Pretrained VAEs on World Models

May 28, 2025

Reference

Moritz Schneider, Robert Krug, Narunas Vaskevicius, Luigi Palmieri, and Joschka Boedecker. The surprising ineffectiveness of pre-trained visual representations for model-based reinforcement learning, 2025. URL https://arxiv.org/abs/2411.10175.

Yuhan Zhang, Guoqing Ma, Guangfu Hao, Liangxuan Guo, Yang Chen, and Shan Yu. Efficient reinforcement learning through adaptively pretrained visual encoder, 2025. URL https://arxiv.org/abs/2502.05555.

The Surprising Ineffectiveness of Pre-Trained Visual Representations for Model-Based Reinforcement Learning

Research Questions

1. Is MBRL more sample efficient when using PVRs (pre-trained visual representations) compared to learning from scratch?
2. Can model-based agents generalize better to OOD settings with PVRs?
3. How important are training properties (data diversity, architecture) of PVRs for downstream RL tasks?
4. How does the quality of learned dynamics models differ between scratch-trained and PVR-based models?

[Schneider et al., 2025] Contributions

- **Benchmarking PVRs for MBRL.** First comprehensive comparison of PVRs for MBRL, studying generalization to OOD settings
- **OOD Evaluation.** Investigated distribution shifts for both PVRs and MBRL agents, finding that scratch-trained agents often outperform PVR-based ones
- **PVR Properties.** Identified data diversity and network architecture as key factors for OOD generalization
- **Model Analysis.** Found that scratch-trained world models are generally more accurate with fewer reward prediction errors

PVR-based DreamerV3 and TD-MPC2 architectures

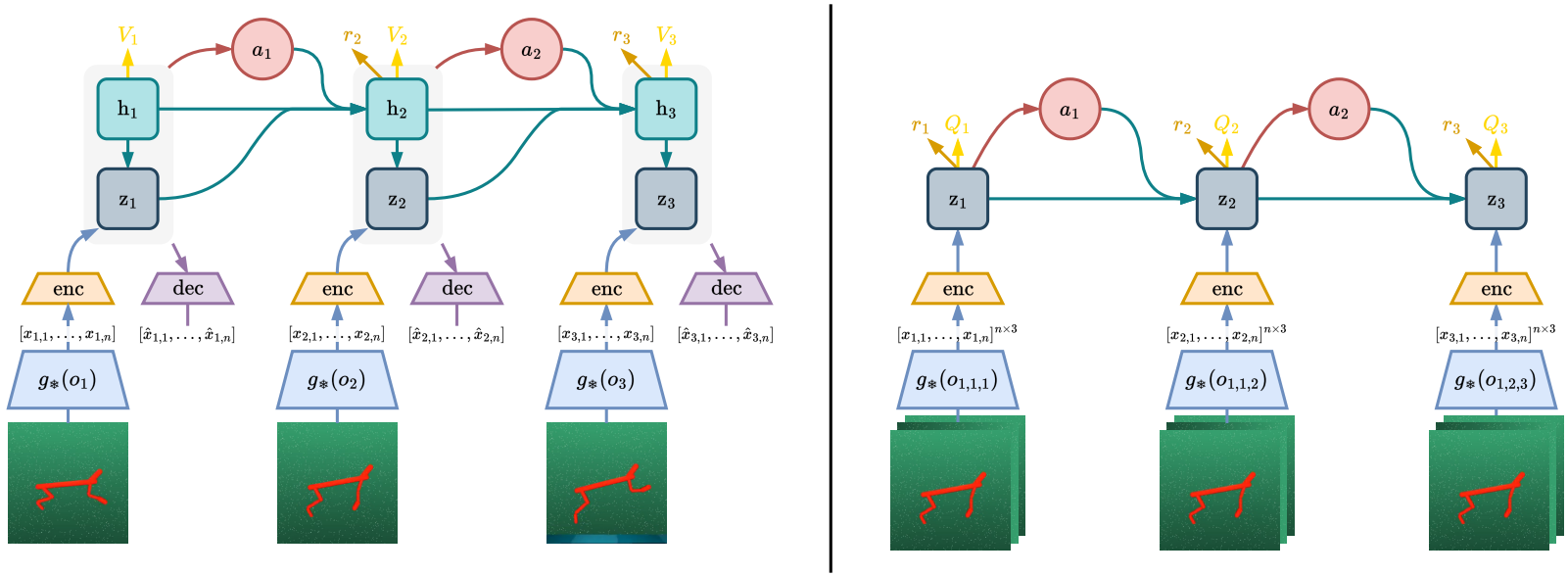


Figure: **Components of PVR-based DreamerV3 (left) and TD-MPC2 (right) architectures.** In DreamerV3, the output x_t of the frozen pre-trained vision module $g_{❄}$ is given to the encoder $\mathrm{enc}(z_t|x_t)$ which maps its input to a discrete latent variable z_t. In TD-MPC2 a stack $x_{t-3:t}$ of the last 3 PVR embeddings is given to the encoder $\mathrm{enc}(x_{t-3:t})$ which maps the inputs to fixed-dimensional simplices. The encoder of DreamerV3 additionally requires the recurrent state h_t as input. The rest of both algorithms remains unchanged.

Experimental Setup

- **MBRL Algorithms:** DreamerV3 and TD-MPC2
  - Freeze pre-trained vision backbone $g(o_t)$
  - Learn single linear layer on frozen features
  - Keep other components unchanged
- **PVRs:** 8 models + custom autoencoders
  - CLIP, R3M, Taskonomy, VIP, DINOv2, OpenCLIP, VC-1, R2D2
  - Mix of self-supervised, contrastive, and autoencoding methods
  - Custom autoencoders pre-trained on same images for ablation
- **Environments:** 10 tasks across 3 domains
  - DeepMind Control Suite (5 tasks)
  - ManiSkill2 (4 tasks)
  - Miniworld (1 task)
  - All use 256×256 RGB observations
- **Training:**
  - DMC: 3M steps, ManiSkill2/Miniworld: 5M steps
  - 12 short rollouts every 50k steps
  - 200 full episodes for final evaluation

Data Efficiency

- **Research Question:** Are PVR-based MBRL agents more data efficient than scratch-trained ones?
- **Key Finding:** Scratch-trained agents often perform better or equal to PVR-based ones
- **Surprising Result:** Even autoencoders pre-trained on task-specific data don't help
- **Why?** Objective mismatch in MBRL makes it harder to adapt to existing representations

