CtrlFormer: Learning Transferable State Representation for Visual Control via Transformer

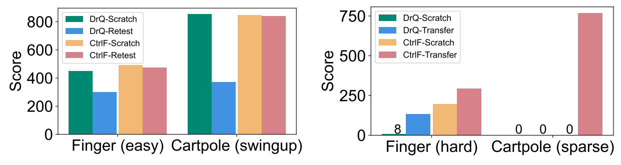
Yao Mu [1] Shoufa Chen [1] Mingyu Ding [1] Jianyu Chen [2] Runjian Chen [1] Ping Luo [1]

Abstract

Transformer has achieved great successes in learning vision and language representation, which is general across various downstream tasks. In visual control, learning transferable state representation that can transfer between different control tasks is important to reduce the training sample size. However, porting Transformer to sample-efficient visual control remains a challenging and unsolved problem. To this end, we propose a novel Control Transformer (CtrlFormer), possessing many appealing benefits that prior arts do not have. Firstly, CtrlFormer jointly learns self-attention mechanisms between visual tokens and policy tokens among different control tasks, where multitask representation can be learned and transferred without catastrophic forgetting. Secondly, we carefully design a contrastive reinforcement learning paradigm to train CtrlFormer, enabling it to achieve high sample efficiency, which is important in control problems. For example, in the DMControl benchmark, unlike recent advanced methods that failed by producing a zero score in the "Cartpole" task after transfer learning with $100k$ samples, CtrlFormer can achieve a state-of-the-art score $769_{\pm 34}$ with only $100k$ samples, while maintaining the performance of previous tasks. The code and models are released in our project homepage.

1. Introduction

Visual control is important for various real-world applications such as playing Atari games (Mnih et al., 2015), Go games (Silver et al., 2017), robotic control (Kober et al., 2013; Yuan et al., 2022), and autonomous driving (Wang et al., 2018). Although remarkable successes have been made, a long-standing goal of visual control, transferring the learned knowledge to new tasks without catastrophic forgetting, remains challenging and unsolved.



(a) Maintainability (b) Transferability

Figure 1. **Effect of CtrlFormer.** The agent first learns state representations in one task, and then transfers them to a new task, *e.g.*, from finger (turn-easy) to finger (turn-hard), and from cartpole (swingup) to cartpole (swingup-sparse). Figure **1a** shows the *maintainability* by comparing the performance in the old task before (scratch) and after (retest) transferring to a new task. CtrlFormer doesn't have catastrophic forgetting after transferring to the new task, and the performance is basically the same as learning from scratch. However, the performance of DrQ (Kostrikov et al., 2020) drops significantly after transferring; Figure **1b** shows the *transferability* by comparing the performance of learning from scratch and from transferring. Transferring previous learned knowledge benefits much for CtrlFormer (labeled as CtrlF).

Unlike a machine, a human can quickly identify the critical information to learn a new task with only a few actions and observations, since a human can discover the relevancy/irrelevancy between the current task and the previous tasks he/she has learned, and decides which information to keep or transfer. As a result, a new task can be learned quickly by a human without forgetting what has been learned before. Moreover, the "state representation" can be strengthened for better generalization to future tasks.

Modern machine learning methods for transferable representation learning across tasks can be generally categorized into three streams. They have certain limitations. In the first stream, many representative works such as (Shah & Kumar, 2021) utilize features pretrained in the ImageNet (Deng et al., 2009) and COCO (Lin et al., 2014) datasets. The domain gap between the pretraining datasets and the target datasets hinders their performance and sample efficiency. In

[1]Department of Computer Science, the University of Hong Kong, Hong Kong [2]Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing, China. Correspondence to: Ping Luo <pluo@cs.hku.hk>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

the second stream, Rusu et al. (2016); Fernando et al. (2017) trained a super network to accommodate a new task. These approaches often allocate different parts of the supernet to learn different tasks. However, the network parameters and computations are proportionally increased when the number of tasks increases. In the third stream, the latent variable models (Ha & Schmidhuber, 2018b; Hafner et al., 2019b;a) learn representation by optimizing the variational bound. These approaches are struggling when the tasks come from different domains.

Can we learn sample-efficient transferable state representation across different control tasks in a single Transformer network? The self-attention mechanism in Transformer mimics the perceived attention of synaptic connections in the human brain as argued in (Oby et al., 2019). Transformer could be powerful to model the relevancy/irrelevancy between different control tasks to alleviate the weaknesses in previous works. However, simply porting Transformer to this problem cannot solve the above limitations because Transformer is extremely sample-inefficient (data-hungry) as demonstrated in NLP (Vaswani et al., 2017; Devlin et al., 2018) and computer vision (Dosovitskiy et al., 2020).

This paper proposes a novel Transformer for representation learning in visual control, named CtrlFormer, which has two benefits compared to previous works. Firstly, the self-attention mechanism in CtrlFormer learns both visual tokens and policy tokens for multiple different control tasks simultaneously, fully capturing relevant/irrelevant features between tasks. This enables the knowledge learned from previous tasks can be transferred to a new task, while maintaining the learned representation of previous tasks. Secondly, CtrlFormer reduces training sample size by using contrastive reinforcement learning, where the gradients of the policy loss and the self-supervised contrastive loss are propagated jointly for representation learning. For example, as shown in Figure 2, the input image is divided into several patches, and each patch corresponds to a token. The CtrlFormer learns self-attentions between image tokens, as well as policy tokens of different control tasks such as "standing" and "walking". In this way, CtrlFormer not only decouples multiple control policies, but also decouples the features for behaviour learning and self-supervised visual representation learning, improving transferability among different tasks.

Our contributions are three-fold. **(1)** A novel control Transformer (CtrlFormer) is proposed for learning transferable state representation in visual control tasks. CtrlFormer models the relevancy/irrelevancy between distinct tasks by self-attention mechanism across visual data and control policies. It makes the knowledge learned from previous tasks transferable to a new task, while maintaining accuracy and high sample efficiency in previous tasks. **(2)** CtrlFormer can improve sample efficiency by combining reinforcement learning with self-supervised contrastive visual representation learning (He et al., 2020; Grill et al., 2020) and can reduce the number of parameters and computations via a pyramid Transformer structure. **(3)** Extensive experiments show that CtrlFormer outperforms previous works in terms of both transferability and sample efficiency. As shown in Figure 1, transferring previously learned state representation significantly improves the sample efficiency of learning new tasks. Furthermore, CtrlFormer does not have catastrophic forgetting after transferring to the new task, and the performance is basically the same as learning from scratch.

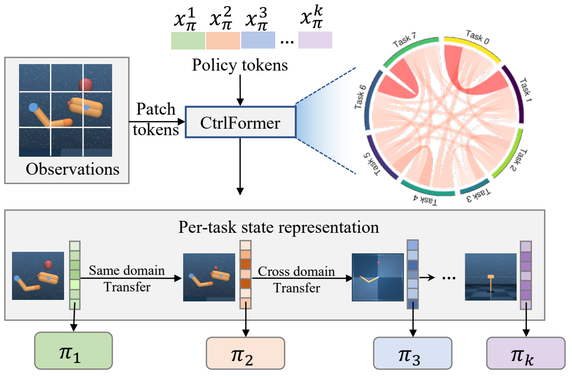


Figure 2. **Overview of CtrlFormer for visual control.** The input image is split into several patch tokens. Each task is assigned a specific policy token, which is a randomly-initialized and learnable variable. Tasks can come from the same domain, such as the Finger-turn-easy and Finger-turn-hard, or cross-domain, such as Reacher-easy and Cartpole-swingup. CtrlFormer learns the self-attention between observed image tokens, as well as policy tokens of different control tasks by vision transformer, which helps the agent leverage the similarities with previous tasks and reuse the representations from previously learned tasks to promote the behaviour learning of the current task. The output of CtrlFormer is used as the input of the downstream policy networks.

2. Related Work

Learning Transferable State Representation. For task-specific representation learning tasks like classification and contrastive objectives, the representation learned on large-scale offline datasets (ImageNet (Deng et al., 2009), COCO (Lin et al., 2014), and etc.) has high generalization ability (He et al., 2020; Yen-Chen et al., 2020). However, the downstream reinforcement learning methods based on such a task-agnostic representation empirically show low sample efficiency since the representation contains a lot of task-irrelevant interference information.

Progressive neural network (Rusu et al., 2016; 2017; Gideon et al., 2017) is a representative structure of transferring state representations, which is composed of multiple columns, where each column is a policy network for a specific task, and lateral connections are added. The parameters need to learn grow proportionally with the number of incoming tasks, which hinders its scalability.

PathNet (Fernando et al., 2017) tries to alleviate this issue by using a size-fixed network. It contains multiple pathways, which are subsets of neurons whose weights contain the knowledge of previous tasks and are frozen during training on new tasks. The pathways that determine which parts of the network could be re-used for new tasks are discovered by a tournament selection genetic algorithm. PathNet fixes the parameters along a path learned on previously learned tasks and re-evolves a new population of paths for a new task to accelerate the behavior learning. However, the process of pathways discovery with the genetic algorithm has a high cost on the computational resources.

Latent variable models offer a flexible way to represent key information of the observations by optimizing the variational lower bound (Krishnan et al., 2015; Karl et al., 2016; Doerr et al., 2018; Buesing et al., 2018; Ha & Schmidhuber, 2018b; Hafner et al., 2019b;a; Tirinzoni et al., 2020; Mu et al., 2021; Chen et al., 2022b). Ha & Schmidhuber (2018a) propose the world model algorithm to learn representation by variational autoencoder (VAE). PlaNet (Hafner et al., 2019b) utilizes a recurrent stochastic state model (RSSM) to learn the representation and latent dynamic jointly. The transition probability is modeled on the latent space instead of the original state space. Dreamer (Hafner et al., 2019a) utilizes the RSSM to make the long-term imagination. Although Dreamer is promising to transfer the knowledge across tasks that share the same dynamics, it is still challenging to transfer among tasks across domains.

