Latent Communication for Zero-shot Stitching in Reinforcement Learning

Antonio Pio Ricciardi
ricciardi@di.uniroma1.it
Sapienza University of Rome, Italy

Valentino Maiorca
maiorca@di.uniroma1.it
Sapienza University of Rome, Italy

Luca Moschella
moschella@di.uniroma1.it
Sapienza University of Rome, Italy

Riccardo Marin
riccardo.marin@uni-tuebingen.de
University of Tübingen, Germany

Emanuele Rodolà
rodola@di.uniroma1.it
Sapienza University of Rome, Italy

Abstract

Visual Reinforcement Learning is a popular and powerful framework that takes full advantage of the Deep Learning breakthrough. It is known that variations in the input (e.g., different colors of the panorama due to the season of the year) or task (e.g., changing the target speed of a car) domains could disrupt agents performance, therefore requiring new training. Recent advancements in Latent Communication Theory, show that it is possible to combine components of different neural networks to create new models in a zero-shot fashion. In this paper, we leverage upon such advancements to show that components of agents trained on different visual and task variations can be combined by aligning the latent representations produced by their encoders, to obtain new agents that can act well in visual-task combinations never seen together during training. Our findings open to more efficient training processes, significantly reducing time and computational costs. We release the code at `https://github.com/antoniopioricciardi/rl_relrepr_gymnasium.git`

1 Introduction

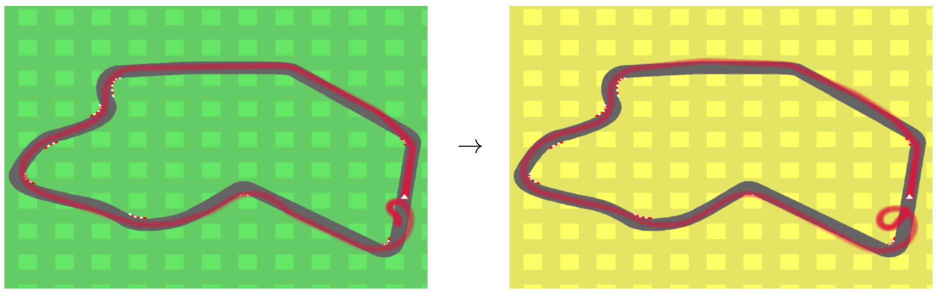


Figure 1: Using translation methods, a controller trained on an environment with a given visual variation *(left)* can be reused without any training or fine-tuning on a different environment (*right*) with comparable performance. In red we see the trajectory of a car driven by the same controller when connected to two different encoders, one for each visual variation.

Reinforcement Learning (RL) drives some of the most prominent achievements in modern artificial intelligence. Its combination with deep learning enables superhuman performances in complex and articulated tasks like strategic games [26, 27], showing micro and macro adaptability to settings with a wide variety of inputs [32].

However, Deep RL's outstanding performance is not accessible to everyone. Training agents without labeled data comes at the cost of time-consuming computations, generating millions of interactions and requiring weeks of training. Furthermore, these architectures face sensitive training procedures: for example, different random seeds may cause divergent training behaviors. Finally, agents specialize in exploiting the world they observed during training, and any shift in the visual, task, or domains usually requires training new agents from scratch.

Many works try to cover for possible domain shift *a priori*, by using domain randomization [31] or data augmentation techniques [9, 36, 35, 11], so that invariance can be produced by "memorizing" what is irrelevant to the task during training, but often requiring longer training and more complex architectures. Moreover, it might be necessary to know what types of perceptual shifts would occur during deployment. Latent alignment techniques [10, 37] aim to produce feature invariance at deployment time, assuming the task remains the same. This is obtained by collecting latent features during training as examples of what the agent knows about the task so that, when presented with a new visual domain, the new latents are trained to match the distribution of those that appeared during training, while other methods work to make neural components reusable by imposing training constraints [4, 14]. Recent advances in the representation learning field show that it is possible to reuse neural components from trained models in a zero-shot fashion, by either projecting the spaces of different models to a common one [23], or by mapping the produced latent space from a model to another [20].