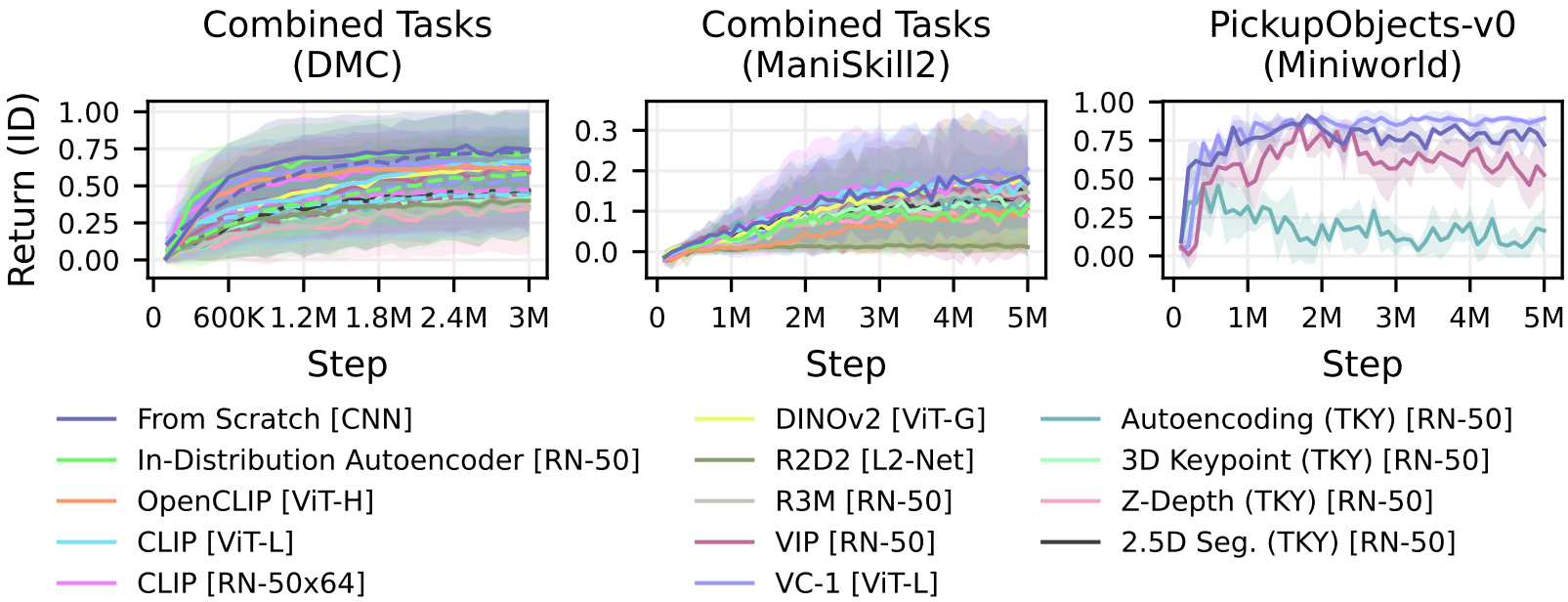


Figure: **Normalized ID performance and data-efficiency comparison** on DMC, ManiSkill2 and Miniworld environments between the different representations. Each line represents the mean over all runs with a given representation, the shaded area represents the corresponding standard deviation. Solid lines represent DreamerV3 runs, whereas dashed lines indicate TD-MPC2 experiments. Especially in the DMC experiments, representations trained from scratch outperform all PVRs also in terms of data-efficiency. Curves of each environment individually can be found in Appendix.

Generalization to OOD Settings

- **Research Question:** Do PVRs perform better than scratch-trained agents in OOD domains?
- **Key Findings:**
  - Most PVRs don't generalize well across domains
  - Only VC-1 shows some cross-domain capability
  - Even task-specific autoencoders underperform
  - Scratch-trained agents consistently outperform PVRs
- **Surprising Result:** Despite being trained on diverse data, visual foundation models don't show better OOD generalization
- **Why?** The objective mismatch in MBRL makes it challenging to leverage pre-trained representations effectively

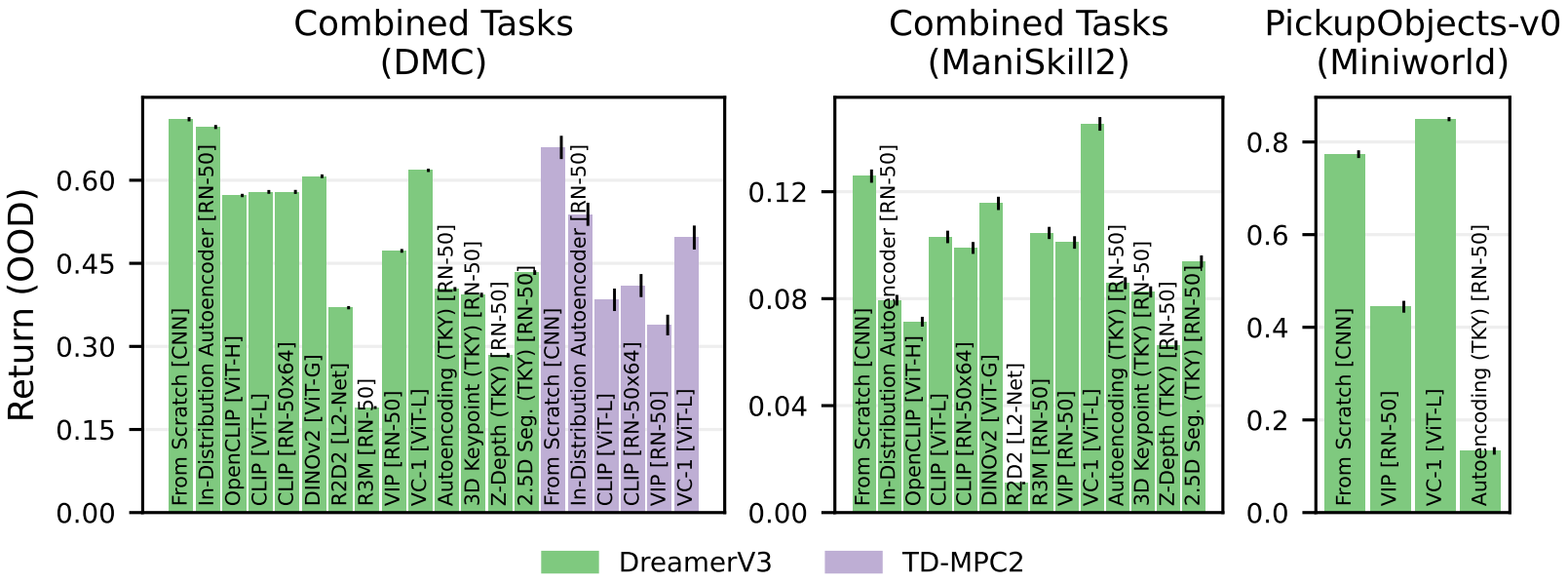


Figure: **Average normalized performance on DMC, ManiSkill2 and Miniworld tasks in the OOD setting.** The baseline representation learned from scratch outperforms all PVRs, even in the OOD settings. Thin black lines denote the standard error.