Vision Transformer. With the great successes of Transformers (Vaswani et al., 2017) in NLP (Devlin et al., 2018; Radford et al., 2015), people apply them to solve computer vision problems. ViT (Dosovitskiy et al., 2020) is the first pure Transformer model introduced into the vision community and surpasses CNNs with large scale pretraining on the private JFT dataset (Riquelme et al., 2021). DeiT (Touvron et al., 2021) trains ViT from scratch on ImageNet-1K (Deng et al., 2009) and achieves better performance than CNN counterparts. Pyramid ViT (PVT) (Wang et al., 2021) is the first hierarchical design for ViT, and proposes a progressive shrinking pyramid and spatial-reduction attention. Swin Transformer (Liu et al., 2021) computes attention within a local window and adopts shifted windows for communication aggregation. More recently, efficient transfer learning is also explored in for vision Transformer (Bahng et al., 2022; Jia et al., 2022; Chen et al., 2022a). In this paper, we take the original ViT (Dosovitskiy et al., 2020) as the visual backbone with simple pooling layers, which are used to reduce the calculation burden, and more advanced structures may bring further gain.

3. Preliminaries

Overview of Vision Transformer. The original Transformer (Vaswani et al., 2017) tasks as input a 1D sequence of token embeddings. To handle 2D images, ViT (Dosovitskiy et al., 2020) splits an input image $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ to a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N\times(P^2\cdot C)}$, where we let H and W denote the height and width of the image, C the number of channels, (P, P) the resolution of each image patch, and $N = \frac{HW}{P^2}$ is the number of flattened patches. After obtaining $\mathbf{x}_p$, ViT map it to D dimensions with a trainable linear projection and uses this constant latent vector size D through all of its layers. The output of this projection is named the patch embeddings.

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\cdots;\, \mathbf{x}_p^N\mathbf{E}] \tag{1}$$

where $\mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D}$ is the projection matrix. $\mathbf{x}_{\text{class}}$ denotes the class token. The Transformer block (Vaswani et al., 2017) includes alternating layers of multiheaded self-attention (MHSA) and MLP blocks. Besides, Layernorm (LN) is applied before every block, and residual connections after every block. The MLP utilizes two layers with a GELU (Hendrycks & Gimpel, 2016) non-linearity. This process can be formulated as:

$$\begin{aligned} \mathbf{z}'_\ell &= \text{MHSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1\ldots L \\ \mathbf{z}_\ell &= \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1\ldots L \\ \mathbf{y} &= \text{LN}(\mathbf{z}_L^0)) \end{aligned} \tag{2}$$

where $\mathbf{y}$ denotes the image representation, which is encoded by the output of the class token at the last block.

To build up a Transformer block, an MLP (Popescu et al., 2009) block with two linear transformations and GELU (Hendrycks & Gimpel, 2016) activation are usually adopted to provide nonlinearity. Note that the dimensions of the parameter matrix will not change when the number of tokens increases, which is a key advantage of the Transformer handling variable-length inputs.

Reinforcement Learning for Visual Control. Reinforcement learning for visual control aims to learn the optimal policy given the observed images, and could be formulated as an infinite-horizon partially observable Markov decision process (POMDP) (Bellman, 1957; Kaelbling et al., 1998). POMDP can be denoted by $\mathcal{M} = \langle \mathcal{O}, \mathcal{A}, \mathcal{P}, p_0, r, \gamma \rangle$, where $\mathcal{O}$ is the high-dimensional observation space (*i.e.*, image pixels), $\mathcal{A}$ is the action space, $\mathcal{P} = Pr(o_{t+1}|o_{\leq t}, a_t)$ represents the probability distribution over the next observation o_{t+1} given the history of previous observations $o_{\leq t}$ and the current action a_t and p_0 is the distribution of initial state. $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ is the reward function that maps the current observation and action to a scalar representing the reward, $r_t = r(o_{\leq t}, a_t)$. The overall objective is to find the optimal policy π^* to maximize the cumulative discounted return

$E_\pi[\sum_{t=0}^{\infty}\gamma^t r_t | a_t \sim \pi(\cdot|s_{\le t}), s_{t+1} \sim p(\cdot|s_{\le t}, a_t), s_0 \sim p_0(\cdot)]$, where $\gamma \in [0, 1)$ is the discount factor, which is applied to pay more attention on recent rewards rather than future ones and is usually set to 0.99 in practice.

By stacking several consecutive image observations into a state, $s_t = \{o_t, o_{t-1}, o_{t-2}, \ldots\}$, the POMDP could be converted into an Markov Decision Process (MDP) (Bellman, 1957), where information at the next time-step is determined by the information at current step, unrelated to those in the history. Thus, for a MDP process, the transition dynamics can be refined as $p = Pr(s'_t|s_t, a_t)$ representing the distribution of next state s'_t given the current state s_t and action a_t, and the reward function is refined as $r_t = r(s_t, a_t)$ similarly. In practice, three consecutive images are stacked into a state s_t and as an MDP, the objective turns into finding the optimal policy, $\pi^*(a_t|s_t)$ to maximize the expected return.

4. Method

In this section, we introduce our CtrlFormer for visual control in details. As shown in Figure 2, the observation is first split into N patches and mapped to N tokens $[\mathbf{x}_p^1; \cdots; \mathbf{x}_p^N]$. Then CtrlFormer takes as inputs the image patches with an contrastive token $\mathbf{x}_{\text{con}}$ to improve the sample efficiency and K policy tokens $[\mathbf{x}_\pi^1; \cdots; \mathbf{x}_\pi^K]$ and interactively encodes them with self-attention mechanism, leading to representations $[\mathbf{z}_{\text{con}}; \mathbf{z}_\pi^1; \cdots; \mathbf{z}_\pi^K; \mathbf{z}_p^1; \cdots; \mathbf{z}_p^N]$. Each task is assigned a policy token x_π^i, which is a randomly-initialized but learnable variable similar to the class token in conventional vision transformers. In training, the policy token learns to abstract the characteristic of the corresponding task and the correlations across tasks via gradient back-propagation. In inference, it serves as the query to progressively gather useful information from visual inputs and previously learned tasks through self-attention layers.

To train the CtrlFormer encoder, the representation of each policy token, $\mathbf{z}_\pi^i$ is utilized as the input of the task-specific policy network and Q-network in the downstream reinforcement learning algorithm, and the goal is to maximize the expected return for each task. The data-regularized method is used to reduce the variance of Q-learning and improve the robustness of the change of representation. Besides, a contrastive objective is applied on $\mathbf{z}_{\text{con}}$ to aid the training process, which significantly improves sample efficiency. The total loss is the sum of the reinforcement learning part and the contrastive learning part in equal proportion.

In Section 4.1, we first introduce the details in CtrlFormer, and then a discussion is presented on how to transfer the learned representations to a new task in Section 4.2. The two objectives for training CtrlFormer, i.e., the policy learning problem and contrastive learning, are discussed in detail respectively in Section 4.3 and 4.4.

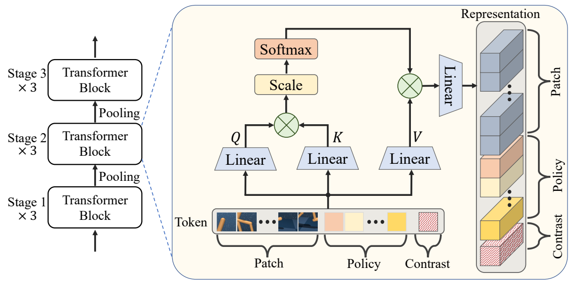


Figure 3. **The structure of CtrlFormer.** CtrlFormer has a pyramid structure consisting of 3 stages and each stage has 3 blocks. The input image is split into several patches and these patches are mapped to a sequence of *patch* tokens. The number of patch tokens is reduced to half by pooling operation between two stages. Each task has an independent *policy* token. The number of policy tokens maintains through 3 stages. All tasks shares an additional *constractive* token. The representations are learned by the self-attention mechanism. The output of the policy token is used for downstream reinforcement learning and the output of the contrastive token is used for contrastive learning.

4.1. The Architecture of CtrlFormer

In CtrlFormer, as shown in Figure 3, each task has independent `[policy]` token $\mathbf{x}_\pi^i$, which is similar to `[class]` token in ViT (Dosovitskiy et al., 2020). Three consecutive frames of images are stacked into a 9-channel input tensor $\mathbf{x}_p \in \mathbb{R}^{H \times W \times 9}$. We split the input tensor into $N = 9HW/P^2$ patches with patch size $P \times P$ and then map it to a sequence of vectors $\{\mathbf{x}_p^1, \mathbf{x}_p^2, \ldots, \mathbf{x}_p^N\}$. Contrastive learning is learned together with reinforcement learning as an auxiliary task to improve the sample efficiency, which is assigned to an `[contrastive]` token $\mathbf{x}_{\text{con}}$. Position embeddings $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{N \times D}$, which is the same as those used in (Dosovitskiy et al., 2020), are added to the patch embeddings $\{\mathbf{x}_p^i\}_{i=1}^N$ to retain positional information.

Thus, the input of the transformer is

$$\mathbf{z}_{\ell_0} = \left[\mathbf{x}_{\text{con}}; \mathbf{x}_\pi^1; \ldots; \mathbf{x}_\pi^K; \mathbf{x}_p^1; \cdots; \mathbf{x}_p^N\right] + \mathbf{E}_{\text{pos}} \quad (3)$$

The blocks with multi-head self-attention (MHSA) (Vaswani et al., 2017) and layer normalization (LN) (Ba et al., 2016) could be formulated as

$$\begin{aligned} \mathbf{z}'_{\ell_j} &= \text{MHSA}\left(\text{LN}\left(\mathbf{z}_{\ell_{j-1}}\right)\right) + \mathbf{z}_{\ell_{j-1}} \\ \mathbf{z}_{\ell_j} &= \text{MLP}\left(\text{LN}\left(z'_{\ell_j}\right)\right) + \mathbf{z}'_{\ell_j} \end{aligned} \quad (4)$$

where $\mathbf{z}_{l_0}$ is the input of transformer and $\mathbf{z}_{l_j}$ is the output of the j-th block. To reduce the number of parameters needed to learn, we utilize a pyramidal structure. There are three stages in the pyramidal vision transformer, and the number of tokens decreases with the stages by a pooling layer. In the pooling layer, the token sequence with a length of N is reshaped into $(H/P) \times (W/P)$ and is pooled by a 2×2 filter with stride $(2, 1)$ in the first stage and with stride $(1, 2)$

in the second stage. After pooling, the tensor is flattened back to the token sequence with a length of N to serve as the input of the next stage. More detailed structure of CtrlFormer is introduced in Figure 8 in Appendix B.2.

