shang, Srijan Das, and Michael S Ryoo. Does self-supervised learning really improve reinforcement learning from pixels? *arXiv preprint arXiv:2206.05266*, 2022.

[49] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[50] Nan Lin, Yuxuan Li, Keke Tang, Yujun Zhu, Xiayu Zhang, Ruolin Wang, Jianmin Ji, Xiaoping Chen, and Xinming Zhang. Manipulation planning from demonstration via goal-conditioned prior action primitive decomposition and alignment. *IEEE Robotics and Automation Letters*, 7(2):1387–1394, 2022.

[51] Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. *arXiv preprint arXiv:2204.09560*, 2022.

[52] Lingheng Meng, Rob Gorbet, and Dana Kulić. The effect of multi-step methods on overestimation in deep reinforcement learning. *arXiv preprint arXiv:2006.12692*, 2020.

[53] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[54] 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, 2015.

[55] Yao Mu, Shoufa Chen, Mingyu Ding, Jianyu Chen, Runjian Chen, and Ping Luo. Ctrlformer: Learning transferable state representation for visual control via transformer. *arXiv preprint arXiv:2206.08883*, 2022.

[56] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. 2020.

[57] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[58] Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning*, pages 16828–16847. PMLR, 2022.

[59] Kei Ota, Devesh K Jha, and Asako Kanezaki. Training larger networks for deep reinforcement learning. *arXiv preprint arXiv:2102.07920*, 2021.

[60] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. *arXiv preprint arXiv:2203.03580*, 2022.

[61] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[62] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[63] Senthil Purushwalkam Sh. *Visual Representation and Recognition without Human Supervision*. PhD thesis, Carnegie Mellon University, 2022.

[64] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[65] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning. *arXiv preprint arXiv:2004.07804*, 2020.

[66] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[67] Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, Devon Hjelm, Philip Bachman, and Aaron Courville. Pretraining representations for data-efficient reinforcement learning. *arXiv preprint arXiv:2106.04799*, 2021.

[68] Younggyo Seo, Danijar Hafner, Hao Liu, Fangchen Liu, Stephen James, Kimin Lee, and Pieter Abbeel. Masked world models for visual control. *arXiv preprint arXiv:2206.14244*, 2022.

[69] Younggyo Seo, Kimin Lee, Stephen L James, and Pieter Abbeel. Reinforcement learning with action-free pre-training from videos. In *International Conference on Machine Learning*, pages 19561–19579. PMLR, 2022.

[70] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*, 2016.

[71] Rutav Shah and Vikash Kumar. Rrl: Resnet as representation for reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*, 2021.

[72] Wenling Shang, Xiaofei Wang, Aravind Srinivas, Aravind Rajeswaran, Yang Gao, Pieter Abbeel, and Misha Laskin. Reinforcement learning with latent flow. *Advances in Neural Information Processing Systems*, 34:22171–22183, 2021.

[73] Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. Robotic telekinesis: learning a robotic hand imitator by watching humans on youtube. *arXiv preprint arXiv:2202.10448*, 2022.

[74] Maks Sorokin, Jie Tan, C Karen Liu, and Sehoon Ha. Learning to navigate sidewalks in outdoor environments. *IEEE Robotics and Automation Letters*, 7(2):3906–3913, 2022.

[75] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

[76] Richard S Sutton. The bitter lesson. `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`, 2021. Accessed: 2022-01-24.

[77] Allison C Tam, Neil C Rabinowitz, Andrew K Lampinen, Nicholas A Roy, Stephanie CY Chan, DJ Strouse, Jane X Wang, Andrea Banino, and Felix Hill. Semantic exploration from language abstractions and pretrained representations. *arXiv preprint arXiv:2204.05080*, 2022.

[78] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[79] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[80] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[81] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, et al. Jump-start reinforcement learning. *arXiv preprint arXiv:2204.02372*, 2022.

[82] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[83] Homer Walke, Jonathan Yang, Albert Yu, Aviral Kumar, Jedrzej Orbik, Avi Singh, and Sergey Levine. Don't start from scratch: Leveraging prior data to automate robotic reinforcement learning. *arXiv preprint arXiv:2207.04703*, 2022.

[84] Che Wang, Yanqiu Wu, Quan Vuong, and Keith Ross. Striving for simplicity and performance in off-policy drl: Output normalization and non-uniform sampling. In *International Conference on Machine Learning*, pages 10070–10080. PMLR, 2020.

[85] Qing Wang, Jiechao Xiong, Lei Han, Han Liu, Tong Zhang, et al. Exponentially weighted imitation learning for batched historical data. In *Advances in Neural Information Processing Systems*, pages 6288–6297, 2018.

[86] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[87] Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.

[88] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.

[89] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. *arXiv preprint arXiv:2102.05815*, 2021.

[90] Denis Yarats, David Brandfonbrener, Hao Liu, Michael Laskin, Pieter Abbeel, Alessandro Lazaric, and Lerrel Pinto. Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning. *arXiv preprint arXiv:2201.13425*, 2022.

[91] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[92] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

[93] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

[94] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[95] Zhecheng Yuan, Zhengrong Xue, Bo Yuan, Xueqian Wang, Yi Wu, Yang Gao, and Huazhe Xu. Pre-trained image encoder for generalizable visual reinforcement learning. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*.

[96] Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A framework for efficient robotic manipulation. *arXiv preprint arXiv:2012.07975*, 2020.

[97] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[98] Chi Zhang, Sanmukh Rao Kuppannagari, and Viktor K Prasanna. Maximum entropy model rollouts: Fast model based policy optimization without compounding errors. *arXiv preprint arXiv:2006.04802*, 2020.

[99] Qihang Zhang, Zhenghao Peng, and Bolei Zhou. Action-conditioned contrastive policy pretraining. *arXiv preprint arXiv:2204.02393*, 2022.

[100] Han Zheng, Xufang Luo, Xuan Song, Dongsheng Li, Jing Jiang, et al. Adaptive q-learning for interaction-limited reinforcement learning. 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] A discussion is provided in section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] A discussion on potential negative societal impacts is provided in Appendix D.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide source code in the supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We discuss major technical details in the main paper, with additional technical details in Appendix A and C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The plots are averaged over multiple runs (some are also over multiple environments), shaded area indicate the standard deviation across runs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] They are discussed in detail in Appendix B.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cited the authors of the code (DrQv2) and testing environments (DMC, Adroit, MuJoCo) we use.
   (b) Did you mention the license of the assets? [Yes] A discussion is provided in Appendix C.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our source code is provided in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Additional Results and Analysis

A.1 Hyperparameter Robustness, Sensitivity and Understanding

In this section we discuss implementation details and present an extensive ablation study on hyperparameters.

Figure 6 shows how each hyperparameter affect training performance. For each figure, the results show how learning is affected when we modify one hyperparameter from our set of default hyperparameters while keeping all the rest fixed as the default values. Following our 3-stage framework, we discuss hyperparameters that affect one or more stages of our training process, and we also give our default value for each hyperparameter. The default hyperparameters are found in our early experiments with random search.

The goal here is to achieve a thorough understanding of what components are the most important for VRL3 to work, and when VRL3 is applied to a new task, which ones of the hyperparameters are the most critical and should be finetuned first. We believe these insights can greatly help researchers and practitioners in applying our framework to new visual tasks, or build research on top of our work. Note that in our study, we actually find some of the hyperparameters to be not important since they do not affect the training performance. However, they are still discussed here for completeness.

A.1.1 Stage 1

Stage 1: whether we use stage 1 pretraining (default: True). If we perform stage 1 pretrainig, then we start with an encoder that is pretrained on the ImageNet dataset, otherwise we start with a random encoder. This will only affect the initialization of encoder for stage 2 and 3. Figure 6a shows that the door task actually does not rely on stage 1 features, but the more challenging relocate task benefit hugely from stage 1 pretraining. This observation is consistent with previous results that show more difficult tasks benefit more from pretrained features, and easier tasks can be learned without them [71]. This result shows that stage 1 pretraining is critical especially if we want to tackle challenging real-world control tasks.