In this work, we show how these methods can be applied in the setting of visual deep RL so that learned visual representations and skills can be reused by recombining neural network models in a *zero-shot* fashion. For example, imagine an agent trained to drive a car on a racing track exclusively during spring: to drive during summer, it would be necessary to retrain the agent to account for the visual variation given by the different grass colors. We empirically show that learned representations from different models can be glued in a *zero-shot* setting. For example, as illustrated in Figure 1, we can reuse and combine neural components between agents trained during spring and agents trained during summer, to create a new agent able to perform on visual and task variations never-seen together during training.

Contributions. We propose a method to combine encoders and controllers trained in different regimes with perceptual shifts or task objective variations. We assemble new agents capable of visual-task pairs *never seen nor collected for training*, without any retraining or fine-tuning. We base our investigation on recent advancements in representation learning [20, 23], and for the first time in RL, we show that it is possible to either: (i) recover a map to translate model representations between them; (ii) unifying their spaces and producing a universal policy. We test our method on various environments with visual, task, and seed variations. All the experiments demonstrate that under the instability of RL training lies a similar understanding of the world. Our opening to direct translation between these representations leads to exciting theoretical and applicative perspectives.

In summary, our work contributes to exploring emerging patterns in the representations learned in different environments and their variations. We propose to investigate end-to-end trained models as a union of two different components, and we use this ground to propose a novel procedure to achieve zero-shot stitching of modules learned from different trainings. This zero-shot stitching allows for policies components trained on a variation of a certain environment to be transferred into another one. Indeed, as shown in Section 4, the proposed methodology can generalize to variations in the observations domain (e.g., different background colors or camera perspective) and changes in the environment objective (e.g., different task or action space).

2 Related Work

Model Stitching The concept of "Model Stitching" has been explored to analyze the similarity of latent spaces across different models. For instance, stitching layers as discussed in [18, 1, 3] provide a metric for measuring similarity. Recently, model stitching has been extended to facilitate model reuse by integrating parts from multiple networks. Rather than designing compatible and reusable components, which can complicate the model architecture [8, 34], these recent approaches demonstrate the feasibility of zero-shot stitching between neural networks [23, 24, 20]. They assume the knowledge of a semantic correspondence between the training distributions, which can be leveraged to unify the representations into a universal space or to estimate a direct translation between them.

RL Modular Agents Transferring agent policies across visual or dynamic environment variations remains an open challenge [40, 22, 13]. The idea of reusing components like value functions [30, 19], policies [5, 4], parameters [16, 6], features [2] has been widely explored. On a similar line, [39] combines reward and latent space distance by minimizing bisimulation metrics to train encoders invariant to certain visual distractors or small differences in the input space. An alternative is to design agents as the composition of multiple modules. Modular RL approaches work in a "sense-plan-act" paradigm [15] to mitigate sample inefficiency by combining different modules, each able to solve different sub-problems [21], such that when combined they can solve more complex tasks [25, 28]. In numerous instances [33], existing prior knowledge and other sources of information are used. In this sense, a module can be considered a policy dedicated to a sub-task. Then, a neural network acts as a planner, selecting the correct policy at the right moment [29, 7]. However, module subdivisions also constitute a major drawback, since they require specific architectures and careful manual design.

Latent alignment for RL Another family of techniques in RL seeks for invariances in the latent space, relying on the assumption that for some input variations, "the task remains the same, and so the agent experiences internally should also remain the same" [37]. For example, PAD [10] trains an inverse dynamics model in a supervised manner, meanwhile ILA [37] uses finetuning to match prior distributions in visual variations without needing paired image data. Both works, however, exclusively address visual variations, without investigating task changes. Differently, [4] perform task-robot stitching in what they call interconnected modules. However, they need to restrict agents to a limited number of hidden units to keep things manageable, while combining trajectory optimization and supervised learning to train a global neural network policy, using regularization techniques to achieve task-invariance and avoid overfitting to the robot-task combinations seen during training. Finally, [14] shows promising results in policy stitching for a robot arm, relying on the relative representation framework Moschella et al. [23], with the caveat of working with low-dimensional signals as input and necessitating the training of new models to project latent spaces to a common one using relative representations.