4.2. Representation Transferring in New Tasks

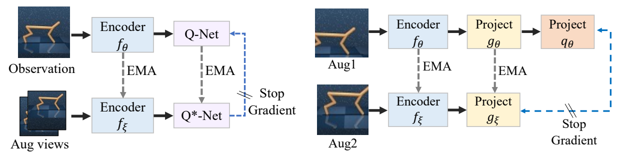
After learning the K-th previous task with policy token $\mathbf{z}_\pi^K$, we pad a new policy token $\mathbf{z}_\pi^{K+1}$, and the new task is learned with policy token $\mathbf{z}_\pi^{K+1}$, policy network $\pi^{K+1}(\cdot)$ and Q-networks $Q_1^{K+1}(\cdot,\cdot)$ and $Q_2^{K+1}(\cdot,\cdot)$. CtrlFormer inherits the merit of the transformer model to resolve variable-length input sequences. The dimension of the CtrlFormer's model parameters remains unchanged when the number of policy tokens changes, *i.e.*, the weight dimension in self-attention is only related to the dimension of the token rather than the number of tokens. The parameter metrics $\mathbf{W}^q \in \mathbb{R}^{D\times D}$,$\mathbf{W}^k \in \mathbb{R}^{D\times D}$ and $\mathbf{W}^v \in \mathbb{R}^{D\times D}$ are all remain its original dimension. When transferring to the K+1 task, the number of input tokens is increased from N+K+1 to N+K+2, hence N+K+2 output representations. Thus there is always a one-to-one correspondence between policy tokens and output representations. Both old and new tasks can be tackled within CtrlFormer. The policy query $\mathbf{q}_\pi \in \mathbb{R}^{(K+1)\times D}$ and the key $\mathbf{k} \in \mathbb{R}^{(N+K+2)\times D}$ are calculated by

$$\begin{aligned}
\mathbf{q}_\pi &= \mathbf{z}_\pi \mathbf{W}^q, \mathbf{k} = \mathbf{z}\mathbf{W}^k \\
\mathbf{z} &= \left[\mathbf{z}_{con}; \mathbf{z}_\pi^1; \ldots; \mathbf{z}_\pi^{K+1}; \mathbf{z}_p^1; \ldots; \mathbf{z}_p^N\right] \\
\mathbf{z}_\pi &= \left[\mathbf{z}_\pi^1; \ldots; \mathbf{z}_\pi^{K+1}\right]
\end{aligned} \tag{5}$$

4.3. Downstream Visual Reinforcement Learning

With the output $\mathbf{z} = [\mathbf{z}_{\text{con}}; \mathbf{z}_\pi^1; \ldots; \mathbf{z}_\pi^K; \mathbf{z}_p^1; \ldots; \mathbf{z}_p^N]$ of the CtrlFormer, we take $\mathbf{z}_\pi^i$ as the representation for the i-th task, which contains the task-relevant information gathered from all image tokens and policy tokens through the self-attention mechanism. We utilize SAC (Haarnoja et al., 2018a) as the downstream reinforcement learning algorithm to solve the optimal policy with the representation $\mathbf{z}_\pi^i$ for the i-th task, which is an off-policy RL algorithm that optimizes a stochastic policy for maximizing the expected trajectory returns. SAC learns a stochastic policy π_ψ and critics Q_{ϕ_1} and Q_{ϕ_2} to optimize a γ-discounted maximum-entropy objective (Ziebart et al., 2008). We use the same structure of the downstream Q-network and policy network as DrQ (Kostrikov et al., 2020). More details about SAC are introduced in Appendix A and the detailed structures of Q-network and policy network are introduced in Appendix B.2. The parameters ϕ_j are learned by minimizing the Bellman error:

$$\mathcal{L}(\phi_j, \mathcal{B}) = \mathbb{E}_{t\sim\mathcal{B}}\left[\left(Q_{\phi_j}(\mathbf{z}_\pi^i, a) - (r + \gamma(1-d)\mathcal{T})\right)^2\right] \tag{6}$$



(a) Reinforcement learning. (b) Contrastive learning.

Figure 4. **Downstream reinforcement learning with contrastive learning co-training.** Both reinforcement learning and contrastive learning are based on a set of momentum-updated learning frameworks, including online encoder f_θ, Q-networks $Q(\cdot,\cdot)$ and online projector g_θ, and target encoder f_ξ, target Q-network $Q^*(\cdot,\cdot)$ and target projector g_ξ, which are updated by EMA method. In the RL part, the observation is firstly encoded by the online encoder as the input of $Q(\cdot,\cdot)$. Different augmented views of the observation are encoded by the target encoder f_ξ as the input of the $Q^*(\cdot,\cdot)$. The target Q-value is the mean of the value calculated by different views. The encoder and Q-networks are updated by the Critic loss calculated by the estimated Q-value and target Q-value. The objective of contrastive learning is to predict the projected representation from one augmented view of observation with the projected representation from another augmented view.

where $(\mathbf{z}_\pi^i, a, \mathbf{z'}_\pi^i, r, d)$ is a tuple with current latent state z_π^i, action a, next latent state $\mathbf{z'}_\pi^i$, reward r and done signal d, $\mathcal{B}$ is the replay buffer, and $\mathcal{T}$ is the target, defined as:

$$\mathcal{T} = \left(\min_{i=1,2} Q^*_{\phi_j}(\mathbf{z'}_\pi^i, a') - \alpha \log \pi_\psi(a'|\mathbf{z'}_\pi^i)\right) \tag{7}$$

In the target equation 7, $Q^*_{\phi_j}$ denotes the exponential moving average (EMA) (Holt, 2004) of the parameters of Q_{ϕ_j}. Using the EMA has empirically shown to improve training stability in off-policy RL algorithms (Mnih et al., 2015). The parameter α is a positive entropy coefficient that determines the priority of the entropy maximization during the policy optimization.

As shown in Figure 4a, to improve the robustness of the policy network and Q-network and further reduce the variance of Q-learning, we regularize the Q-target function $Q^*_{\phi_j}(\cdot,\cdot)$ by data augmentation as shown in equation 8

$$Q^*_{\phi_j}\left(\mathbf{z}_\pi^i, a\right) = \frac{1}{K}\sum_{m=1}^{K} Q^*_{\phi_j}\left(\mathbf{z}_\pi^{i,m}, a'_m\right) \tag{8}$$

where $\mathbf{z}_\pi^{i,m}$ is encoded by a random augmented image input, $a_m \sim \pi\left(\cdot \mid \mathbf{z}_\pi^{i,m}\right)$. Such a Q-regularization method allows the generation of several surrogate observations with the same Q-values, thus providing a mechanism to reduce the variance of Q-function estimation.

While the critic is given by Q_{ϕ_j}, the actor samples actions from policy π_ψ and is trained by maximizing the expected return of its actions as in:

$$\mathcal{L}(\psi) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(\mathbf{z}_\pi^i, a) - \alpha \log \pi_\psi(a|\mathbf{z}_\pi^i)\right] \tag{9}$$

4.4. Contrastive Learning for Efficient Downstream RL

Transformers are empirically proven hard to perform well when trained on insufficient amounts of data. However, reinforcement learning aims to learn behaviour with as little as possible interaction data, which is considered expensive to collect. DeiT (Touvron et al., 2021) introduces a teacher-student strategy for data-efficient transformer training, which relies on an auxiliary distillation token, ensuring the student learns from the teacher through attention. By referencing this idea, we propose a sample-efficient co-training method with contrastive learning for transformer in reinforcement learning. The goal of the contrastive task is to learn a representation $\mathbf{z}_{\text{con}}$, which can then be used for downstream tasks. As shown in Figure 4b, we use a momentum learning framework, which contains an online network and a target network that is updated by the exponential moving average (EMA) method. The online network is defined by a set of weights θ and is comprised of three stages: an encoder f_θ, a projector g_θ, a predictor q_θ. The target network, which is defined by a set of weights ξ, provides the regression targets to train the online network. As shown in Figure 4b, the observation o is randomly augmented by two different image augmentation t and t' respectively from two distributions of image augmentations $\mathcal{T}$ and $\mathcal{T}'$. Two augmented views $v \triangleq t(o)$ and $v' \triangleq t'(o)$ are produced from o by applying respectively image augmentations $t \sim \mathcal{T}$ and $t' \sim \mathcal{T}'$. The details of the image augmentation implementation is introduced in AppendixB.5.

From the first augmented view v, the online network outputs a representation $\mathbf{z}_{\text{con}} \triangleq f_\theta(v)$ by the vision transformer and a projection $y_\theta \triangleq g_\theta(\mathbf{z}_{\text{con}})$. The target network outputs $z'_{\text{con}_\xi} \triangleq f_\xi(v')$ and the target projection $y'_\xi \triangleq g_\xi(z'_{\text{con}_\xi})$ from the second augmented view v', which are all stop-gradient. We then output a prediction $q_\theta(y_\theta)$ of y'_ξ and ℓ_2-normalize both $q_\theta(y_\theta)$ and y'_ξ to $\overline{q_\theta}(y_\theta) \triangleq q_\theta(y_\theta) / \|q_\theta(y_\theta)\|_2$ and $\bar{y}'_\xi \triangleq y'_\xi / \left\|y'_\xi\right\|_2$. The contrastive objective is defined by the mean squared error between the normalized predictions and target projections,

$$\mathcal{L}_{\theta,\xi} \triangleq \left\|\overline{q_\theta}(y_\theta) - \bar{y}'_\xi\right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(y_\theta), y'_\xi \rangle}{\|q_\theta(y_\theta)\|_2 \cdot \left\|y'_\xi\right\|_2}. \quad (10)$$

The updates of online network and target network are summarized as

$$\begin{aligned} \theta &\leftarrow \text{optimizer}(\theta, \nabla_\theta \mathcal{L}_{\theta,\xi}, \eta), \\ \xi &\leftarrow \tau\xi + (1-\tau)\theta, \end{aligned} \quad (11)$$

where optimizer is an optimizer and η is a learning rate.