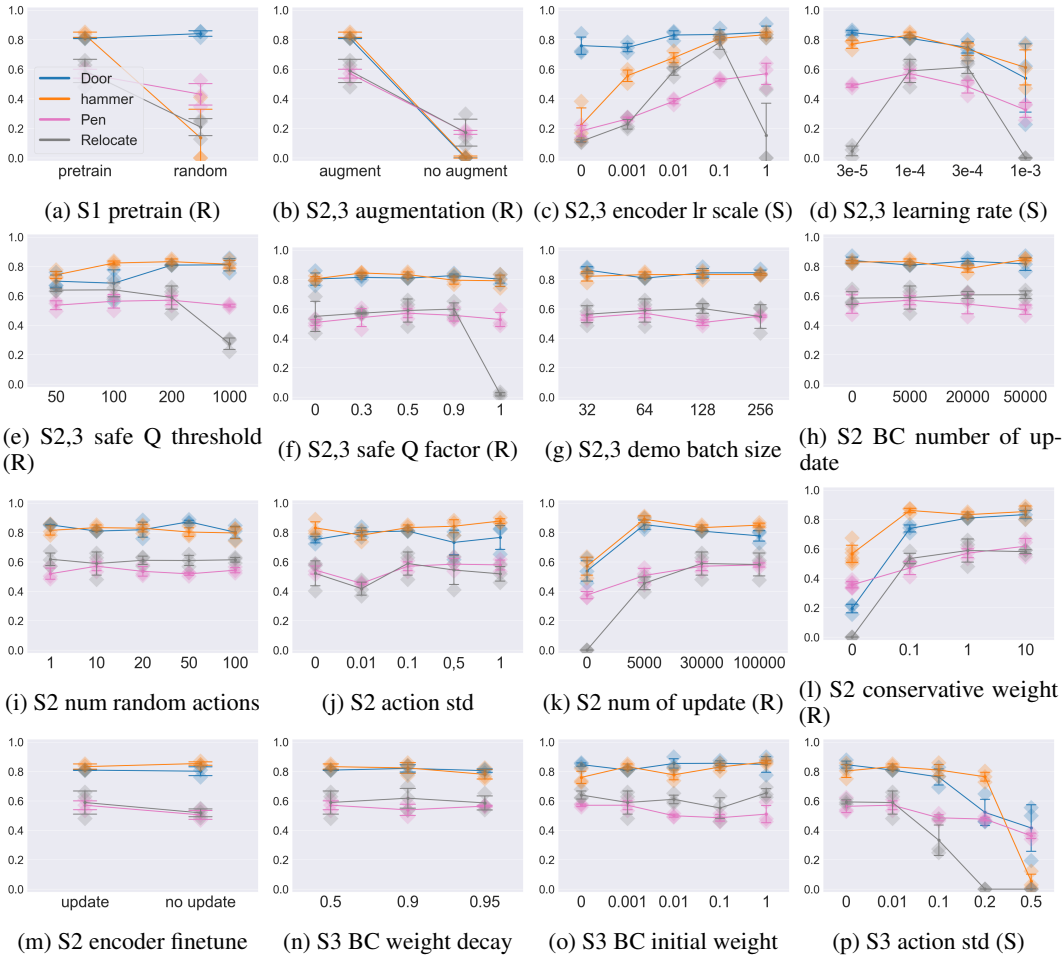
A.1.2 Hyperparameters that affect both stage 2 and 3

During our training in stage 2 and stage 3, we can also add an additional BC loss, which can help the agent learn more human-like behavior [64]. We do not include it in VRL3 since it does not have a significant impact on performance, but will discuss it in this ablation study.

Stage 2 and 3: whether we use data augmentation (default: True). Figure 6b shows that data augmentation is critical for both tasks. This result is consistent with previous literature that found data augmentation such as random shift can greatly improve performance in visual-based tasks [91, 92]. Although previous works mainly apply data augmentation to tasks with visuals that are different from the real-world such as DMControl[78], we show that it can also improves performance in tasks with more realistic visuals.

Stage 2 and 3: demonstration replay batch size (default: 64). In stage 2, before the offline RL updates, we perform a number of BC updates where we sample from a demonstration replay buffer. In stage 3, we also do BC with a decaying loss. Figure 6g shows that the batch size for computing BC loss is not important and it is also not needed to achieve the performance VRL3. By default, we use a small batch size of 64 to reduce computation time.

Stage 2 and 3: encoder learning rate scale (default: 0.01 for relocate, and 1 for the other tasks). Figure 6c shows that when encoder is fixed (learning rate scale is 0), then the performance becomes worse, especially for relocate. Note that when encoder learning rate is too high, learning can also become difficult. For relocate, we reduce its encoder learning rate to be 100 times slower than the learning rate for other parts of the agent, similar to what is done in [67]. This result, together with Figure 6a shows that when task-agnostic representation is finetuned towards task-specific representation learned directly from RL objective with data augmentation gives the best performance.



(a) S1 pretrain (R) (b) S2,3 augmentation (R) (c) S2,3 encoder lr scale (S) (d) S2,3 learning rate (S)

(e) S2,3 safe Q threshold (R) (f) S2,3 safe Q factor (R) (g) S2,3 demo batch size (h) S2 BC number of update

(i) S2 num random actions (j) S2 action std (k) S2 num of update (R) (l) S2 conservative weight (R)

(m) S2 encoder finetune (n) S3 BC weight decay (o) S3 BC initial weight (p) S3 action std (S)

Figure 6: Hyperparameter sensitivity study for all three stages, blue shows the door task, orange shows the more difficult relocate task. For each figure, caption shows the type of hyperparameter, X-axis shows different hyperparameter values, Y-axis shows the average success rate in the first 1M frames of training, averaged over three seeds. Each dot is the average success rate of one seed, error bar shows one standard deviation. We use (S) to denote critical hyperparameter that is relatively sensitive and should be tuned with high priority when applied to a new task. (R) denotes critical hyperparameter that is robust or easy to tune. Hyperparameters that are mentioned in the main paper are discussed in the appendix.

Stage 2 and 3: learning rate (default: $1e-4$). Figure 6d shows that (unsurprisingly), when learning rate becomes too small or too large, performance will drop.

Stage 2 and 3, safe Q target value threshold (default: 200): Figure 6e shows that having a reasonable Q value threshold greatly helps improve performance on relocate. As described in the main text, This threshold decides how large the Q value can become. Having this safe Q target value technique allows easy control excessive Q bias accumulation with minimal negative effect.

Stage 2 and 3, safe Q factor (default: 0.5): Figure 6f shows that having a safe Q factor that is < 1 (when it is 1, then we are not limiting how large the Q values can be) can help stabilize training and prevent performance issues caused by Q divergence.

A.1.3 Stage 2 hyperparameters

In stage 2, most of the hyperparameters here are about the offline updates. As mentioned in the previous section, we can have optional BC updates before the offline RL updates. For the offline RL updates, since we mainly use the idea of CQL, there are a number of CQL relevant hyperparameters.

Stage 2, number of BC update (default: 5000): Figure 6h shows that BC updates in stage 2 before taking offline RL updates in fact does not affect performance too much. In terms of performance, it is not necessary when offline RL is applied at stage 2. However, it can still help the agent learn more human-like behaviors, as discussed in [64].

Stage 2, CQL number of random actions (default: 10): in the author's implementation of CQL, a number of random actions are generated from a uniform disribution and these actions are used in the log-sum-exp computation along with actions proposed by the current policy. Though this is not really discussed in the CQL paper [35], the number of random actions might have an effect on offline training. Figure 6i shows that in our case, the number of random actions for CQL updates in stage 2 does not have a significant effect. However, we recommend future researchers to test this hyperparameter when applying VRL3 to new tasks, since the effect of this small detail is not fully addressed by the CQL authors.

Stage 2, CQL action std (default: 0.1): since our off-policy backbone is DrQv2, we no longer have the entropy term in SAC, but use a fixed action std during updates and during exploration. We also test the effect of this std in stage 2 offline RL updates. Figure 6j shows that in our case, the standard deviation for the action distribution during CQL updates do not have a significant effect on training. Note that however, as we will discuss in the next subsection, this action std has a significant effect in stage 3 online learning.

Stage 2, number of conservative updates (default: 30000): Figure 6k shows that when we do not use offline RL updates (0 updates), then BC updates in stage 2 alone cannot guarantee good performance in stage 3. Some of this difficulty has been studied in [56].

Stage 2, conservative term weight (default: 1): this weight decides how hard the Q values are being affected by the conservative loss term [35]. Figure 6l shows that when CQL weight is greater than 0, the performance is in general robust with different values.

Stage 2, allow encoder finetune in stage 2 (default: True): by default, in stage 2 we finetune encoder with Q loss. Does this stage 2 finetuning have a significant effect on performance? Figure 6m shows that this does not affect performance too much in our case, indicating that for the encoder, stage 1 pretraining plus stage 3 finetuning is enough to obtain good performance. This shows in stage 2 we can do more than just using stage 2 data to learn representations. We can get the best performance by fully exploiting stage 2 data to learn an offline RL agent as well as improved representations,

A.1.4 Stage 3 hyperparameters

Stage 3, BC weight decay (default: 0.5): this value decides how fast the BC loss is decayed, following [64]. Figure 6n shows that under our current design of VRL3, the decaying speed of BC loss does not affect performance too much.

Stage 3, BC initial weight (default: 0.001): The inital weight of the BC loss. Figure 6o shows that initial BC weight in stage 3 does not have a significant effect on performance. Again, we keep the BC loss here since it can also help the robotic hand learn human-like behaviors.