Properties of PVRs for Generalization

- **Research Question:** What properties make PVRs effective for OOD generalization?
- **Key Findings:**
  - Language conditioning not necessary for good OOD performance
  - Sequential data training somewhat beneficial (especially in ManiSkill2)
  - Data diversity important for performance in both domains
  - ViT architecture shows good performance but not the only factor
- **Surprising Results:**
  - Language conditioning, despite being popular, doesn't help control tasks
  - Data diversity more important than network architecture
  - ViT-based CLIPs perform similarly to ResNet-based ones

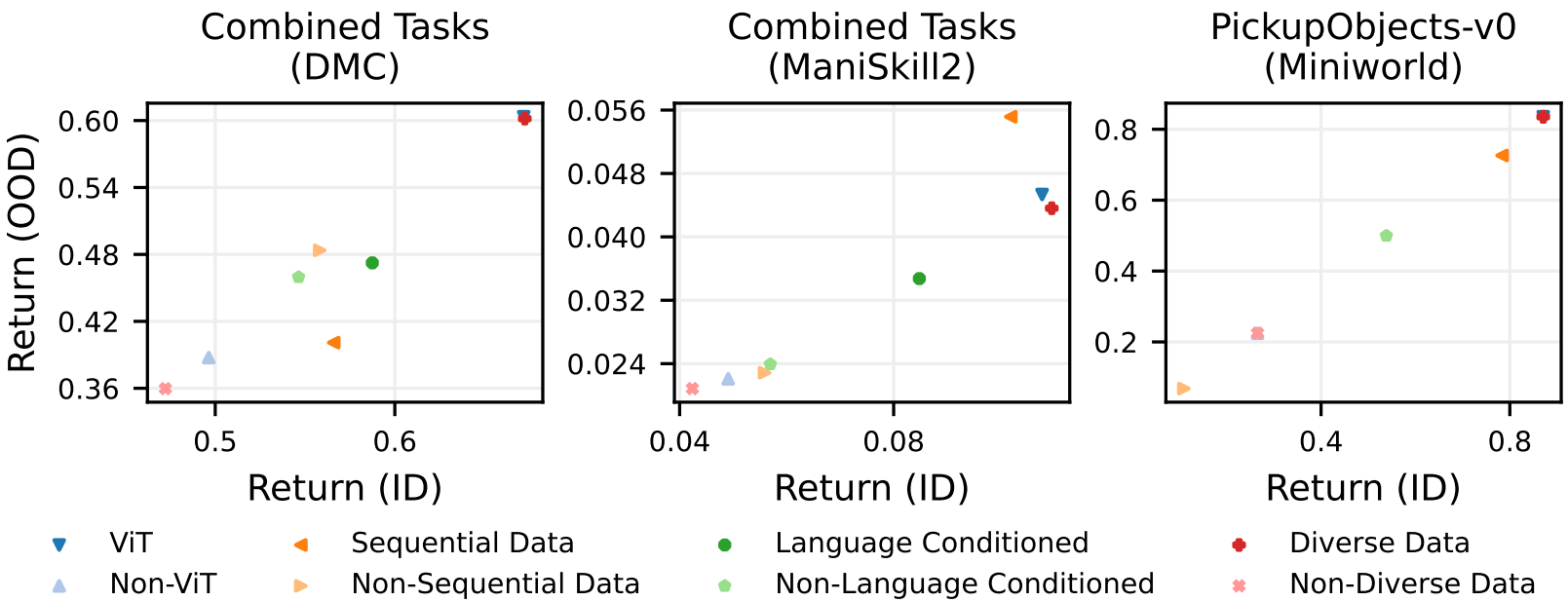


Figure: **IQM return of different categorizations.** Each marker shows interquartile-mean performance. ViT and diverse data representations excel in OOD settings. Sequential data helps in ManiSkill2 and Miniworld but not DMC. See Appendix for individual environment plots.

World Model Differences

- **Research Question:** Why do PVR-based agents underperform despite having powerful vision backbones?
- **Key Findings:**
  - Dynamics prediction error similar between PVR and scratch-trained agents
  - Reward prediction error significantly higher for most PVRs
  - Strong correlation between reward prediction accuracy and task performance
  - Latent space organization differs: scratch-trained encoders better separate states by reward
- **Interpretation:**
  - State prediction not the bottleneck - PVRs perform similarly to scratch
  - Reward prediction is critical - only scratch-trained encoders learn reward-relevant features
  - PVRs' frozen features lack reward-relevant information needed for planning
  - This explains why PVR-based agents struggle compared to end-to-end learning

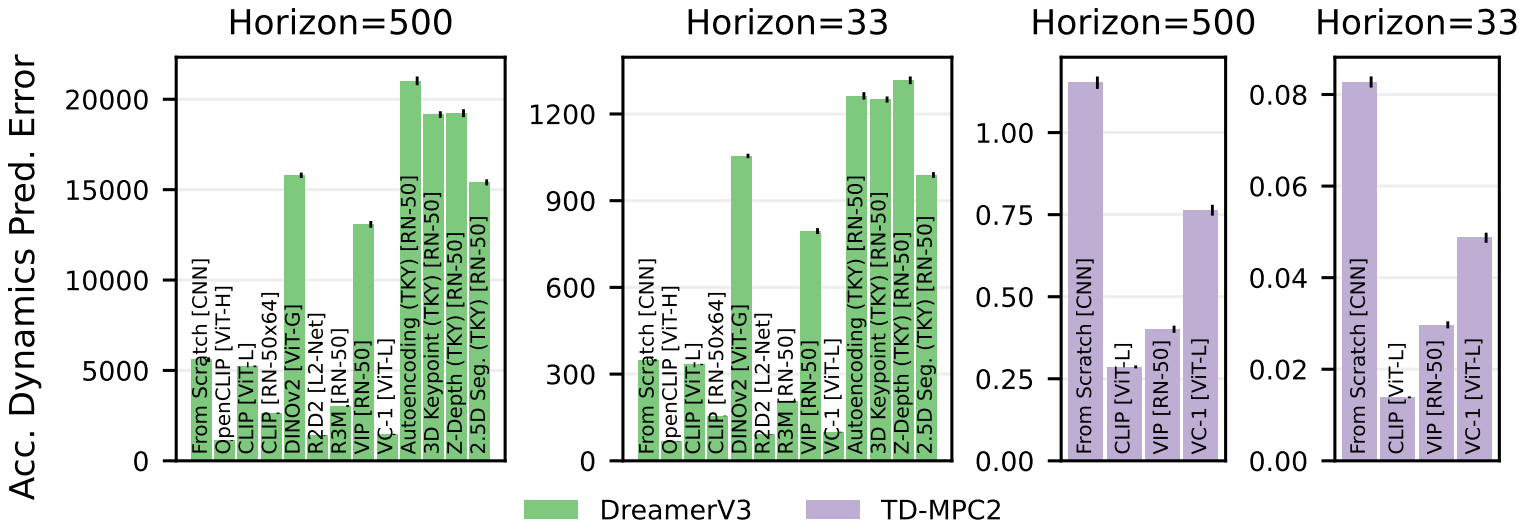


Figure: **Dynamics Prediction Errors** on `Pendulum Swingup` (200 trajectories). DreamerV3: KL divergence between prior/posterior distributions. TD-MPC2: MSE between predicted and encoded latent states. Error bars show standard error.