5. Experiments

In this section, we evaluate our proposed CtrlFormer on multiple domains in DMControl benchmark (Tassa et al., 2018). We test the transferability among the tasks in the same domain and the tasks across different domains. Throughout these experiments, the encoder, actor, and critic neural networks are trained using the Adamw optimizer (Loshchilov & Hutter, 2017) with the learning rate $lr = 10^{-4}$ and a mini-batch size of 512. The soft target update rate τ of the critic is 0.01, and target network updates are made every 2 critic updates (same as in DrQ (Kostrikov et al., 2020)). The full set of parameters is in Appendix B.4. The Pytorch-like pseudo-code is provided in Appendix B.8.

Domain	Task 1	Task 2
Walker	`stand`	`walk`
Cartpole	`swingup`	`swingup-sparse`
Reacher	`easy`	`hard`
Finger	`turn-easy`	`hard`

Table 1. Domains and the corresponding tasks.

5.1. Benchmark

The DeepMind control suite is a set of continuous control tasks and has been widely used as the benchmark for visual control (Tassa et al., 2018). We mainly test the performance of CtrlFormer in 4 typical domains from DeepMind control suite. The dimensions of action space are the same for tasks within the same domain and different across domains. In order to evaluate the performance of CtrlFormer among the tasks in same-domain and cross-domain scenarios, we conduct extensive experiments on multiple domains and tasks, as shown in Table 1. The detailed introduction of the domains and tasks we use is in Appendix B.9.

5.2. Baselines

We mainly compare CtrlFormer with Dreamer (Hafner et al., 2019a), DrQ (Kostrikov et al., 2020) and SAC (Haarnoja et al., 2018b) whose the representation encoded by a pre-trained ResNet (He et al., 2016). Dreamer is a representative method with a latent variable model for state representation transferring, achieving the state-of-the-art performance for model-based reinforcement learning. To extract useful information from historical observations, it encodes the representation by a recurrent state space model (RSSM) (Hafner et al., 2019b). DrQ is the state-of-the-art model-free algorithm on DMControl tasks, which surpasses other model-free methods with task-specific representations (*i.e.*, methods update the encoder with the actor-critic gradients), such as CURL (Laskin et al., 2020) and SAC-AutoEncoder (Yarats et al., 2019). To test the transferability of DrQ, every task is assigned a specific head for the Q-network and the policy network, and all tasks share a CNN network to extract state representations. We utilize ResNet-50 (He et al., 2016) as encoder and pre-trained it on the ImageNet (Deng et al., 2009) to test the performance of task-agnostic representation for reinforcement learning.

Method	Learn from scratch		Retest after
	100k	500k	new task fine-tune
DrQ	$549_{\pm 36}$	$\mathbf{854}_{\pm 22}$	$373_{\pm 24}$
Dreamer	$326_{\pm 27}$	$762_{\pm 27}$	$704_{\pm 33}$
Resnet+SAC	$192_{\pm 19}$	$357_{\pm 85}$	$357_{\pm 85}$
CtrlFormer	$\mathbf{759}_{\pm 48}$	$846_{\pm 25}$	$\mathbf{842}_{\pm 22}$

Method	Learn from scratch		Learn with transfer	
	100k	500k	100k	500k
DrQ	0	$505_{\pm 335}$	0	$76_{\pm 41}$
Dreamer	$8_{\pm 4}$	$376_{\pm 214}$	0	$589_{\pm 122}$
Resnet+SAC	0	0	0	0
CtrlFormer	0	$\mathbf{671}_{\pm 81}$	$\mathbf{769}_{\pm 34}$	$\mathbf{804}_{\pm 26}$

(a) **Left:** Learn old task in Cartpole (`swingup`) **Right:** Transfer to new task Cartpole (`swingup-sparse`)

Method	Learn from scratch		Retest after
	100k	500k	new task fine-tune
DrQ	$\mathbf{346}_{\pm 33}$	$448_{\pm 65}$	$300_{\pm 42}$
Dreamer	$25_{\pm 18}$	$245_{\pm 159}$	$182_{\pm 34}$
Resnet+SAC	$298_{\pm 17}$	$300_{\pm 29}$	$300_{\pm 29}$
CtrlFormer	$281_{\pm 67}$	$\mathbf{493}_{\pm 35}$	$\mathbf{475}_{\pm 43}$

Method	Learn from scratch		Learn with transfer	
	100k	500k	100k	500k
DrQ	$8_{\pm 24}$	$274_{\pm 137}$	$133_{\pm 26}$	$455_{\pm 34}$
Dreamer	0	$17_{\pm 9}$	0	$38_{\pm 18}$
Resnet+SAC	0	$17_{\pm 10}$	0	$17_{\pm 10}$
CtrlFormer	$\mathbf{197}_{\pm 78}$	$\mathbf{344}_{\pm 47}$	$\mathbf{294}_{\pm 37}$	$\mathbf{569}_{\pm 32}$

(b) **Left:** Learn old task in Finger (`turn-easy`) **Right:** Transfer to new task Finger (`turn-hard`)

Method	Learn from scratch		Retest after
	100k	500k	new task fine-tune
DrQ	$558_{\pm 38}$	$971_{\pm 27}$	$243_{\pm 52}$
Dreamer	$314_{\pm 155}$	$793_{\pm 164}$	$485_{\pm 67}$
Resnet+SAC	$322_{\pm 285}$	$382_{\pm 299}$	$382_{\pm 299}$
CtrlFormer	$\mathbf{642}_{\pm 42}$	$\mathbf{973}_{\pm 53}$	$\mathbf{906}_{\pm 31}$

Method	Learn from scratch		Learn with transfer	
	100k	500k	100k	500k
DrQ	$\mathbf{194}_{\pm 84}$	$\mathbf{616}_{\pm 274}$	$96_{\pm 43}$	$524_{\pm 68}$
Dreamer	$13_{\pm 32}$	$115_{\pm 98}$	$63_{\pm 07}$	$148_{\pm 12}$
Resnet+SAC	$26_{\pm 4}$	$31_{\pm 12}$	$26_{\pm 4}$	$31_{\pm 12}$
CtrlFormer	104 ± 48	$548_{\pm 131}$	$\mathbf{147}_{\pm 44}$	$\mathbf{657}_{\pm 68}$

(c) **Left:** Learning old task in Reacher (`easy`) **Right:** Transfer to new task Reacher (`hard`)

Method	Learn from scratch		Retest after
	100k	500k	new task fine-tune
DrQ	$875_{\pm 76}$	$973_{\pm 65}$	$698_{\pm 57}$
Dreamer	$583_{\pm 21}$	$\mathbf{974}_{\pm 31}$	$912_{\pm 19}$
Resnet+SAC	$177_{\pm 32}$	$190_{\pm 24}$	$190_{\pm 24}$
CtrlFormer	$\mathbf{877}_{\pm 42}$	$954_{\pm 38}$	$\mathbf{950}_{\pm 42}$

Method	Learn from scratch		Learn with transfer	
	100k	500k	100k	500k
DrQ	$504_{\pm 191}$	$\mathbf{947}_{\pm 101}$	$321_{\pm 54}$	$947_{\pm 36}$
Dreamer	$277_{\pm 12}$	$897_{\pm 49}$	$851_{\pm 44}$	$949_{\pm 22}$
Resnet+SAC	$63_{\pm 7}$	$148_{\pm 12}$	$63_{\pm 7}$	$148_{\pm 12}$
CtrlFormer	$\mathbf{593}_{\pm 52}$	$903_{\pm 43}$	$\mathbf{857}_{\pm 47}$	$\mathbf{959}_{\pm 42}$

(d) **Left:** Learn old task in Walker (`stand`) **Right:** Transfer to new task Walker (`walk`)

Table 2. **Transferring the state representation among tasks under same domain.** We list the sample efficiency of learning from scratch in both the previous task and the score retested after the new task fine-tuning in the sub-tables on the *left side*, and the comparison between learning new task from scratch and learning new task with transfer in the sub-tables on the *right side*.

5.3. Transferability

Settings. In all transferability testing experiments, the agent first learns a previous task, then adds a new policy token, uses the previous policy token as the initialization of the current task policy token, and then starts learning a new task. Finally, we retested the score (average episode return) of the old task using the latest Encoder and the Actor and Critic networks corresponding to the old task after learning the new task.

Transferring in the same domain. In the same domain, the dynamics of the agents from different tasks are similar, while the sizes of the target points and the characteristic of reward (sparse or dense) might be different. We design an experimental pipeline to let the agent first learn an easier task and then transfer the obtained knowledge to a harder one in the same domain. The results are summarized in Table 2. Compared to baselines, the sample efficiency of CtrlFormer in the new task is improved significantly after transferring the state representation from the previous task. Taking Table 2a and Figure 1b as examples, CtrlFormer achieves an average episode return of $769_{\pm 34}$ with representation transferring using 100k samples, while learning from scratch is totally failed using the same number of samples. Furthermore, when retesting on the previous task after transferring to new task, as shown in Figure 1a, CtrlFormer does not show an obvious decrease. However, the previous task's performance of DrQ is damaged significantly after learning the new task. Moreover, When learning from scratch, the sample efficiency of CtrlFormer is also comparable to the DrQ with multi-heads. Although Dreamer shows promising transferability in the domain like Walker, where different tasks share the same dynamics, it transfers poorly in domains like Reacher, where the dynamics are different among tasks. The representation encoded by a pre-trained ResNet-50 shows same performance among tasks but shows lower sample efficiency compared with CtrlFormer and DrQ.