Stage 3, action distribution standard deviation for exploration (default: 0.01): Figure 6p shows that we need a small std value to obtain good performance. This is likely because the Adroit tasks have sparse reward, thus it is easier to have less action noise so that the agent focus on exploring state space that is close to the expert demonstrations. This is another important hyperparameter that will require tuning on a new task. Note that in some other benchmarks such as DMControl, the exploration action noise is typically much larger.

A.2 Hyperparameter Table

We now present a summary of hyperparameters, and discuss additional details. Since our backbone algorithm is DrQv2, we keep most of the original hyperparameters when applied (see hyperparameter table of DrQv2 in appendix B of [91]). Table 2 shows a summary of the important hyperparameters used in VRL3. As discussed in the previous section, some of the studied hyperparameters are shown to have no effect on performance, thus are removed from the table.

For stage 1 our training is the same as in a typical ResNet training setting, we use an encoder with 5 convolutional layers (batch norm after each conv layer), with 32 channels in the first conv layer, during training for the 3 stages, we always use input image of 84x84. For stage 2, we use 25K offline RL data for both Adroit (expert demonstrations provided by the authors of the benchmark) and DMC (data collected by trained DrQv2 agents). When we move to stage 3, we are different from the default hyperparameters of DrQv2 in: a) we use a reduced encoder learning rate scale for relocate (DrQv2 does not discuss this option so it uses a value of 1). b) We do not collect random data at the beginning of stage 3, as we already have some offline data in stage 2. c) we use a fixed action std of 0.01 for exploration, while DrQv2 uses a more complex exploration noise schedule.

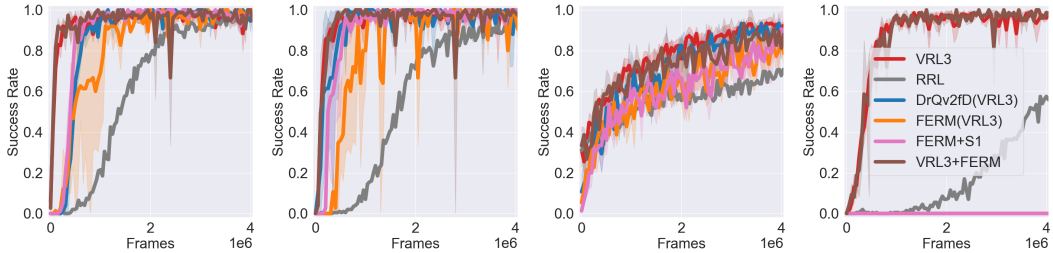
For the DMC tasks, we are largely the same as discussed in the DrQv2 paper [91], however, we use a fixed std of 0.1. We additionally found that adding a pretanh value penalty can sometimes help avoid exploration issue, so we have a pretanh penalty weight (discussed in Section A.5) of 0.001 and a penalty threshold of 5.

Table 2: VRL3 default hyperparameters for Adroit

Parameter	Value
Encoder	
image input size	84 x 84
n of channel in first conv layer	32
number of convolutional layers	5
batch norm	after each conv layer
Stage 2	
offline RL data	25,000
number of conservative updates	30,000
CQL number of random actions	10
CQL action std	0.1
CQL weight	1
Stage 2, 3	
optimizer	Adam [31]
discount (γ)	0.99
encoder lr scale	0.01 for relocate, 1 for others
learning rate	1e-4
safe Q target threshold	200
safe Q factor	0.5
target smoothing coefficient (ρ)	0.01
replay buffer size	10^6
mini-batch size	256
nonlinearity	ReLU
action repeat	2
n-step returns	3
agent update frequency	2
encoder feature dimension	50
Q and policy hidden layers	2
Q and policy hidden dimension	1024
Stage 3	
random starting data	0
action std for exploration	0.01

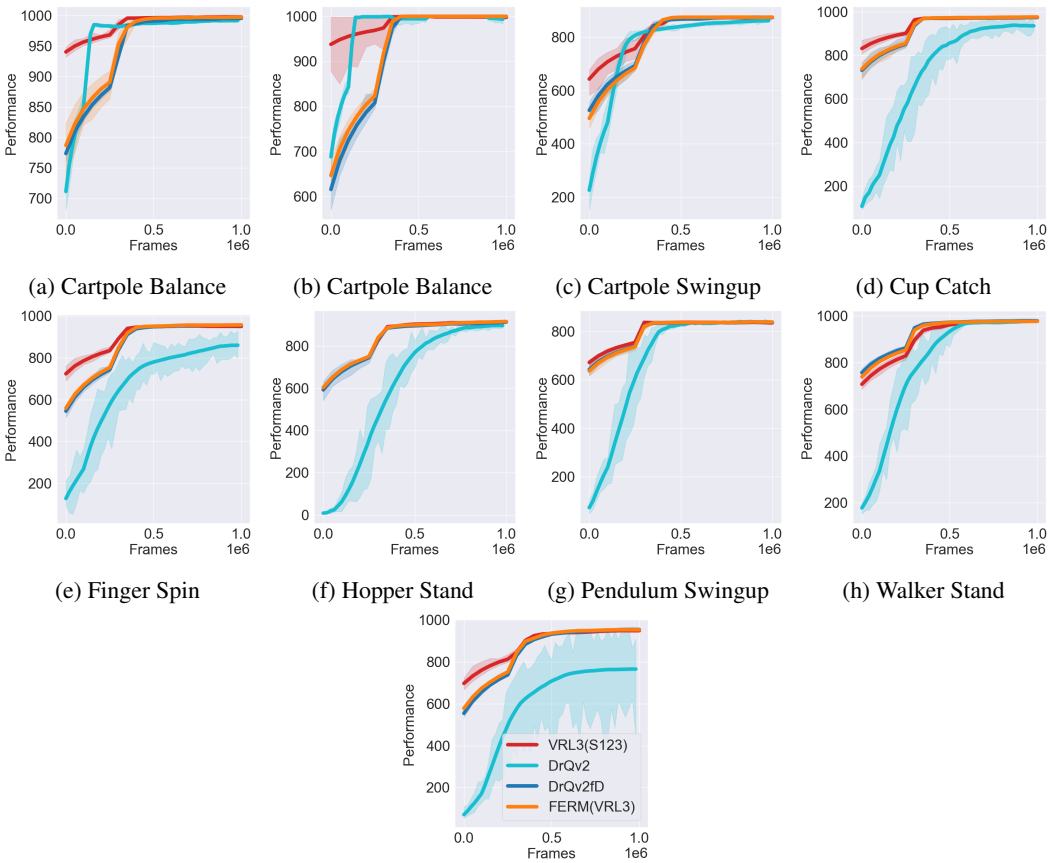
A.3 Additional Figures

Here we present per-task figures for results in the main paper, for each set of figures, additional discussions are provided in the caption.



(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 7: Per-task success rate comparison of VRL3 and other methods. Note that VRL3 is most effective on the hardest Relocate tasks (Prior to VRL3, RRL is the only method that achieves non-trivial performance on Relocate, but takes more than 12M frames to reach 90% success rate, while VRL3 solves it in 1M frames), but also significantly outperform other variants on Door and Hammer. VRL3 outperforms RRL on Pen, but is similar to the other competitors in sample efficiency.