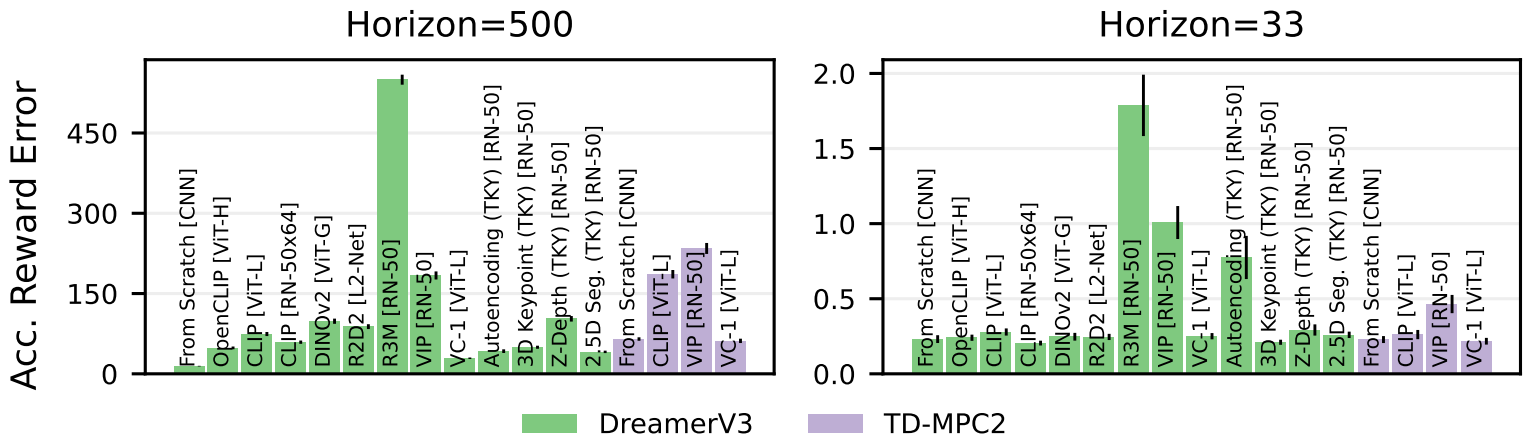


Figure: **Reward Prediction Errors** on `Pendulum Swingup` (200 trajectories). Error is $|r_t - \hat{r}_t|$. Error bars show standard error.

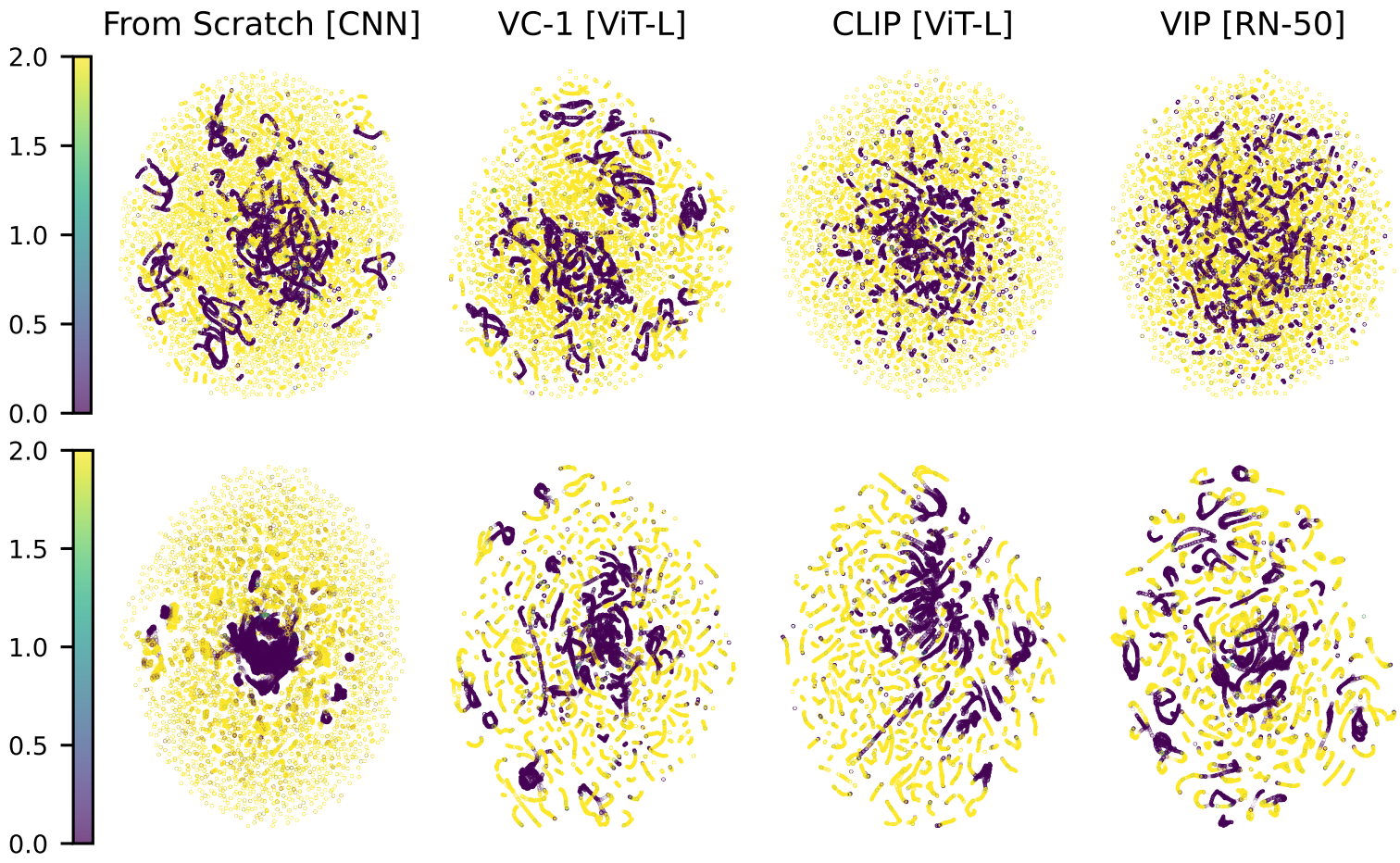


Figure: **UMAP projections of DreamerV3 (top) and TD-MPC2 (bottom) encodings** color-coded by reward. Points show states from `Pendulum Swingup`. Scratch-trained representations better separate high/low reward states than PVR embeddings.