Transferring across domains. Compared to the tasks in the same domain whose learning difficulty is readily defined in the DMControl benchmark, it is not easy to distinguish which task is easier to learn for tasks from different domains. Thus, we test the bi-directional transferability in different domains, *i.e.*, transferring from Finger domain to Reacher domain and transferring by an inverse order. As shown in

Method	Scratch (previous)	Transfer (new task)		Retest (previous)
	500 k	100 k	500 k	500 k
DrQ	**971**$_{\pm 27}$	283$_{\pm 121}$	332$_{\pm 96}$	124$_{\pm 22}$
Resnet+SAC	382$_{\pm 299}$	298$_{\pm 17}$	300$_{\pm 29}$	382$_{\pm 299}$
CtrlFormer	918$_{\pm 33}$	**299**$_{\pm 38}$	**547**$_{\pm 56}$	**889**$_{\pm 34}$

(a) Transfer from Reacher(`easy`) to Finger(`turn-easy`)

Method	Scratch (previous)	Transfer (new task)		Retest (previous)
	500 k	100 k	500 k	500 k
DrQ	**448**$_{\pm 65}$	203$_{\pm 87}$	693$_{\pm 282}$	184$_{\pm 57}$
Resnet+SAC	300$_{\pm 29}$	322$_{\pm 285}$	382$_{\pm 299}$	300$_{\pm 29}$
CtrlFormer	424$_{\pm 35}$	**416**$_{\pm 117}$	**770**$_{\pm 71}$	**409**$_{\pm 31}$

(b) Transfer from Finger(`turn-easy`) to Reacher(`easy`)

Table 3. **Transferring the state representation among tasks under cross-domain.** The first column is the performance of learning the previous task from scratch, the second column is the performance of learning the new task with the state representation transferring from the previous task, and the third column is the retest performance of the previous task using the latest encoder after learning the new task.

Table 3, the CtrlFormer has the best transferability and does not show a significant decrease after fine-turning on the new task. In contrast, the performance of DrQ decreases significantly after learning a new task and has worse performance damage than in the same domain because of a large gap across different domains.

Sequential transferring among more tasks. In order to test whether the improvement by state representation transfer is more significant with the increase of the number of tasks learned, we test the performance of CtrlFormer in four tasks from easy to difficult sequentially and further compare the transfer leaning via CtrlFormer to the method that learns the four tasks simultaneously. As shown in Table 4, the method of sequentially learning four tasks via CtrlFormer has the best sample efficiency. Furthermore, we can find that transferring state representation from more tasks performs better than transferring from only one task. CtrlFormer can surpass the performance of learning from scratch at 500k while only using 100k samples under this setting.

Method	Task 0 $\rightarrow$	Task 1 $\rightarrow$	Task 2 $\rightarrow$	Task 3
Scratch (100k)	967$_{\pm 27}$	869$_{\pm 61}$	759$_{\pm 48}$	0
Train together (100k)	433$_{\pm 23}$	143$_{\pm 34}$	310$_{\pm 41}$	0
CtrlFormer (100k)	**967**$_{\pm 27}$	**981**$_{\pm 29}$	**988**$_{\pm 36}$	**853**$_{\pm 69}$
Scratch (500k)	995$_{\pm 18}$	949$_{\pm 44}$	846$_{\pm 25}$	671$_{\pm 81}$
Train together (500k)	947$_{\pm 32}$	942$_{\pm 53}$	632$_{\pm 44}$	40$_{\pm 15}$
CtrlFormer (500k)	**995**$_{\pm 18}$	**1000**$_{\pm 0}$	**992**$_{\pm 26}$	**878**$_{\pm 64}$

Table 4. **Performance comparison with a series tasks.** Tasks 0-3 are `balance`, `balance_sparse`, `swingup` and `swingup_sparse`, from Cartpole domain.

As shown in Figure 5, we also develop an experiment on the DeepMind manipulation benchmark (Tunyasuvunakool et al., 2020) to further show the advantage of CtrlFormer, which provides a Kinova robotic arm and a list of objects for building manipulation tasks. We test the performance of CtrlFormer in 2 tasks: (1) Reach the ball: push the small red object near the white ball by the robot arm; (2) Reach the chess piece: push the small red object near the white chess piece by the robot arm. As shown in Table 2, CtrlFormer surpasses DrQ in terms of both transferability and sample efficiency when learning from scratch.

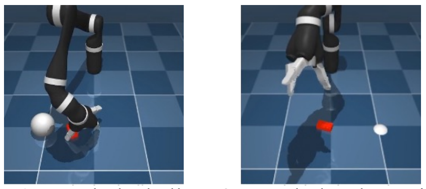

T1: Reach the ball T2: Reach the chess piece

Figure 5. Tasks in DeepMind manipulation

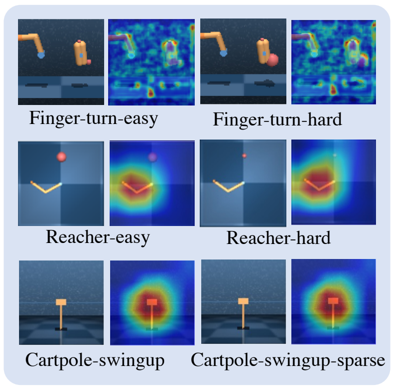
	DrQ	CtrlFormer
Scratch Task1(500k)	154$_{\pm 41}$	**175**$_{\pm 63}$
Retest Task1	87$_{\pm 33}$	**162**$_{\pm 75}$
Scratch Task2(500k)	141$_{\pm 47}$	**164**$_{\pm 33}$
Transfer T1 to T2 (100k)	73$_{\pm 48}$	**116**$_{\pm 34}$

Table 5. Performance comparison on DeepMind manipulation.

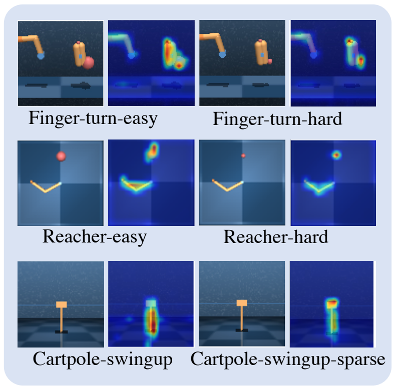
5.4. Visualization

Visualization of the similarity of different tasks. We visualize the cosine similarity of policy tokens between tasks in Figure 6c, which reflects the similarity of the visual representation between different tasks. The width and color of the bands in Figure 6c represents the similarity of the representations between two tasks. The thicker the line and the darker the color, the higher the similarity between the two tasks. As shown in Figure 6c, the similarity between Walker (`stand`) and Walker (`walk`) and between Cartpole (`swingup`) and Cartpole (`swingupsparse`) in the same domain is the top two strongest, indicating that it has better feature transfer potential. This is consistent with our test results, CtrlFormer significantly improves the sample efficiency after transferring the state representation. For the cross-domain, the similarity between different tasks is quite different, for example, the representation similarity between finger (`turn-easy`) and reacher (`easy`) is significantly higher than the similarity between finger (turn-easy) and reacher (`hard`). This is because, in the reacher (`hard`) task, the size of the target point is quite small, and the model pays too much attention to the target point and relatively less attention to the rod, while the finger task focuses more on the rod control.

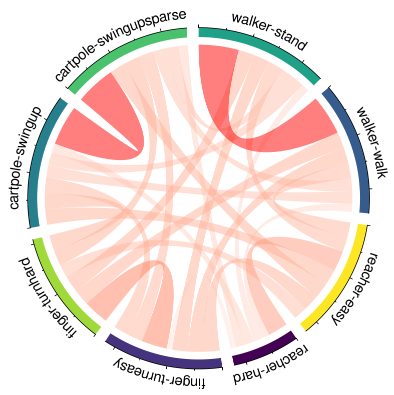
Visualization of the attention on input image: we visualize the attention of CtrlFormer and pre-trained ResNet-50 on the input image by Grad-CAM (Selvaraju et al., 2016),



(a) Visualization of Pre-trained ResNet

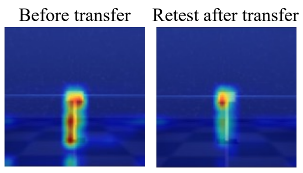


(b) Visualization of CtrlFormer

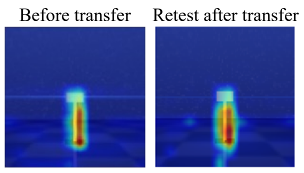


(c) Visualized similarity between tasks

Figure 6. Visualization of the attention on the input image and the similarity of different tasks in DMControl benchmark.



(a) DrQ



(b) CtrlFormer

Figure 7. **Comparison of the attention map change before and after the transferring.** Figure 7a shows that DrQ fails to pay attention to some key areas after transferring. In contrast, our CtrlFormer (Figure 7b) has consistent attention areas before and after transferring, demonstrating its superior property of transferring without catastrophic forgetting.