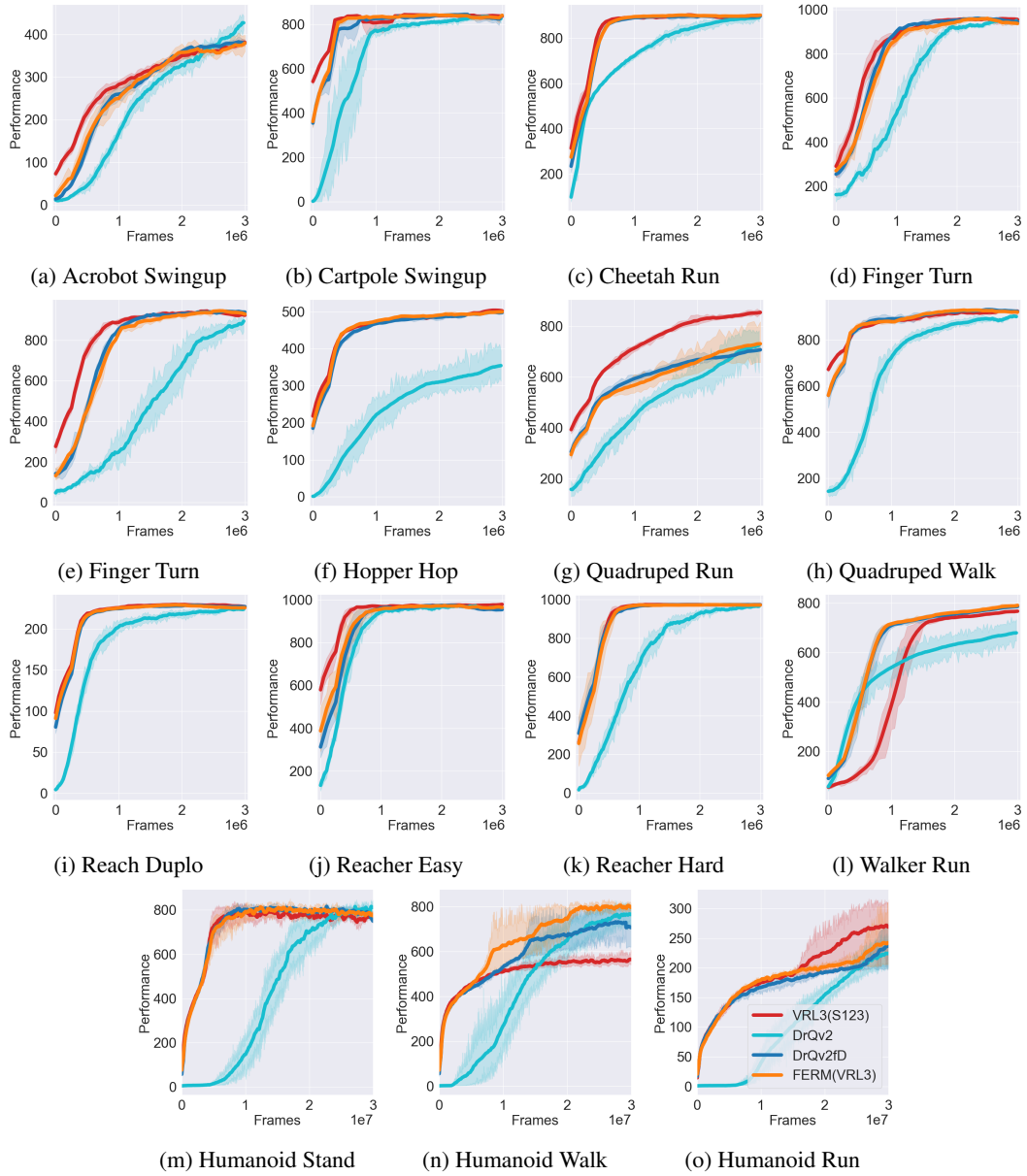


(a) Cartpole Balance (b) Cartpole Balance (c) Cartpole Swingup (d) Cup Catch

(e) Finger Spin (f) Hopper Stand (g) Pendulum Swingup (h) Walker Stand

(i) Walker Walk

Figure 8: Per-task performance comparison of VRL3 and other methods on all 9 Easy DMC tasks. The methods that utilize stage 2 data (VRL3, FERM, DrQv2fD) tend to learn faster. Overall, VRL3 has a small advantage over DrQv2 and FERM, but the performance gap is smaller compared to the results on Medium/Hard DMC tasks and the more challenging Adroit tasks. For VRL3, FERM and DrQv2fD, We use the same hyperparameter setting for all tasks. The DrQv2 results are from the authors.



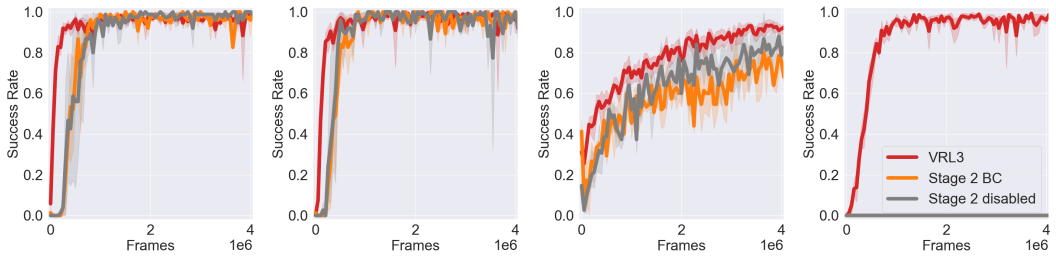
(a) Acrobot Swingup (b) Cartpole Swingup (c) Cheetah Run (d) Finger Turn

(e) Finger Turn (f) Hopper Hop (g) Quadruped Run (h) Quadruped Walk

(i) Reach Duplo (j) Reacher Easy (k) Reacher Hard (l) Walker Run

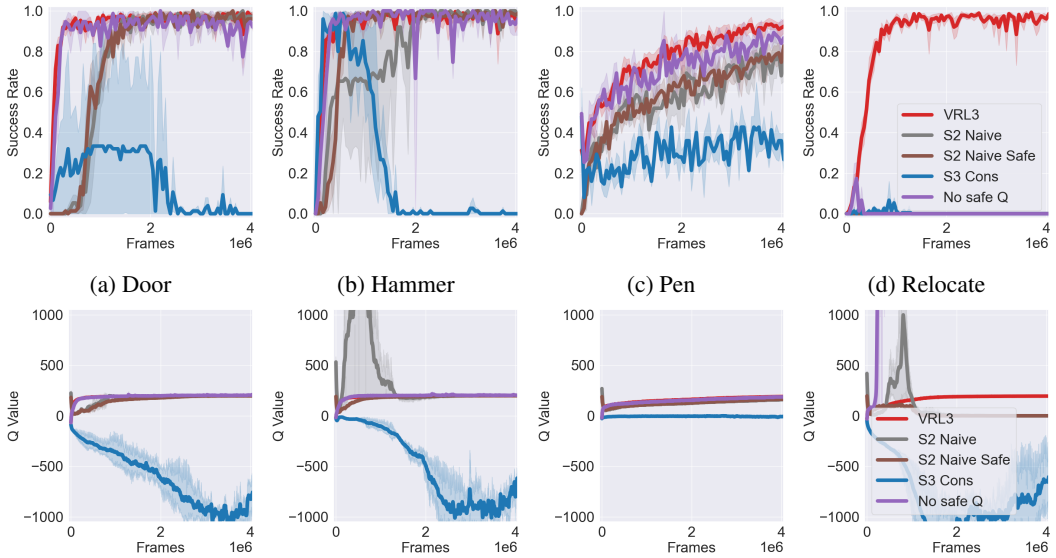
(m) Humanoid Stand (n) Humanoid Walk (o) Humanoid Run

Figure 9: Per-task performance comparison of VRL3 and other methods on all 15 Medium and Hard DMC tasks. The methods that utilize stage 2 data (VRL3, FERM, DrQv2fD) tend to learn faster. Overall, VRL3 has an advantage over DrQv2 and FERM in some environments, but overall, their performances are similar. This again shows that the advantage of multi-stage training procedures such as VRL3 is most prominent in the most challenging tasks with sparse reward and realistic visual input. For VRL3, FERM and DrQv2fD, We use the same hyperparameter setting for all tasks (they are the same for all 24 tasks). The DrQv2 results are from the authors.



(a) Door (b) Hammer (c) Pen (d) Relocate

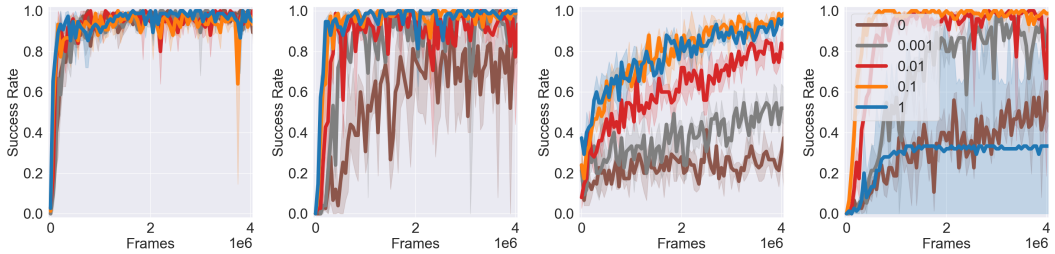
Figure 10: Per-task success rate comparison for VRL3, VRL3 with BC in stage 2 instead of conservative updates (Stage 2 BC), and VRL3 with stage 2 entirely disabled (Stage 2 disabled). Results show that using conservative RL updates in stage 2 is important, especially in Relocate. In the other three tasks, we also observe a significant performance gap in the early stage of training: VRL3 is able to quickly achieve a high performance with small amounts of online data, while the other 2 variants take much longer to reach the same level of performance.