Efficient reinforcement learning through adaptively pretrained visual encoder

APE [Zhang et al., 2025] Pipeline

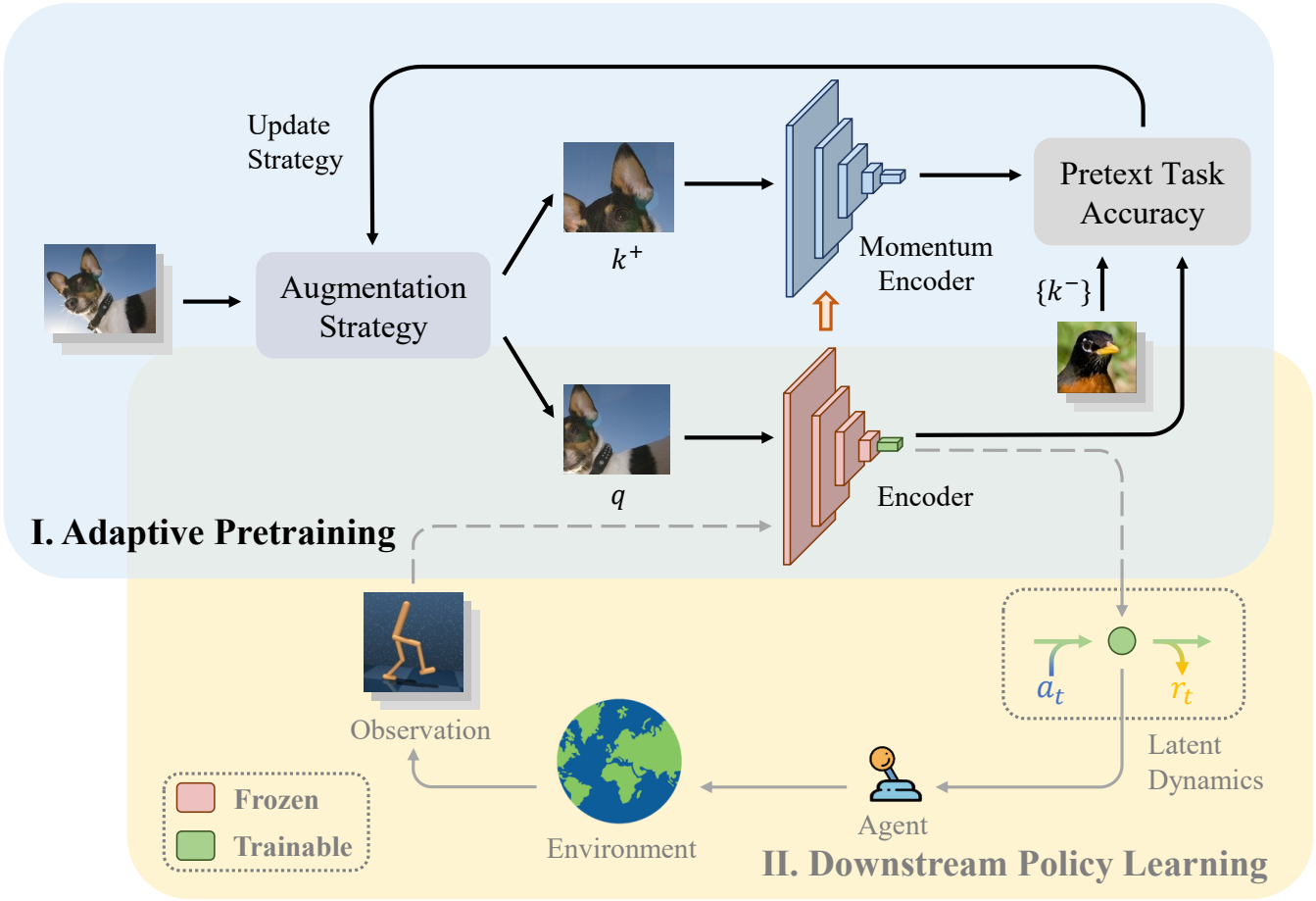


Figure: APE pipeline for MBRL. Training consists of two phases: Adaptive Pretraining (blue) and Downstream Policy Learning (yellow). The first phase uses adaptive data augmentation on real-world images, dynamically adjusting augmentation probabilities. The second phase integrates the pretrained encoder as a perception module in the RL framework.

Two-Stage Architecture: Adaptive Pretraining (1/2)

- **Backbone:** MoCo v2 contrastive framework
  - Query encoder f_q and momentum encoder f_k
  - For each image, sample two augmentations to get embeddings:

$$q = f_q(\text{view}_1)$$
$$k^+ = f_k(\text{view}_2)$$

  - Negatives $\{k^-\}$ come from other images in batch/memory queue
  - Contrastive loss (InfoNCE) between views:

$$\ell_q = -\log \frac{\exp(q \cdot k^+/\tau)}{\exp(q \cdot k^+/\tau) + \sum_{k^-} \exp(q \cdot k^-/\tau)}$$

Two-Stage Architecture: Adaptive Pretraining (2/2)

- **Multiple Augmentation Compositions**
  - Divide batch into N sub-batches with distinct augmentations
  - Weighted loss over compositions:

$$\mathcal{L}_z = \sum_{i=1}^{N} p_i \, \ell_q^{(i)}$$

  - Closed-loop probability updates based on pretext accuracy:

$$p_i^{\text{new}} = \frac{\exp\left(\alpha \left(1 - \text{Acc}_i\right)\right)}{\sum_j \exp\left(\alpha \left(1 - \text{Acc}_j\right)\right)}$$

    where $\alpha = 0.8$ for $N = 7$ and $\alpha = 1$ for $N = 3$
  - **Intuition:** Harder compositions ($\text{Acc}_i \ll 1$) get more weight
- **Key Outcome:**
  - Encoder learns invariance to visual changes
  - Mastery of diverse augmentation recipes
  - Rich, general features for downstream RL

Two-Stage Architecture: Downstream Policy Learning

It is exactly the structure of the DreamerV3 model.

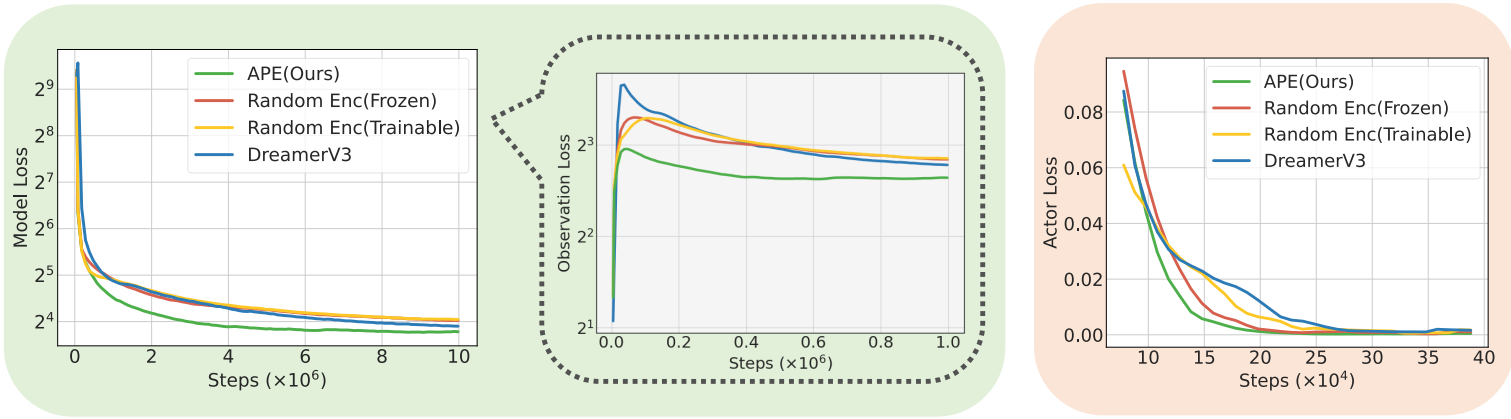
$$
\begin{aligned}
\mathcal{L}_{rew}(\theta) &= -\log\Big(p_\theta^R(\hat{r}_t \mid z_t, z_{t-1}, a_{t-1})\Big) \\
\mathcal{L}_{con}(\theta) &= -\log\Big(p_\theta^C(\hat{c}_t \mid z_t, z_{t-1}, a_{t-1})\Big) \\
\mathcal{L}_{rec}(\theta) &= -\log(g_\theta(\hat{x}_t \mid z_t, z_{t-1}, a_{t-1})) \\
\mathcal{L}_{obs}(\theta) &= \beta_1 \max(1, \mathrm{KL}[\mathrm{sg}(f_\theta(z_t \mid z_{t-1}, a_{t-1}, o_t)) \\
&\qquad \| \, p_\theta^D(\hat{z}_t \mid z_{t-1}, a_{t-1})]) \\
&\qquad + \beta_2 \max(1, \mathrm{KL}[(f_\theta(z_t \mid z_{t-1}, a_{t-1}, o_t)) \\
&\qquad \| \, \mathrm{sg}(p_\theta^D(\hat{z}_t \mid z_{t-1}, a_{t-1}))])
\end{aligned}
$$