CtrlFormer	w/o contrastive		w/ contrastive	
	$100k$	$500k$	$100k$	$500k$
Cartpole(swingup)	$391_{\pm 42}$	$835_{\pm 29}$	$\mathbf{759}_{\pm 48}$	$\mathbf{846}_{\pm 25}$
Cartpole(swingup-sp)	0	$137_{\pm 54}$	0	$\mathbf{671}_{\pm 81}$
Reacher(easy)	$622_{\pm 38}$	$917_{\pm 64}$	$\mathbf{642}_{\pm 42}$	$\mathbf{973}_{\pm 53}$
Reacher(hard)	$49_{\pm 15}$	$234_{\pm 67}$	$\mathbf{104}_{\pm 48}$	$\mathbf{548}_{\pm 131}$
Walker(stand)	$865_{\pm 33}$	$930_{\pm 44}$	$\mathbf{877}_{\pm 42}$	$\mathbf{954}_{\pm 38}$
Walker(walk)	$406_{\pm 73}$	$708_{\pm 45}$	$\mathbf{593}_{\pm 52}$	$\mathbf{903}_{\pm 43}$
Finger(turn-easy)	$15_{\pm 7}$	$344_{\pm 38}$	$\mathbf{281}_{\pm 67}$	$\mathbf{493}_{\pm 35}$
Finger(turn-hard)	$99_{\pm 44}$	$136_{\pm 38}$	$\mathbf{197}_{\pm 78}$	$\mathbf{344}_{\pm 47}$

Table 6. Ablation study on the effectiveness of co-training with contrastive learning.

which is a typical visualization method, more details are introduced in Appendix C. As shown in Figure 6a, the attention of ResNet-50 is disturbed by a lot of things unrelated to the task, which is the key reason why it has low sample efficiency in both Table 2 and Table 3. The attention of CtrlFormer is highly correlated with the task, and different policy tokens learn different attention to the input image. The attention learned from similar tasks has similarities, but each has its own emphasis. Moreover, we also compare the attention map change on the old task before and after transferring in Figure 7, the attention map of CtrlFormer is not obviously changed, while the attention map of CNN-based model used in DrQ changed obviously, which provides its poor retesting performance with a reasonable explanation.

5.5. Ablation Study

To illustrate the effect of the co-training method with contrastive learning, we compare the sample efficiency of CtrlFormer and CtrlFormer without co-training, as shown in Table 6, CtrlFormer shows higher sample efficiency, proving the effectiveness of the proposed co-training for improving the sample efficiency of the transformer-based model.

In conclusion, CtrlFormer surpasses the baselines and shows great transferability for visual control tasks. The experiments in DMControl benchmark illustrated that CtrlFormer has great potential to model the correlation and irrelevance between different tasks, which improves the sample efficiency significantly and avoids catastrophic forgetting.

6. Conclusion

In this paper, we propose a novel representation learning framework CtrlFormer that learns a transferable state representation for visual control tasks via a sample-efficient vision transformer. CtrlFormer explicitly learns the attention among the current task, the tasks it learned before, and the observations. Furthermore, each task is co-trained with contrastive learning as an auxiliary task to improve the sample efficiency when learning from scratch. Various experiments show that CtrlFormer outperforms previous work in terms of transferability and the great potential to be extended in multiple sequential tasks. We hope our work could inspire rethinking the transferability of state representation learning for visual control and exploring the next generation of visual RL framework.

Acknowledgement. Ping Luo is supported by the General Research Fund of HK No.27208720 and 17212120.

References

Agarap, A. F. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bahng, H., Jahanian, A., Sankaranarayanan, S., and Isola, P. Visual prompting: Modifying pixel space to adapt pre-trained models. *arXiv preprint arXiv:2203.17274*, 2022.

Bellman, R. A markovian decision process. *Indiana Univ. Math. J.*, 1957.

Buesing, L., Weber, T., Racaniere, S., Eslami, S., Rezende, D., Reichert, D. P., Viola, F., Besse, F., Gregor, K., Hassabis, D., et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

Chen, S., Ge, C., Tong, Z., Wang, J., Song, Y., Wang, J., and Luo, P. Adaptformer: Adapting vision transformers for scalable visual recognition. *arXiv preprint arXiv:2205.13535*, 2022a.

Chen, X., Mu, Y., Luo, P., Li, S., and Chen, J. Flow-based recurrent belief state learning for pomdps. *arXiv preprint arXiv:2205.11051*, 2022b.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Doerr, A., Daniel, C., Schiegg, M., Duy, N.-T., Schaal, S., Toussaint, M., and Sebastian, T. Probabilistic recurrent state-space models. In *International Conference on Machine Learning*, pp. 1280–1289. PMLR, 2018.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmassan, Stockholm, Sweden, July 10-15, 2018*, 2018.

Gideon, J., Khorram, S., Aldeneh, Z., Dimitriadis, D., and Provost, E. M. Progressive neural networks for transfer learning in emotion recognition. *arXiv preprint arXiv:1706.03256*, 2017.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. 2018a.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018b.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870. PMLR, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels, 2019b.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Holt, C. C. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Jia, M., Tang, L., Chen, B.-C., Cardie, C., Belongie, S., Hariharan, B., and Lim, S.-N. Visual prompt tuning. *arXiv preprint arXiv:2203.12119*, 2022.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 1998.

Karl, M., Soelch, M., Bayer, J., and Van der Smagt, P. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Krishnan, R. G., Shalit, U., and Sontag, D. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv e-prints*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Mu, Y., Zhuang, Y., Wang, B., Zhu, G., Liu, W., Chen, J., Luo, P., Li, S. E., Zhang, C., and HAO, J. Model-based reinforcement learning via imagination with derived memory. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=jeATherHHGj.

Oby, E. R., Golub, M. D., Hennig, J. A., Degenhart, A. D., Tyler-Kabara, E. C., Byron, M. Y., Chase, S. M., and Batista, A. P. New neural activity patterns emerge with long-term learning. *Proceedings of the National Academy of Sciences*, 116(30):15210–15215, 2019.

Popescu, M.-C., Balas, V. E., Perescu-Popescu, L., and Mastorakis, N. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A. S., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *arXiv preprint arXiv:2106.05974*, 2021.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Rusu, A. A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, pp. 262–270. PMLR, 2017.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv e-prints*, 2013.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*, 2016.

Shah, R. and Kumar, V. Rrl: Resnet as representation for reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*, 2021.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tirinzoni, A., Poiani, R., and Restelli, M. Sequential transfer in reinforcement learning with a generative model. In *International Conference on Machine Learning*, pp. 9481–9492. PMLR, 2020.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *arXiv e-prints*, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Wang, S., Jia, D., and Weng, X. Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*, 2018.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

Yen-Chen, L., Zeng, A., Song, S., Isola, P., and Lin, T.-Y. Learning to see before learning to act: Visual pre-training for manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7286–7293. IEEE, 2020.

Yuan, Z., Ma, G., Mu, Y., Xia, B., Yuan, B., Wang, X., Luo, P., and Xu, H. Don't touch what matters: Task-aware lipschitz data augmentationfor visual reinforcement learning. *arXiv preprint arXiv:2202.09982*, 2022.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, 2008.

A. Extended Background

Soft Actor-Critic The Soft Actor-Critic (SAC) (Haarnoja et al., 2018b) learns a state-action value function Q_θ, a stochastic policy π_θ and a temperature α to find an optimal policy for an MDP (S, A, p, r, γ) by optimizing a γ-discounted maximum-entropy objective (Ziebart et al., 2008). θ is used generically to denote the parameters updated through training in each part of the model. The actor policy $\pi_\theta(a_t|s_t)$ is a parametric tanh-Gaussian that given s_t samples $a_t = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t)\epsilon)$, where $\epsilon \sim N(0, 1)$ and μ_θ and σ_θ are parametric mean and standard deviation.

The policy evaluation step learns the critic $Q_\theta(s_t, a_t)$ network by optimizing a single step of the soft Bellman residual

$$\begin{aligned} J_Q(\mathcal{D}) &= E_{\substack{(s_t,a_t,s'_t)\sim\mathcal{D} \\ a'_t\sim\pi(\cdot|s'_t)}}[(Q_\theta(s_t, a_t) - y_t)^2] \\ y_t &= r(s_t, a_t) + \gamma[Q_{\theta'}(s'_t, a'_t) - \alpha \log \pi_\theta(a'_t|s'_t)], \end{aligned} \tag{12}$$

where $\mathcal{D}$ is a replay buffer of transitions, θ' is an exponential moving average of the weights. SAC uses clipped double-Q learning (van Hasselt et al., 2015; Fujimoto et al., 2018), which we omit from our notation for simplicity but employ in practice.

The policy improvement step then fits the actor policy $\pi_\theta(a_t|s_t)$ network by optimizing the objective

$$J_\pi(\mathcal{D}) = E_{s_t\sim\mathcal{D}}[D_{KL}(\pi_\theta(\cdot|s_t)|| \exp\{\frac{1}{\alpha}Q_\theta(s_t, \cdot)\})].$$

Finally, the temperature α is learned with the loss

$$J_\alpha(\mathcal{D}) = E_{\substack{s_t\sim\mathcal{D} \\ a_t\sim\pi_\theta(\cdot|s_t)}}[-\alpha \log \pi_\theta(a_t|s_t) - \alpha\mathcal{H}],$$

where $\mathcal{H}$ is the target entropy hyper-parameter that the policy tries to match.

Deep Q-learning DQN (Mnih et al., 2013) also learns a convolutional neural net to approximate Q-function over states and actions. The main difference is that DQN operates on discrete actions spaces, thus the policy can be directly inferred from Q-values. The parameters of DQN are updated by optimizing the squared residual error

$$\begin{aligned} J_Q(\mathcal{D}) &= E_{(s_t,a_t,s'_t)\sim\mathcal{D}}[(Q_\theta(s_t, a_t) - y_t)^2] \\ y_t &= r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s'_t, a'). \end{aligned}$$

In practice, the standard version of DQN is frequently combined with a set of tricks that improve performance and training stability, wildly known as Rainbow (van Hasselt et al., 2015).

B. Supplumentary Materials of Experiments

Our PyTorch code is implanted based on the Timm (Wightman, 2019), Pytorch version SAC (Haarnoja et al., 2018b), and the official code of DrQ-v1(Kostrikov et al., 2020). All the experiments are run on the GeForce RTX 3090 with 5 seeds.

B.1. Detailed Structure of Vision Transformer in CtrlFormer

We utilize a simple pyramidal vision transformer as the encoder of CtrlFormer, which has 3 stages and each stage contains 3 blocks. With the 192 dimension output of the vision transformer, we add a fully-connected layer to map the feature dimension to 50, and apply tanh nonlinearity to the 50 dimensional output as the final output of the encoder.