(a) Door (b) Hammer (c) Pen (d) Relocate

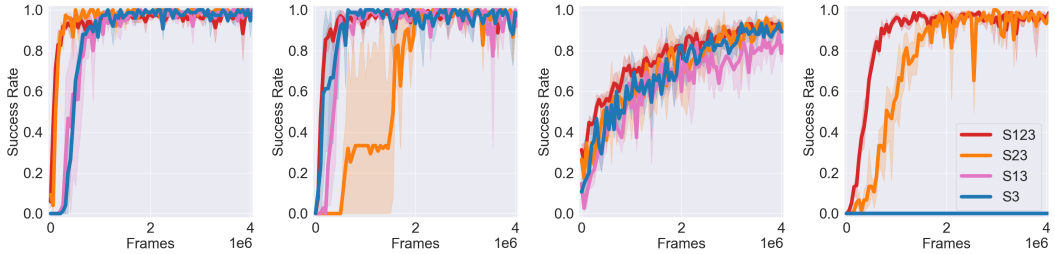
(e) Door (f) Hammer (g) Pen (h) Relocate

Figure 11: Per-task success rate (first row) and Q value (second row) comparison for VRL3 with different stage 2 to stage 3 transition methods. VRL3 by default uses conservative updates in stage 2, standard Q updates (without the conservative term) in stage 3, and uses safe Q in both stages 2 and 3. VRL3 has the best performance and the most stable Q values throughout stage 3 training. Each of the variants shown here is different from VRL3 in just one aspect. We can see that using naive RL updates in stage 2, and disable Safe Q lead to significantly reduced performance in all 4 tasks, and from the second row we see a significant overestimation issue. Note that using naive RL in stage 2 but enabling Safe Q leads to better performance in Hammer, and a now stable Q value in all four tasks. This shows that Safe Q is effective in stabilizing Q value estimates, but it cannot replace the role of proper stage 2 training. S3 Cons uses conservative learning in stage 3 instead of standard Q learning, giving the worst performance, and a significant underestimation of Q values in all tasks. And if we use conservative updates in stage 2, but disable Safe Q (No safe Q), then we can learn well in the easier 3 tasks: Door, Hammer and Pen. But in Relocate we run into severe overestimation and the agent fails entirely. This shows that although Safe Q might not be needed in easier tasks, it is critical in more challenging tasks. Note that Safe Q is not the only technique that can achieve this result, and we use it because we are aiming for a minimalist design, and Safe Q is easy to understand, to implement, and has negligible computation overhead.



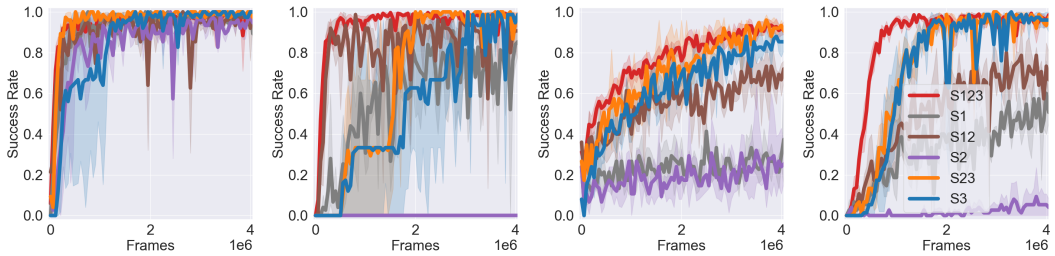
(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 12: Per-task success rate comparison for VRL3 with different encoder learning rate scales. Notice that in the easiest Door task, using an encoder learning rate scale of 0 (which means the encoder is pretrained in stage 1, and frozen afterwards) can still work. But in the more challenging tasks, tuning the encoder learning rate scale leads to much better results. In the most challenging Reloate task, note that the best encoder learning rate is 0.1 and 0.01: the encoder is being learned 10 or 100 times slower than rest of the agent (the policy and the Q networks).



(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 13: Per-task success rate comparison for VRL3 with different training stages enabled. Stage 1 pretraining is important for the Hammer and Relocate tasks (S123 > S23), while having minimal effect on the easiest Door task. Stage 2 is consistently important in all tasks (S123 > S13). Note that interestingly, stage 1 plus stage 3 is not always stronger than stage 3 alone (S13 < S3 on Pen). This shows that stage 1 pretraining should be treated carefully when there is a domain gap. Note that when all three stages are combined, we consistently get the best performance on all environments.

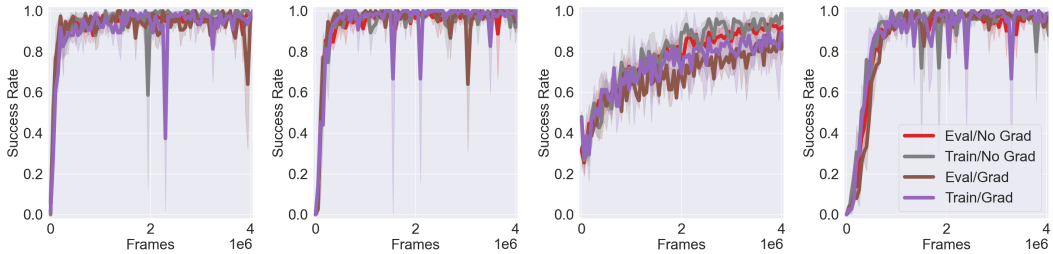


(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 14: Per-task success rate comparison for VRL3 with encoder training enabled in different stages. Note that since in stage 2, we only perform a small number of offline updates (30K updates) on a limited amount of RL data, stage 2 (S2) alone is not enough to obtain good representations for efficient learning (Although the Door task is easy and can be learned in all settings). Stage 1 (S1) alone is also not sufficient and gives bad performance, but combining stage 1 and 2 (S12) gives much stronger performance. Stage 3 alone also gives sub-optimal performance, especially on Hammer and Relocate. Compared to Figure 13, the setting in this figure isolates the effect of encoder feature learning from RL training.

A.4 Finetuning batch norm layers

When we finetune the convolutional encoder, one issue that is rarely studied in the RL setting is how the batch norm layers should be treated. In our case, by default, we set the batch norm layers to be in evaluation mode and disable further gradient updates for more stability. However, we also provide a set of ablations to show that in Adroit, these options are not really affecting performance too much. Figure 15 shows the results. We can see that all four variants give similar performance.

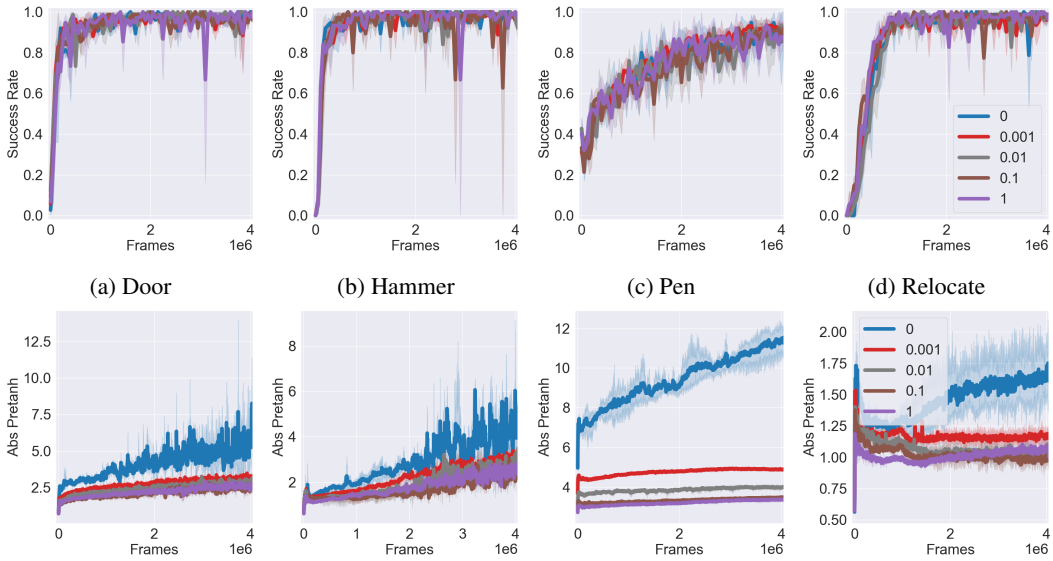


(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 15: Per-task success rate comparison for VRL3 with different batch norm training settings. Train and Eval indicate whether during finetuning in the stages 2 and 3, the batch norm layers are in train or evaluation mode (in evaluation mode, batch norm will use running statistics and not the standard per-batch statistics). No Grad means the batch norm layers will not be further updated with gradients from the RL objective.

A.5 Pretanh Penalty

Another issue that we considered (but is not discussed in the main paper) is whether the tanh activation function in the policy network might saturate during training. Note that since Adroit and DMC are tasks with continuous and bounded action spaces, a tanh function is typically used to bound the output action value from the policy network. So the "pretanh value" refers to the action value before going through the tanh activation. Note that if the absolute value of the pretanh value is too large then no gradient can be propagated back. We found that applying a penalty on the average absolute pretanh value can effectively prevent tanh saturation in the policy network. Though in terms of performance, it does not have a significant effect in Adroit and thus is not necessary. In DMC we found it occasionally can prevent exploration issues, so we apply a penalty weight of 0.001 in all DMC tasks. To implement the penalty we essentially add an L2 regularization term on the pretanh value to the policy loss. To avoid affecting the agent learning when unnecessary, we set a threshold value of 5 so that the penalty only applies when absolute pretanh values are larger than this threshold. Details are provided in our source code.