$$
\mathcal{L}(\theta) = \mathcal{L}_{rew}(\theta) + \mathcal{L}_{con}(\theta) + \mathcal{L}_{rec}(\theta) + \mathcal{L}_{obs}(\theta)
$$

Experiments Overview

- **Key Research Questions:**
  - Can APE improve learning speed and visual generalization?
  - Does it work for both model-based and model-free RL?
  - What makes APE effective and how sensitive is it to choices?
- **Pretraining Setup:**
  - Self-supervised on ImageNet-100
  - ResNet-18 backbone
  - Five augmentation primitives:
    - Color Jitter, Grayscale, Gaussian Blur
    - Random Resized Crop, Horizontal Flip
  - Dynamic re-weighting based on difficulty
  - Gaussian blur as main augmentation (f_{main})
- **Evaluation:**
  - Linear probe accuracy on ImageNet-100:
    - MoCo v2: 90.8%
    - APE (jitter): 91.1%
    - APE (blur): 91.7% (best)
  - Consistent hyperparameters across all experiments
  - Results averaged over $\geq$ 3 runs

DMC results



(a) Model loss comparison

(b) Actor loss comparison

Figure: Loss comparison between DreamerV3, encoder with frozen random initialized parameters, encoder with trainable random initialized parameters and APE. The last layer of the frozen random initialized encoder is finetuned during training. The absolute value of actor loss is used.

- **Starting Strong:** APE's pretrained encoder helps the model learn faster by giving it good initial features
- **Working Together:** When different parts of the model share good features, they all learn better and more steadily
- **Better Search:** Better features help the agent explore more of its environment, which is important for harder tasks
- **Overall:** Pretraining the vision model helps the agent learn faster and perform better on complex tasks

Results on Other Benchmarks

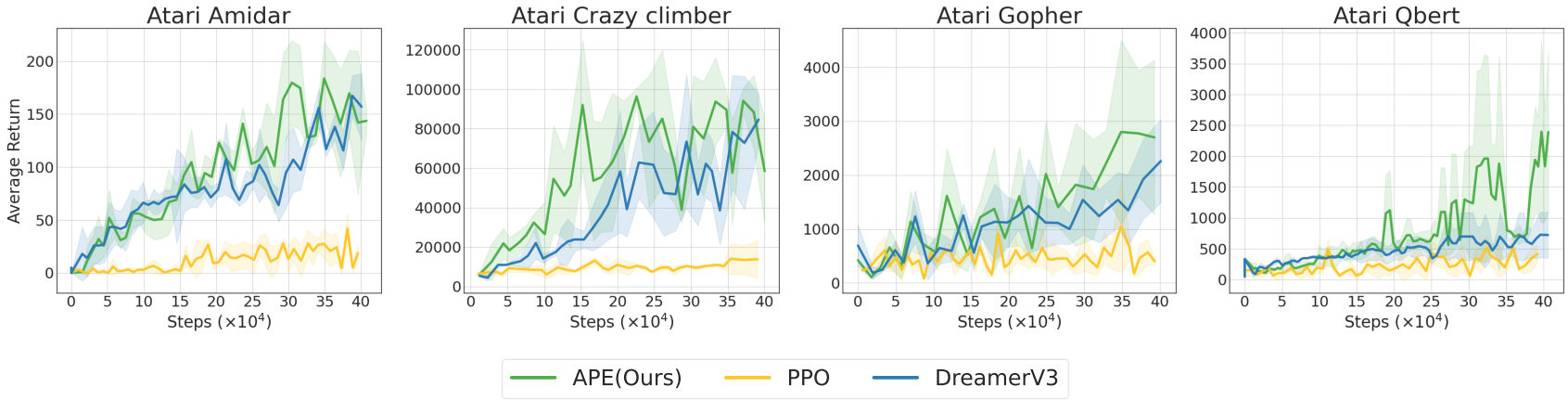


Figure: Training curves for Atari 100k benchmarks.

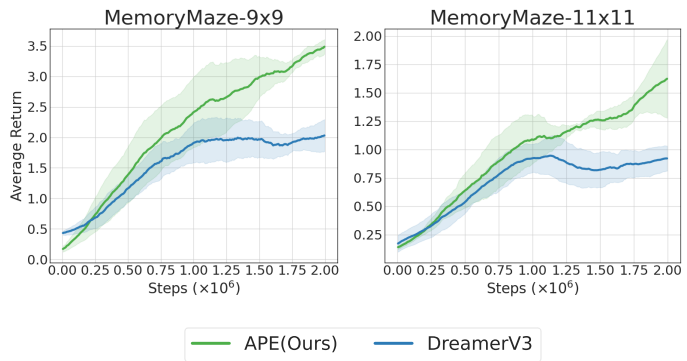


Figure: Training curves for Memory Maze benchmarks.