The detailed structure of CtrlFormer is shown as Figure 8. The hyper-parameters are listed in Table 7. The implantation is based on Timm (Wightman, 2019), which is a collection of SOTA computer vision models with the ability to reproduce ImageNet training results. We train the transformer model using Adamw (Kingma & Ba, 2014) as optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, a batch size of 512 and apply a high weight decay of 0.1.

B.2. Actor and Critic Networks

The structure of actor and critic networks are the same in CtrlFormer and DrQ(CNN+multiple heads). Clipped double Q-learning (van Hasselt et al., 2015; Fujimoto et al., 2018) is used for the critic, where each Q-function is parametrized as a 3-layer MLP with ReLU activations after each layer except for the last. The actor is also a 3-layer MLP with ReLUs that outputs mean and covariance for the diagonal Gaussian that represents the policy. The hidden dimension is set to 1024 for both the critic and actor.

Parameters	Value
Image size	84×84
Patch size	8
Num patches	196
Input channels	9
Embedding dim	192
Depth	9
Num stage	3
Num blocks per stage	3
num heads	3

Table 7. Hyper-parameter of CtrlFormer

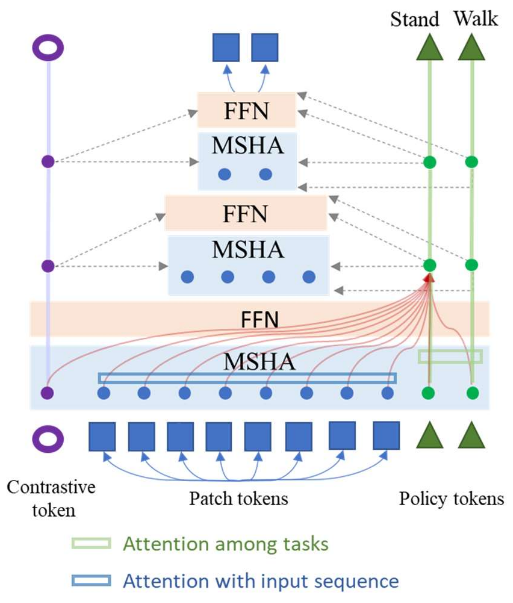


Figure 8. The detailed structure of CtrlFormer

B.3. CNN-based Encoder Network used in DrQ

The CNN-based model (Kostrikov et al., 2020) employs an encoder consists of four convolutional layers with 3×3 kernels and 32 channels, which is same to DrQ (Kostrikov et al., 2020). The ReLU activation is applied after each convolutional layer. We use stride to 1 everywhere, except for the first convolutional layer, which has stride 2. The output of the convolutional network is fed into a single fully-connected layer normalized by LayerNorm (Ba et al., 2016). Finally, we apply tanh non-linearity to the 50 dimensional output of the fully-connected layer. We initialize the weight matrix of fully-connected and convolutional layers with the orthogonal initialization (Saxe et al., 2013).

The actor and critic networks share the weights of the convolutional layers of the encoder. Furthermore, only the critic optimizer is allowed to update these weights (e.g. the gradients from the actor is stopped before they propagate to the shared CNN layers).

B.4. Detailed Training and Evaluation Setup for Visual Control Tasks

The agent first collects 1000 seed observations using a random policy. The further training observations are collected by sampling actions from the current policy. The agent performs one training update every time when receiving a new observation. The action repeat parameters are used as same as the DrQ (Kostrikov et al., 2020), and is listed in Table 8, and the number of training observations is only a fraction of the environment steps (e.g. a 1000 steps episode at action repeat 8 will only result in 125 training observations). In order to avoid damaging the CtrlFormer due to the low-quality policy gradient caused by the inaccurate policy network and Q-networks at the beginning of the behavior learning in the new task, the learning rate of CtrlFormer is set α_l times lower than policy network and Q-networks. In this way, the agent is guided to quickly learn the policy of the current task based on the previously learned representation and fine-tune the model according to the relationship between the current task, the previous tasks, and the input. The training curves using different α_ℓ are shown in the Figure 9, showing that α_ℓ in the range of 0.04-0.08 is appropriate. We evaluate our agent every 10000 environment step by computing the average episode return over 10 evaluation episodes as same as DrQ (Kostrikov et al., 2020). During the evaluation, we take the mean of the policy output action instead of stochastic sampling. In Table 9 we provide a comprehensive overview of all the other hyper-parameters.

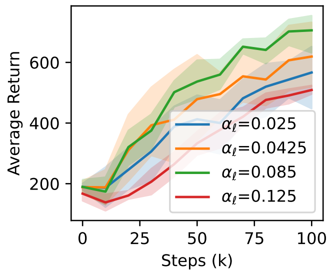


Figure 9. Training curves with different α_ℓ

Task name	Action repeat
Cartpole Swingup	8
Cartpole Swingup sparse	8
Cartpole balance	8
Cartpole balance sparse	8
Reacher Easy	4
Reacher Hard	4
Finger Turn easy	2
Finger Turn Hard	2
Walker Walk	2
Walker stand	2

Table 8. The action repeat hyper-parameter used for each task.

B.5. Image Preprocessing and Augmentation

We construct an observational input as a 3-stack of consecutive frames, which is the same as DQN (Mnih et al., 2013) and DrQ (Kostrikov et al., 2020), where each frame is an RGB rendering of size 84×84 from the camera. We then divide each pixel by 255 to scale it down to $[0, 1]$ range as the input of the encoder.

The images from the DeepMind control suite are 84×84.

The image augmentation used in 4.3 and 4.4 is implanted by random crop which is also used in DrQ (Kostrikov et al., 2020). We pad each side by 4 repeating boundary pixels and then select a random 84×84 crop, yielding the original image shifted by ± 4 pixels. This procedure is repeated every time an image is sampled from the replay buffer.

Parameter	Setting
Replay buffer capacity	100000
Seed steps	1000
Batch size	512
Discount γ	0.99
Optimizer	Adamw
Learning rate	10^{-4}
Critic target update frequency	2
Critic Q-function soft-update rate τ	0.01
Actor update frequency	2
Actor log stddev bounds	$[-10, 2]$
Init temperature	0.1

Table 9. Hyper-parameters of downstream reinforcement learning.

B.6. Implementation Details of Contrastive Learning

We implanted the contrastive optimization by BYOL (Grill et al., 2020), which is a typical approach to self-supervised image representation learning.

BYOL relies on two neural networks, referred to as *online* and *target* networks, that interact and learn from each other and do not rely on negative pairs. From an augmented view of an image, we train the online network f_θ to predict the target network f_ξ representation of the same image under a different augmented view. At the same time, we update the target network with a slow-moving average of the online network.

The output of the contrastive token in CtrlFormer is a 192 dimension vector. We project it to a 96 dimension vector by a multi-layer perceptron (MLP) and similarly for the target projection. This MLP consists of a linear layer with output size 384 followed by batch normalization (Ioffe & Szegedy, 2015), rectified linear units (ReLU) (Agarap, 2018), and a final linear layer with output dimension 96. The predictor q_θ uses the same architecture as the projector. For the target network, the exponential moving average parameter τ starts from $\tau_{\text{base}} = 0.996$ and is increased to one during training.

B.7. Additional Results

Ablation on Q-regularization. To illustrate the effect of the Q-regularization technique, we compare the transferability of CtrlFormer, CtrlFormer without Q-regularization, and DrQ without Q-regularization. Table 10 shows that the Q-regularization technique helps to improve the performance of CtrlFormer, and CtrlFormer still outperforms DrQ on transferability without Q-regularization.

	Scratch Task1	Retest	Scratch Task2	Transfer	Benifit
Our w/ rQ	$973_{\pm 53}$	$906_{\pm 31}$	$548_{\pm 124}$	$657_{\pm 68}$	$+16.5\%$
Our w/o rQ	$774_{\pm 32}$	$738_{\pm 54}$	$474_{\pm 56}$	$551_{\pm 57}$	$+13.8\%$
DRQ w/o rQ	$756_{\pm 47}$	$329_{\pm 58}$	$481_{\pm 198}$	$410_{\pm 47}$	-14.76%

Table 10. Ablation on Q-regularization by transferring from Reacher(easy) to Reacher(hard).

Contrastive co-training on other baselines. We provide an additional baseline that applies contrastive co-training on the CNN-based model and compare it with CtrlFormer. The results show DrQ with contrastive co-training (DrQ-C) achieves better transferring results from Walker-Stand (T1) to Walker-Walk (T2) than the original DrQ algorithm. But it still suffers from catastrophic forgetting.

	Retest(T1 500k)	Transfer T2 (100k)
DrQ	$698_{\pm 57}$	$321_{\pm 54}$
DrQ-C	$707_{\pm 68}$	$472_{\pm 68}$
Our	**950**$_{\pm 42}$	**857**$_{\pm 47}$

Table 11. Ablation on Contrastive co-training

B.8. PyTorch-style Pseudo-code

We provide detailed PyTorch-style pseudo-codes of the method we visualize the policy attention shown in Figure 6, and the method we update the encoder, the actor and the critic. The pseudo-code of Grad-CAM is shown in Listing 1. The pseudo-code of the actor and the entropy temperature is shown in Listing 2. The pseudo-code of the encoder and the critic is shown in Listing3.

B.9. DMControl Benchmark

The DeepMind Control Suite (DMControl) (Tassa et al., 2018) is a set of stable, well-tested continuous control tasks that are easy to use and modify. DMControl contains many well designed tasks, which are written in Python and physical models are defined using MJCF. It is currently one of the most recognized standard test environments for visual control. The domain in DMControl refers to a physical model, while a task refers to an instance of that model with a particular MDP structure. For example, the difference between the `swingup` and `balance` tasks of the `cartpole` domain is whether the pole is initialized pointing downwards or upwards, respectively. We list the detailed descriptions of the domains used in this paper below, names are followed by three integers specifying the dimensions of the state, control and observation spaces i.e. $\Big(\dim(\mathcal{S}), \dim(\mathcal{A}), \dim(\mathcal{O})\Big)$.