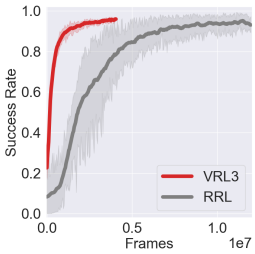
(a) Door (b) Hammer (c) Pen (d) Relocate

(e) Door (f) Hammer (g) Pen (h) Relocate

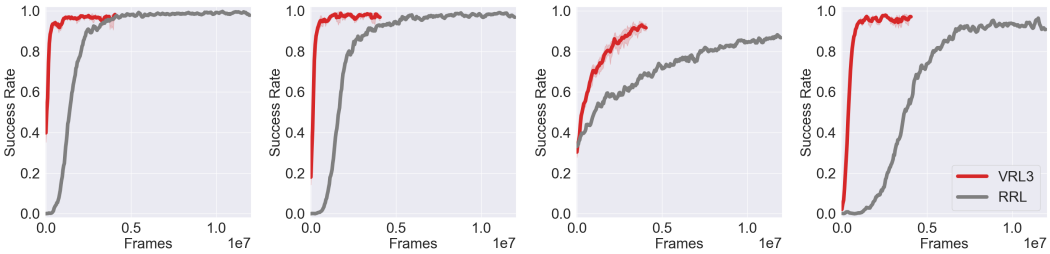
Figure 16: Per-task success rate comparison for VRL3 with different pretanh penalty weights, first row shows success rate, second row shows average absolute pretanh value. We can see that when we do not apply a pretanh penalty, the average absolute value of the pretanh values become fairly large, indicating tanh saturation. Although in the case of Adroit tasks, they seem to not affecting performance too much (This might be due to the particular implementation design in our backbone algorithm: the action noise is injected after the tanh activation. Although this design cannot prevent tanh saturation, it can in fact allow better exploration when tanh is saturated). Related issues have been studied in the past [24, 84], however, it might be worth it to revisit this important problem in future research.

A.6 Comparing to RRL Long-term Performance

Figure 17 shows that in terms of the long-term performance, VRL3 and RRL (trained until 12M, data provided by RRL authors) achieves similar results, VRL3 is slightly stronger in Relocate, and in Pen, VRL3 reaches 90% success rate at 3.25M data and then continues to increase, RRL fails to reach 90% success rate even at 12M data. Based on these numbers, it is fair to say that VRL3 outperforms RRL in both early stage and late stage. But note the long-term performance difference is smaller. That is why in our main text, we focused on discussing the improvement in sample efficiency (measured as the number of data needed to reach 90% or more success rate), compared to RRL, we are on average 780% better. In appendix B.1 we provide details on how we compute these numbers and also provide comparison on how fast we are to reach 50% and 75% success rate. In all cases, VRL3 has a significant advantage.



(a) Averaged over all tasks

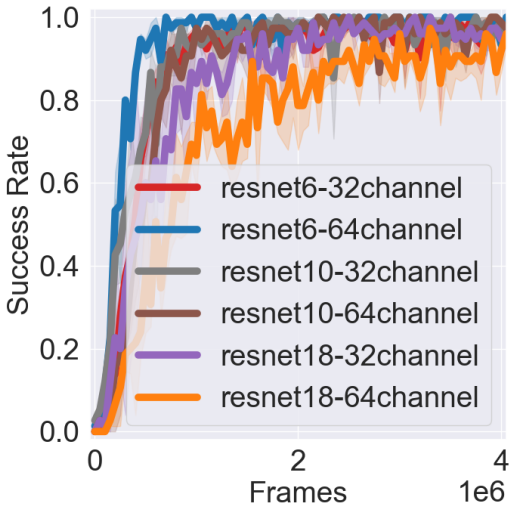


(b) Door (c) Hammer (d) Pen (e) Relocate

Figure 17: Success rate comparison for VRL3 at 4M and RRL at 12M. Results show that the long-term performance of RRL and VRL3 are similar. The advantage of VRL3 is mainly on sample efficiency.

A.7 Using Deeper ResNet Encoders

For the majority of our results, we use a shallow encoder with 5 convolutional layers. This is consistent with many prior works in pixel-input DRL research [33, 91, 41, 53, 54, 82] and also makes computation faster. However, it is also beneficial to know whether our method can work with deeper ResNet encoders. In Figure 18 we show that, on the hardest Relocate task, we can achieve similar performance with deeper and wider encoders. Here we study 6 different encoders: ResNet6 with 32 channels (default for VRL3), ResNet6 with 64 channels, ResNet10 with 31 channels, ResNet10 with 64 channels, ResNet18 with 64 channels, ResNet18 with 64 channels. For ResNet10 and ResNet18, we have skip connections as in typical ResNets. Results show that VRL3 can still learn effectively with these encoders of different capacity. However, note that ResNet18 achieves weaker performance, likely due to the fact that we are using the default hyperparameters. Since these encoders have much higher capacity, it might be possible to achieve better performance by finetuning the learning rate. Also note that when using ResNet6 with 64 channels, we observe an even better performance: it is able to reach 90% success rate on the hardest Relocate task with only 0.45M data, making it 24 times more sample efficient compared to the previous SOTA.



(a) Relocate

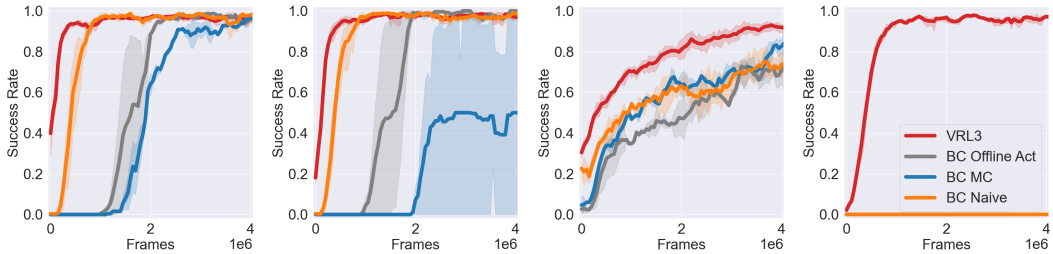
Figure 18: Success rate comparison for VRL3 with different ResNet encoders on the hardest Relocate task, all using the same default hyperparameters. When increasing the capacity of a ResNet6 encoder, we can obtain even stronger performance.

A.8 Alternative BC Baselines

In Figure 19 we present results for two other BC baselines: a) In stage 2, we perform BC updates, and we also train the value network to predict discounted cumulative reward of the offline trajectories (we call this "BC MC"). b) we train the value network with a slightly different method, we perform the standard Q-learning updates, but when computing the Q target, we replace the "next action from current policy" with the actual next actions in the offline trajectories, so as to circumvent the extrapolation error issue (we call this "BC Offline Act"). We then compare these 2 new variants with VRL3 (with offline RL in stage 2) and BC naive (the original BC baseline that only learns policy).

It seems natural to assume that these two new baselines would obtain better performance than BC naive. However, the result is quite interesting: First we observe that BC Offline Act achieves better performance than BC MC (which is not surprising because we expect BC Offline Act to learn a better value function since it uses actions in the offline trajectory), but then we also observe that both new variants achieve weaker performance compared to BC Naive. We did not expect this result because intuitively we would think that a somewhat decent policy-value match should certainly outperform a policy with untrained value function.

We pondered on this new result, and a potential explanation is that if we train the value network in a naive way in stage 2, this might inject harmful inductive bias into the value network, making it hard to perform reinforcement learning in the later stage. There have been a few recent papers that discuss this issue: for example, [58] show that early stage training, especially on a small dataset can impose a permanent effect on RL agents and reduce their long-term performance. Then it can be possible that an untrained Q network learns better in stage 3 compared to a naively trained Q network, because it is free from the inductive bias generated by the value learning process in BC MC and BC Offline Act. In a related work, [51] show that networks gradually lose capacity during DRL training, and propose to maintain this capacity with a simple regularization.