- **Robustness:** The encoder we trained before works right away on Atari games without needing to change it for each game
- **Generalization:** What we learned from real photos (ImageNet-100) still helps with Atari's simple game graphics because we can change how we

Comparison with Other Pretrained Algorithms (1/2)

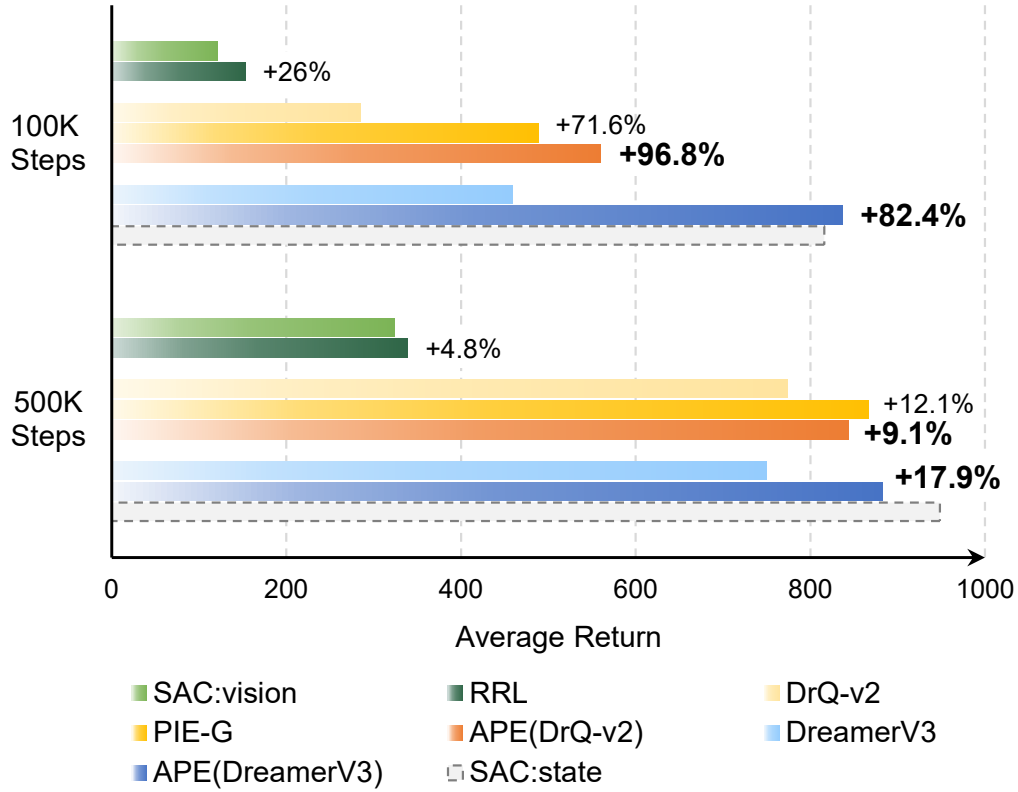


Figure: Comparison of DreamerV3-based and DrQ-v2-based APE against other ResNet pretrained algorithms.

Comparison with Other Pretrained Algorithms (2/2)

Models:

- **SAC:vision** and **DrQ-v2** are our "pure pixel" baselines—model-free methods that learn directly from raw images
- **RRL** and **PIE-G** are two prior works that take a ResNet pretrained on ImageNet (or similar) and plug it into an RL pipeline, either freezing or fine-tuning it
- **APE (DreamerV3)** is our model-based variant (main text)
- **APE (DrQ-v2)** is an additional run we did to confirm that any downstream algorithm—model-based or model-free—can benefit from APE's pretrained encoder
- **SAC:state** is an oracle agent that learns from the simulator's true, low-dimensional state (not pixels)—this represents a strong upper bound on sample efficiency

Results:

- **Performance at 100K/500K steps:**
  - APE significantly outperforms RRL, PIE-G, and pixel methods
  - At 100K steps, APE nearly matches SAC:state performance
  - By 500K steps, APE's advantage grows while pixel methods remain limited by poor features
- **Works Everywhere:**
  - APE works well with both model-free (DrQ-v2) and model-based (DreamerV3) methods
  - This shows that our pretraining approach is flexible and not tied to any specific learning method

Pretraining Does Work

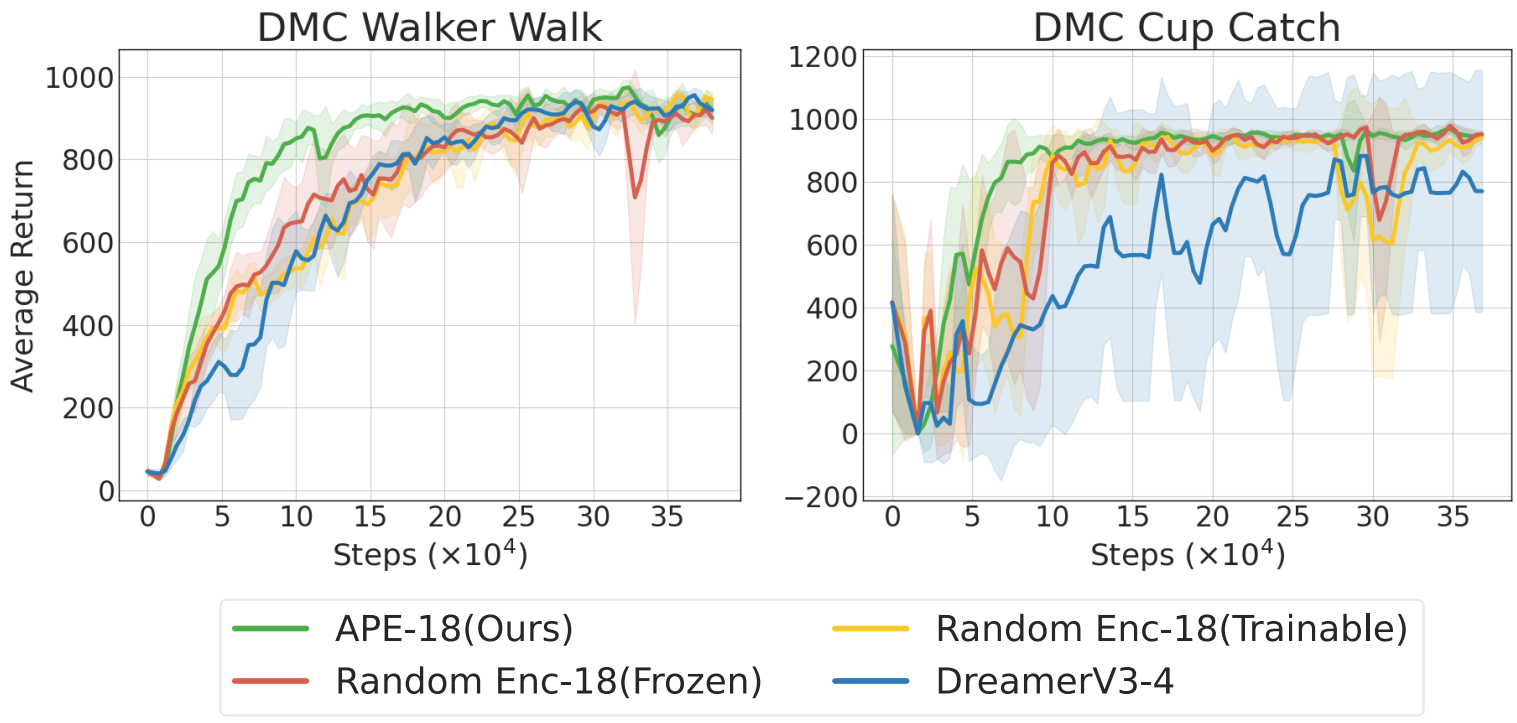


Figure: Choosing suitable pretraining strategy weighs more than increasing the depth of encoder network. We compare APE with random initialized encoder with frozen parameters, random initialized encoder with trainable parameters and DreamerV3. The last layer of the frozen random initialized encoder is finetuned during training. '-18' and '-4' denote the number of layers used in the encoder.