Figure 10. **Walker (18, 6, 24):** The **Walker** domain contains a series of control tasks for two-legged robots. In the `stand` task, the reward is a combination of terms encouraging an upright torso and some minimal torso height. The `walk` and `run` tasks include a component encouraging forward velocity.

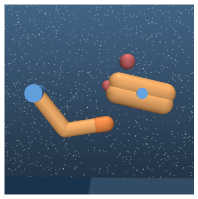


Figure 11. **Finger(6, 2, 12):** **Finger** domain aims to rotate a body on an unactuated hinge. In the `turn_easy` and `turn_hard` tasks, the tip of the free body must overlap with a target (the target is smaller for the `turn_hard` task). In the `spin` task, the body must be continually rotated.

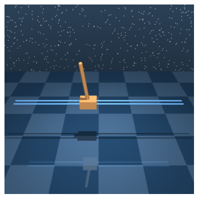


Figure 12. **Cartpole(4, 1, 5):** **Cartpole** domain aims to control the pole attached by an un-actuated joint to a cart. In both the `swingup` task and the `swingup-sparse` task, the pole starts pointing down and aim to make the unactuated pole keeping upright upward by applying forces to the cart, while in `balance` and `balance_sparse` the pole starts near the upright.

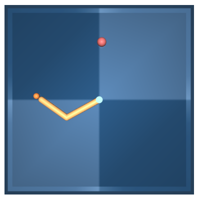


Figure 13. **Reacher (4, 2, 7):** **Reacher** domain aims to control the two-link planar reach a randomised target location. The reward is one when the end effector penetrates the target sphere. In the `easy` task the target sphere is bigger than on the `hard` task (shown on the left).

C. Visualization

We use the Grad-CAM (Selvaraju et al., 2016) method to visualize the encoder's attention on the input image, which is a typical technique for visualizing the regions of input that are "important". Grad-CAM uses the gradient information flowing to produce a coarse localization map of the important regions in the image. From Figure 14 and Figure 15 we can observe that our attention map is more focused on objects and task-relevant body parts, while the attention of the pre-trained ResNet-50 (same as the network used in the experiments) is disturbed by irrelevant information and not focused. In this way, our CtrlFormer learns better policies by the representation highly relevant to the task. The attention learned from similar tasks has similarities, but each has its own emphasis.

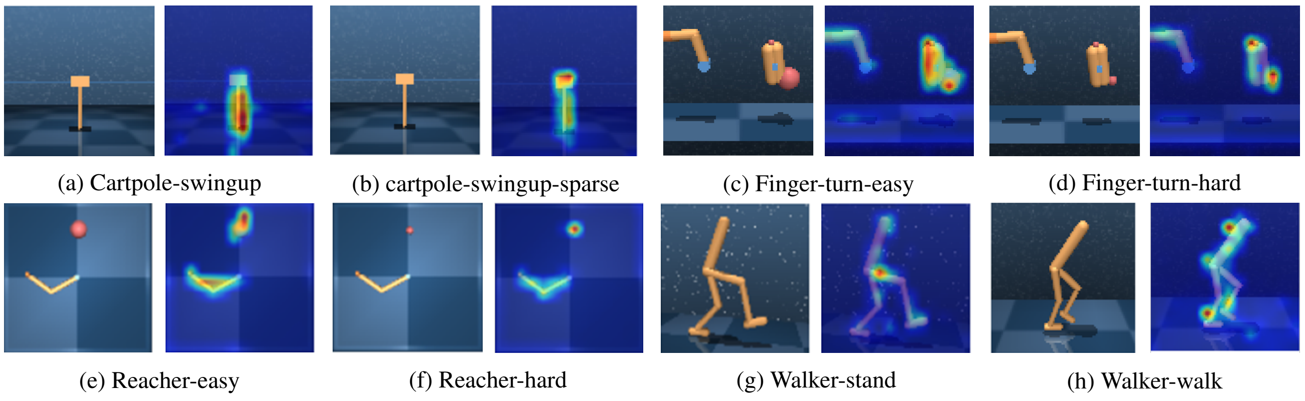


Figure 14. Visualization of the CtrlFormer's attention on the input image

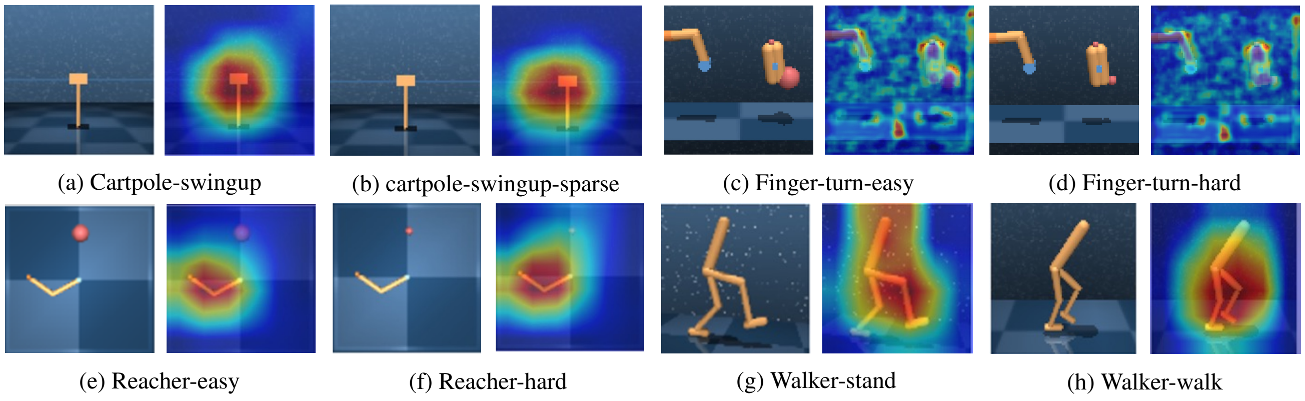


Figure 15. Visualization of the Resnet's attention on the input image

```python
class ViTAttentionGradRollout:
    def __init__(self, model, attention_layer_name='attn_drop',
        discard_ratio=0.9):
        self.model = model
        self.discard_ratio = discard_ratio
        for name, module in self.model.named_modules():
            if attention_layer_name in name:
                module.register_forward_hook(self.get_attention)
                module.register_backward_hook(self.get_attention_gradient)

        self.attentions = []
        self.attention_gradients = []

    def get_attention(self, module, input, output):
        self.attentions.append(output.cpu())

    def get_attention_gradient(self, module, grad_input, grad_output):
        self.attention_gradients.append(grad_input[0].cpu())

    def __call__(self, input_tensor):
        self.model.zero_grad()
        output = self.model(input_tensor,detach=False)
        loss = (output).sum()
        loss.backward()
        return grad_rollout(self.attentions, self.attention_gradients,
            self.discard_ratio)

```

Listing 1. PyTorch-style pseudo-code for attention visualization use in this paper.

```python
def update_actor_and_alpha(obs):
        # detach encoder layers
        dist = actor(obs, detach_encoder=True)
        action = dist.rsample()
        log_prob = dist.log_prob(action).sum(-1, keepdim=True)
        # detach encoder layes
        actor_Q1, actor_Q2 = critic(obs, action, detach_encoder=True)
        actor_Q = torch.min(actor_Q1, actor_Q2)
        actor_loss = (alpha.detach() * log_prob - actor_Q).mean()
        # optimize the actor
        actor_optimizer.zero_grad()
        actor_loss.backward()
        actor_optimizer.step()
        actor.log(logger, step)
        log_alpha_optimizer.zero_grad()
        alpha_loss = (alpha *
                      (-log_prob - target_entropy).detach()).mean()
        alpha_loss.backward()
        log_alpha_optimizer.step()
```

Listing 2. PyTorch-style pseudo-code for the actor updating in down stream reinforcement learning.

```
def update_critic_and_encoder(self, obs, obs_aug, action, reward, next_obs, next_obs_aug, not_done, logger, step
     ):
        cons  = encoder.forward_rec(imgs)
        cons = encoder.byol_project(cons).detach()
        aug_cons = encoder.forward_cons(aug_imgs)
        project_cons = encoder.project(aug_cons)
        predict_cons = encoder.predict(project_cons)
        cons_loss = F.mse_loss(predict_cons,cons)
        with torch.no_grad():
            dist = actor(next_obs)
            next_action = dist.rsample()
            log_prob = dist.log_prob(next_action).sum(-1)
            target_Q1, target_Q2 = critic_target(next_obs,next_action)
            target_V = torch.min(target_Q1,
                                 target_Q2) - alpha.detach() * log_prob
            target_Q = reward + (not_done * discount * target_V)

            dist_aug = actor(next_obs_aug)
            next_action_aug = dist_aug.rsample()
            log_prob_aug = dist_aug.log_prob(next_action_aug).sum(-1)
            target_Q1, target_Q2 = critic_target(next_obs_aug, next_action_aug)
            target_V = torch.min(
                target_Q1, target_Q2) - alpha.detach() * log_prob_aug
            target_Q_aug = reward + (not_done * discount * target_V)
            target_Q = (target_Q + target_Q_aug) / 2
        # get current Q estimates
        current_Q1, current_Q2 = critic(obs, action)
        critic_loss = F.mse_loss(current_Q1, target_Q) + F.mse_loss(
            current_Q2, target_Q)
        Q1_aug, Q2_aug = critic(obs_aug, action)
        critic_loss += F.mse_loss(Q1_aug, target_Q) + F.mse_loss(
            Q2_aug, target_Q) + rec_loss
        logger.log('train_critic/loss', critic_loss, step)
        # Optimize the critic
        critic_optimizer.zero_grad()
        critic_loss.backward()
        critic_optimizer.step()
        critic.log(logger, step)

```

Listing 3. PyTorch-style pseudo-code for the updating of encoder and critic