(a) Door (b) Hammer (c) Pen (d) Relocate

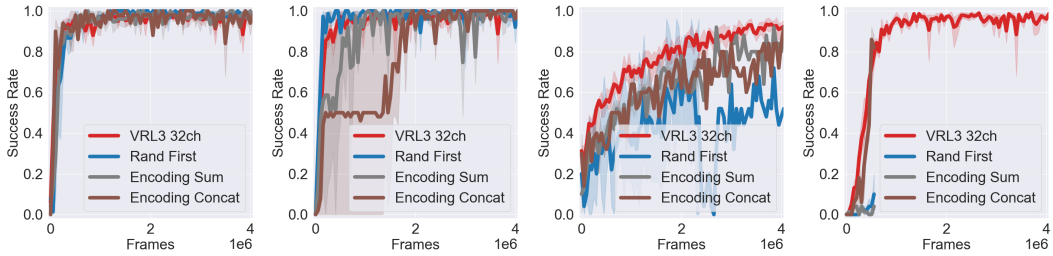
Figure 19: Success rate comparison of VRL3 and three BC baselines. Surprisingly, learning a value function during stage 2 along with the policy learned by BC does not improve performance over BC naive, where only the policy is learned.

A.9 CCE Ablations

In Figure 20 we present ablation on CCE and compare to three variants: 1) We initialize the first encoder layer randomly (Rand First). 2) We treat each input image separately, and then sum their encoding (Encoding Sum) 3) Similar to 2), but we concatenate their encoding (Encoding Concat) [57, 60, 88]. Results show that Rand First and Encoding Sum have a significant performance drop in Pen and Relocate. Encoding Concat has a performance drop in Hammer and Pen, but is similar to CCE on Door and Relocate. Additionally, On a V100 GPU, for the Relocate task, 1000 updates for CCE/Rand First take about 47 seconds, while for Encoding Sum, Encoding Concat and Encoding Diff Concat, they take about 200 seconds, about 4 times slower than using CCE.

Based on these results, it seems CCE is still the best fit for our framework, because it has strong performance, is easy to use (can be done in one line of code), and has no computation overhead.

We are also testing on another variant where we also concatenate the difference of image encodings, according to [72]. This method will incur even more computation overhead due to the additional concatenation. We will update the figures when these new experiments finish.



(a) Door (b) Hammer (c) Pen (d) Relocate

Figure 20: Success rate comparison of CCE and 4 other variants.

B Efficiency Comparison

B.1 Sample Efficiency Comparison

Table 3 shows a comparison of VRL3 and RRL in terms of sample efficiency. Sample efficiency is important because online data can be hard and expensive to obtain in DRL control tasks. Being able to reach a high sample efficiency means we can solve the task faster and with much less cost. Results show that VRL3 has much better sample efficiency in all tasks, giving an average of 780% times better sample efficiency in reaching 90% success rate in all Adroit tasks, with an impressive 1200% better sample efficiency on the most challenging relocate task. If we consider the newest result with doubling the number of channel for Relocate, then VRL3 can learn Relocate in 0.45M, making it 24 times more sample efficient than RRL.

Table 3: Sample efficiency comparison of VRL3 and RRL. The numbers show the number of data (frames) collected when the specified success rate is achieved. (to be precise, the frame number indicates the agent has an average success rate that is equal or greater than the specified success rate for the past 20K frames of training.) The last column show how many times VRL3 is more sample efficient than RRL in reaching that success rate. VRL3 is efficient in learning especially when aiming for a high success rate. Averaged over all Adroit tasks, we are 7.8 times more sample efficient than RRL, the previous SOTA method, to reach 90% success rate. To ensure our performance is not the result of "lucky random seeds"[25, 26, 15], the performance for VRL3 is averaged over 10 seeds. To ensure consistency with prior work, results for RRL are directly from the training log files provided by the authors. Note in the pen task, RRL's success rate at 12M is slightly less than 90%, but close.

Score	RRL	VRL3	VRL3 faster
Door at 50%	1360K	200K	6.8
Door at 75%	1880K	300K	6.3
Door at 90%	2560K	400K	6.4
Hammer at 50%	2080K	250K	8.3
Hammer at 75%	3120K	350K	8.9
Hammer at 90%	4440K	500K	8.9
Pen at 50%	1120K	400K	2.8
Pen at 75%	6200K	1600K	3.9
Pen at 90%	12000K	3250K	3.7
Relocate at 50%	4360K	500K	8.7
Relocate at 75%	6800K	650K	10.5
Relocate at 90%	11000K	900K	12.2

B.2 Parameter Efficiency Comparison

Table 4 gives a comparison of number of parameters for VRL3 and RRL. Note that in addition to the encoder, other networks in the agent can also have a lot of parameters, so we have to approach this comparison carefully. Note that VRL3 has a much smaller encoder and relatively larger MLP critic and actor networks. If we count all the parameters in the agent, then VRL3 has about 37% of the pararmeters as used in RRL with a ResNet34. Note that being parameter-efficient is also not exactly the same as being computationally efficient. In the next section, we discuss computation efficiency.

Table 4: Parameter efficiency comparison of RRL and VRL3. We compare the number of parameters in the encoder part, as well as parameters for all networks in the agent. Note that VRL3 has a much smaller encoder, but relatively larger critic and policy (actor) networks, VRL3 also has a pair of critic networks instead of just one. For encoder, VRL3 is more than 50 times more parameter efficient. Overall, VRL3 is about 3 times more parameter efficient.

Number of parameters (M)	RRL	VRL3	VRL3/RRL ratio
Encoder	21.28	0.40	0.0187
Policy network	2.0	3.0	1.5
Critic networks	2.0	6.0	3
All	25.28	9.4	0.37

B.3 Computation Efficiency Comparison

We now compare the computation efficiency of RRL and VRL3 . We do not take into account the time used for agent evaluation (in which case RRL is much slower due to the large ResNet34 encoder) for RRL we use the authors' codebase. On a single NVIDIA V100 GPU, for door, RRL runs at 83 FPS while VRL3 runs at 68 FPS; for relocate, RRL runs at 53 FPS while VRL3 runs at 47 FPS.

Note that RRL is faster in per-frame computation mainly because its encoder is frozen, so images only need to be processed by the encoder once when they are collected. For VRL3, although the encoder is 50 times smaller, the encoder will be updated in all stages, and the off-policy method we are based on also use a larger batch size and larger actor and critic networks. Note that altough VRL3 will take a bit longer time to train for the same number of frames, it is significantly faster when we consider the fact that it can learn a task with a small amount of online data. In terms of the computation time used to reach 90% success rate, VRL3 is much faster on all tasks, and more than 10 times faster on the most challenging relocate task. Note that here we are not considering acceleration from parallel or multi-core computing.

Table 5: Computation efficiency comparison of RRL and VRL3.

Computation Time (Hours)	RRL	VRL3	VRL3/RRL ratio
Door, Pen, Hammer (4M)	13.39	16.34	1.22
Relocate (4M)	20.96	23.64	1.13
Door (90% success)	8.57	1.63	0.19
Hammer (90% success)	14.86	2.04	0.14
Pen (90% success)	40.16	13.28	0.33
Relocate (90% success)	57.65	5.32	0.09

B.4 Computing Infrastructure

Our experiments are conducted on NVIDIA Tesla P40, P100 and V100 GPUs. When calculating the computation efficiency table, we use a single V100 GPU.

C Other Technical Details

C.1 DMC, Adroit and MuJoCo Usage

To use the MuJoCo[80] physics simulator, please refer to `https://mujoco.org/`, as of the writing of this work, MuJoCo is now free and a license is not required. For DMC[78] please refer to `https://www.deepmind.com/publications/deepmind-control-suite`. For Adroit [64] please refer to `https://github.com/facebookresearch/RRL` and `https://sites.google.com/view/deeprl-dexterous-manipulation`.

C.2 DrQv2 Codebase

Our backbone algorithm is DrQv2[91], the code can be found at `https://github.com/facebookresearch/drqv2`. The code is under MIT License.

C.3 Stage 1 Pretraining

Our code for stage 1 pretraining largely follows the example code given here: `https://github.com/pytorch/examples/blob/main/imagenet/README.md`.

C.4 ImageNet Dataset

The training ImageNet data can be found here: `https://www.kaggle.com/competitions/imagenet-object-localization-challenge`, to organize the validation data, we use the script here: `https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh`

C.5 Algorithm Psuedocode

For better clarity, in the main paper we provide a high-level description of our framework. Here we also provide the psuedocode.

Algorithm 1 Visual DRL in 3 stages

Initialize parameterized encoder f_ξ, policy π_ϕ, Q functions $Q_{\theta_1}, Q_{\theta_2}$, Q target functions $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$, empty buffer $\mathcal{D}$. Set target parameters $\bar{\theta}_i \leftarrow \theta_i,$ for $i = 1, 2$. N_{S2} is the number of offline RL updates in stage 2; N_{S3} number of online RL updates for stage 3.
Stage 1:
Load pretrained encoder parameters into ξ and setup encoder for RL training.
Stage 2:
Load offline RL data into $\mathcal{D}$
for N_{S2} updates **do**
    Sample a mini-batch B from $\mathcal{D}$
    Update f_ξ and $Q_{\theta_1}, Q_{\theta_2}$ with $\mathcal{L}_Q + \mathcal{L}_{QC}$, update π_ϕ with $\mathcal{L}_\pi$. Update $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$ with polyak averaging.
Stage 3:
for N_{S3} updates **do**
    Sample an action from policy, take action and store the new data in $\mathcal{D}$
    Sample a mini-batch B from $\mathcal{D}$
    Update f_ξ and $Q_{\theta_1}, Q_{\theta_2}$ with $\mathcal{L}_Q$, update π_ϕ with $\mathcal{L}_\pi$. Update $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$ with polyak averaging.