Augmentations Matter

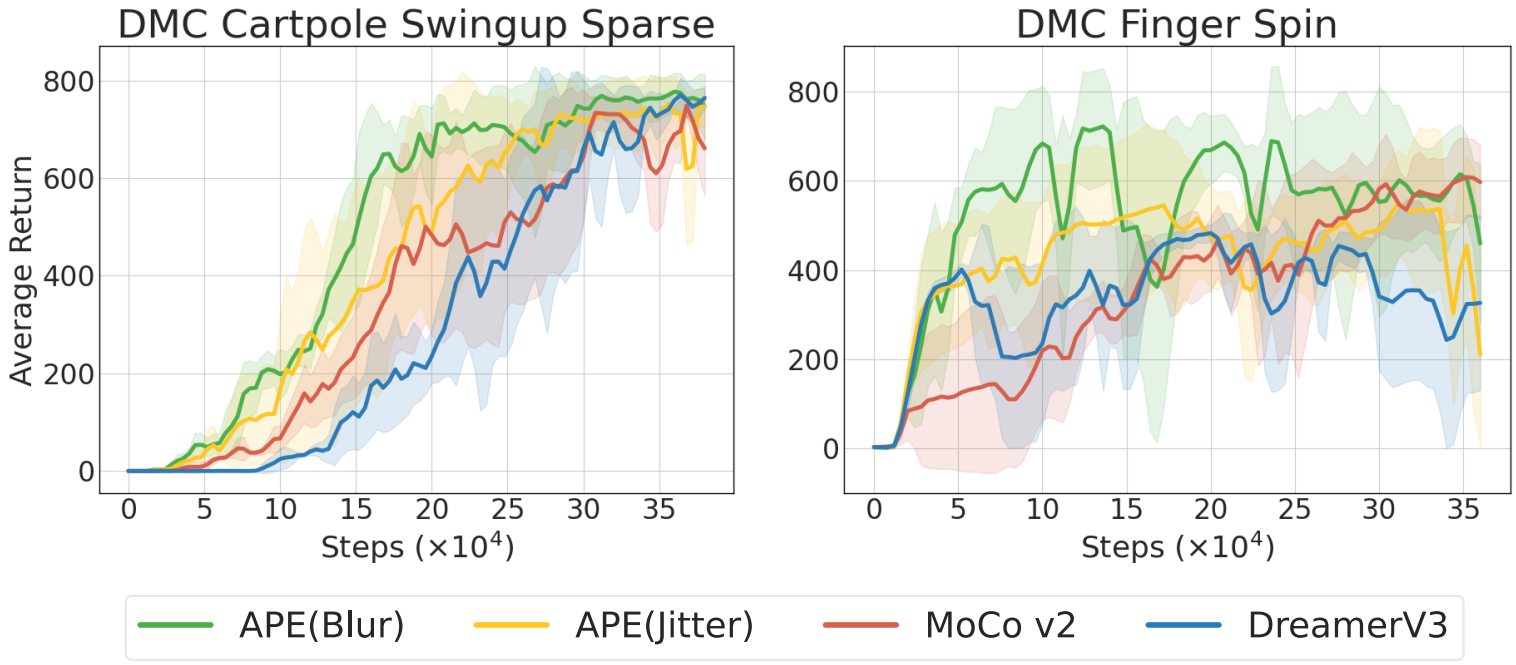


Figure: Different choices of augmentation strategy. APE with random gaussian blur as its main augmentation strategy outperforms other settings.

Different Choices of Network Architectures

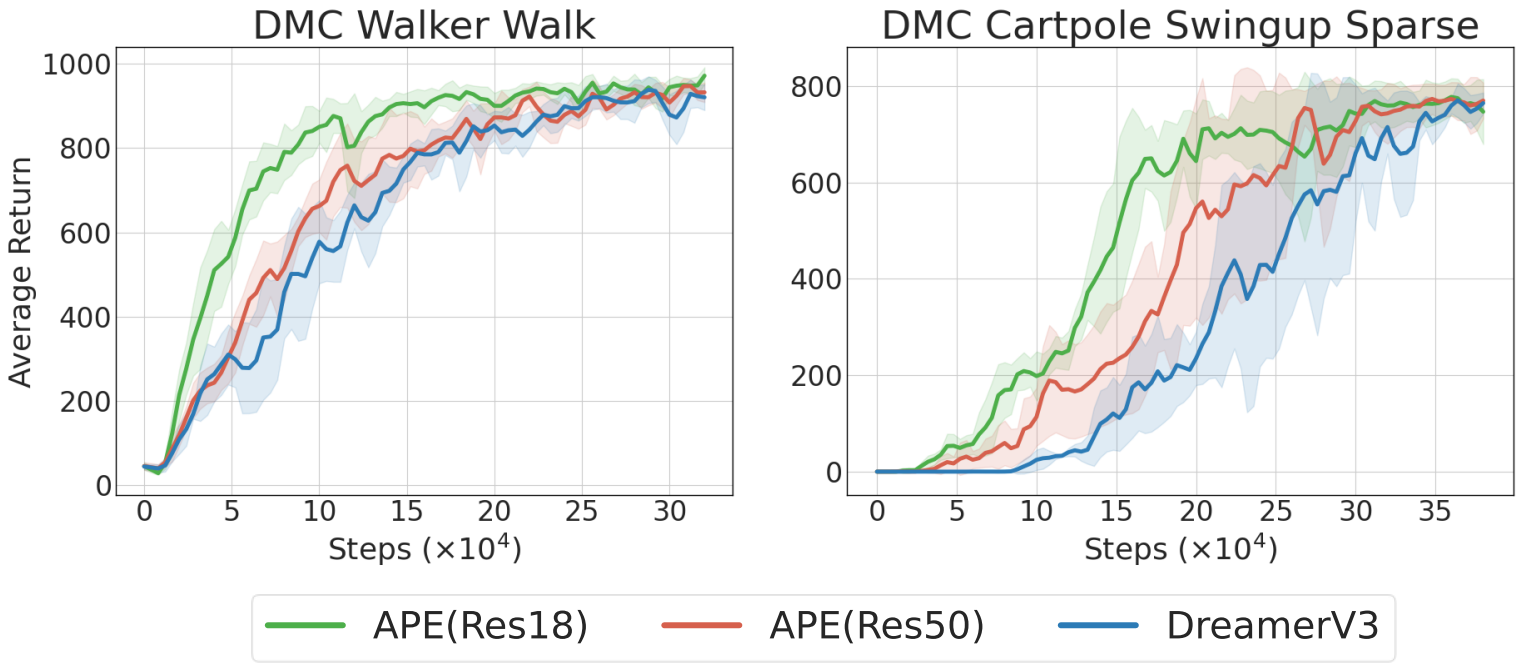


Figure: Different choices of network architectures. This figure indicates that APE with ResNet18 achieves better results compared with a deeper APE (ResNet50).