D Further Discussions

D.1 Additional Related Work on Bias Reduction

Overestimation and underestimation bias issues have been studied in a large number of previous works, and has long been identified as one of the major obstacles in off-policy learning [79]. A number of important works have achieved significant performance gain over previous methods mainly with better bias control, For example, DDQN [82] achieves much stronger performance over DQN[53, 54] with double Q learning, TD3[19] greatly improve robustness of DDPG[49] with Clipped Double Q (CDQ). CDQ is also used in other popular methods such as SAC[22] and DrQ[33, 91]. Other methods such as weighted Bellman updates can reduce bias propagation[43], multi-step methods can also help[52]. Recently, it has been shown that the underestimation issues can also occur in DRL training and be problematic [39], finer-grained bias control can further improve performance [37], and ensemble-based bias reduction can exploit an increased number of network updates and greatly improve sample efficiency [7]. Note that the bias problem is a well-established issue that has been studied in many papers and this is not an exhaustive list.

Fundamentally, the Safe Q technique is another bias reduction technique that helps training by posing a soft constraint on the Q value estimates. The stage 2-3 transition scenario is a bit similar to the increased network updates setting as studied in [7], where the bias issue is exacerbated, and thus additional bias control is required. Note that VRL3 uses Safe Q on top of CDQ. Figure 11 essentially shows that CDQ alone (all variants in the figure use CDQ) is not enough to tackle the bias problem during stage transition. Note there are likely other methods that can achieve the same result as the Safe Q technique, and we use the Safe Q technique here because of its unique advantages: a) it is effective, and it is easy to see that it becomes difficult for Q values to diverge under Safe Q, while methods like CDQ do not provide such robostness (they do not care if the Q values are becoming too large). b) Safe Q is a minimalist solution for stage 2-3 transition, is easy to implement and has negligible overhead.

D.2 Additional Discussion on Future Work

The success of VRL3 opens up a number of exciting future research directions in data-driven DRL.

One direction is to use on-policy methods in the VRL3 framework. Although on-policy methods tend to have worse sample efficiency due to the difficulty of using off-policy data, recent studies show that on-policy methods can use data many more times to learn useful features with value distillation [11]. In this way it might possible to combine the unique advantages of on-policy methods with the effectiveness of a data-driven approach.

Model-based methods (to list a few, [27, 5, 40, 98, 38, 10, 65, 13, 36]) can be used together with VRL3. Model-based methods tend to be more complex than model-free ones, however, these models are typically learned in a supervised manner, making them good candidates for building advanced data-driven methods. Recently, there are also model-based methods that can tackle offline RL setting [94, 30, 1]. Transformer-based methods are also an promising direction to pursue and they might provide unique advantages since they follow a supervised learning pipeline [28, 6], recently, an interesting work has shown that pretraining with data from different domains can help transformer-based RL learning [66].

Contrastive pretraining techniques can also be useful in such a data-driven approach. Although our experiments show that contrastive representation learning in stage 2 alone do not work well, they can be combined with data-augmented offline and online RL. Recently, a number of methods have shown that model-based and contrastive methods can help accelerate online learning [67, 89]. In our current design, for stage 2 and 3, we use data coming from the exact RL task we want to solve. It is also possible to utilize data from RL tasks that are related, but are not exactly the task we are solving. Here contrastive pretraining has a lot of potential.

Another important direction is to understand how the domain difference between data sources affect the learning process and how we can best utilize stage 1 data when the RL task environment has different visuals.

D.3 Discussion on Potential Negative Societal Impacts

This particular work focuses on testing in simulated benchmarks, but the long-term goal is to build towards general RL methods that can achieve strong performance in challenging real-world visual control tasks.

Since our method is data-driven and utilizes large non-RL image datasets for pretraining, a potential issue is our model can be affected by the bias in these datasets. In supervised learning, there is a large body of research on how bias in the data can be harmful in many different aspects, and many of these problems can also apply to the reinforcement learning setting.

For instance, an autonomous vehicle that drives on a highway can be unsafe if its training data is unevenly distributed and certain obstacle or specific types of objects are not well represented in the data. In such cases, the agent might not be able to properly recognize these novel objects and this leads to significant safety issues. Even if the data is evenly distributed, there might be potential adversarial attacks that are designed to break data-driven intelligent systems, and special training procedures are necessary to tackle these possibilities. In a multi-stage training setting, special care should be taken on each and every training stage to ensure safety. The most popular and sample efficient DRL methods nowadays, such as SAC [22] and DrQv2 [91] do not yet have built-in procedures that aim to tackle these issues and much further research is needed to develop agents that are both efficient and reliable.

D.4 Additional Discussion on Data Augmentation Technique

In our work, we use random shift data augmentation, same as our backbone algorithm DrQv2. This data augmentation technique has been shown to achieve superior performance in DRL setting, compared to other alternatives, and has been extensively studied in the DrQ paper [33]. Later on, a slightly modified version of this method is proposed to make computation more efficient [91].

In [33], section E "Image Augmentations Ablation" in the appendix, the authors provide ablation against the following alternatives: Cutout, Horizontal/Vertical Flip, Rotate, Intensity on 6 DMC tasks. Random shift achieves the best performance, and outperform the second best, Rotate, by a large margin.

D.5 Additional Discussion on Other Recent Papers

Here we discuss how our work is related to and different from a number of recent papers (most are submitted to arxiv in 2022).

[48] show that combining self-supervised learning and RL does not necessarily leads to consistent performance improvement, this is an interesting observation and in our results we also show that self-supervised contrastive learning can be safely removed in stage 2 when offline RL is applied and updates the encoder. This work studies a different topic and is not tested on Adroit. They do not [74] show that features from pretrained networks can be used to generate semantic scene segmentation and help robotic navigation. They do not test on standard benchmarks and do no consider offline RL data. [99] show that policy pretraining on YouTube videos can help driving, they do not study offline RL and is focused on a different task. [12] study how pretrained model can be used for zero-shot goal specification, this is a difference topic and they do not test on Adroit. [55] study how transformer representations can be used for control, they do not test on the Adroit benchmark. [95] is a short paper that showed up in openreview in July, they are very similar to RRL, except they do not use demonstrations and train in an actor-critic way, they are not tested on Adroit. [69] use video prediction for pretraining, they do not consider and offline RL setting and do not test on Adroit. [77] use pretrained representations to facilitate exploration, they do not consider how to combine 3 different data sources and do not test on Adroit. [14] show that representations learned via Neural Radiance Fields can improve RL performance compared to other representations, they do not consider offline RL data and do not test on Adroit. [68] show that decoupling representation learning and dynamics learning can be helpful in model-based RL, which is different from our setting which is model-free RL. They do not have non-RL pretraining and is not test on Adroit. [9] study a more sohpisticated method on learning spatial features for visual control tasks. They do not consider the offline RL setting and do not test on Adroit. [81] study the setting where an existing policy can be used to help training. They do not study representation pretraining, and they tested on the raw-state Adroit environment, not the pixel-input one. [50] tackle robotic manipulation tasks with a sophisticated algorithm, it seems they benchmarked on raw-state Adroit setting, not the pixel-based setting. [83] study how data from multiple tasks can be used to improve performance on new tasks, the multi-task setting is different from ours, and they do not test on Adroit. [23] propose a new imitation learning method, they study a different subject, and they do not have non-RL pretraining, and do not test on Adroit. [73] show that human videos can help agents learn robotic manipulation, their subject is not exactly reinforcement learning, and they also do not benchmark on the Adroit tasks.

Note all above works are different from our work in one or more aspects, we also located a few works that have tested on the pixel-input Adroit benchmark: [63] is a thesis submitted in May, they show that performance of RRL can be slightly improved with a better pretrained representation, on Hammer, they reach 90% success rate at 3.2M data, while we are at 0.5M, so their performance is weaker than ours (we are more than 6 times more sample efficient than this result). [88] show masked image pretraining can lead to strong performance on a new suite of robotic tasks. However, they do not consider offline RL training, and do not test on Adroit. [60] and [57] show that pretraining on different datasets and with more sophisticated methods can be combined with imitation learning. Compared to them, we study RL, which is a different subject, and we also achieve better success rate on Adroit (95%, while they have 85% and <70%, respectively).