Our Positioning Following the recent methodologies proposed by [23] and [20], we frame our work in the context of latent communication for reinforcement learning agents, focusing on performing zero-shot stitching between different neural models trained under different conditions. In particular, we enable communication between policy encoders by projecting their latent spaces to a common space or mapping from one space to another. By employing **relative representations**, we can create a shared latent space that facilitates compositionality between encoders and controllers, as controllers can interpret the latent spaces of other encoders if they share the same space. Unlike Jian et al. [14], we work with high-dimensional features (images) rather than low-dimensional states and avoid the need for fine-tuning during stitching. Furthermore, we explore **latent alignment** techniques for reinforcement learning, which directly map between different latent spaces. These techniques allow stitching arbitrary models trained on various visual and task variations of a given environment and create new policies for unseen visual-task combinations in a zero-shot manner.

3 Method

Context We formally model an RL problem as a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, R, P, \gamma)$, where $\mathcal{S}$ defines the set of states, $\mathcal{A}$ the set of actions, $o \in \mathcal{O}$ the observation produced as input for the agents, $P : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ the probability distribution $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ of transitioning to state $\mathbf{s}'$ upon executing action $\mathbf{a}$ in state $\mathbf{s}$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R}$ the reward function, and γ the discount factor that reduces the importance of rewards obtained far in the future. The agent's behavior is dictated by a policy $\pi : \mathcal{O} \to \mathcal{A}$ that receives an observation and selects an action at each state, and is trained to maximize the discounted long-term returns $\mathbb{E}[\sum_{i=0}^{\infty} \gamma^i \mathcal{R}(\mathbf{s}_i, \mathbf{a}_i)]$.

Environments Variations We are interested in studying the effects of variations between different training and agents. To better frame the concept of "variation", we redefine the environments as follows. First, we will refer to $\mathcal{M}_u^i = (\mathcal{O}_u, T_i)$ as the environment that produces *observations* $o_u \in \mathcal{O}_u$ and has a *task* $T_i : \mathcal{S}_i \times \mathcal{A}_i \times \mathcal{R}_i \times P_i \mapsto \mathcal{R}_i$ to solve. We will say that a distribution of observations $\mathcal{O}_u$ is different from another when there is a major variation between them (e.g., background color, camera perspective). At the same time, task variations can impact the agent behavior, and can depend on transition dynamics, internal states, action spaces, and reward functions. Information about the task T_i is not explicitly given to the agent, and it must be inferred using the guidance of the reward signal provided while acting in the environment. During training, we train agents on different combinations of visual-task variations, separately for each combination. When testing, we also use environments with visual-task variations never seen during training, i.e., testing on $\mathcal{M}_u^j$ means that no agent has been trained end-to-end to solve the task T_j for the observations O_u.

Modular Agents The standard practice to obtain a policy π_u^i is to end-to-end train a neural network on environment $\mathcal{M}_u^i$. However, we argue that this network can be seen as a composition of two functions: an encoder $\phi_u^i : \mathcal{O}_u^i \mapsto \mathcal{X}_u^i$ trained on an environment with an observation space $\mathcal{O}_u$ to produce a latent representation $\mathbf{x}_u^i$, and a controller $\psi_u^i : \mathcal{X}_u^i \mapsto \mathcal{A}_i$ trained to act on task T_i given the latent representation coming from the encoder. Given an observation o_u, π_u^i can be defined by composition:

$$\pi_u^i(o_u) = \psi_u^i[\phi_u^i(o_u)] = \psi_u^i(\mathbf{x}_u^i). \tag{1}$$

Latent representation Consider the existence of a second environment $\mathcal{M}_v^j = (\mathcal{O}_v, \mathcal{T}_j)$, where the observations $\mathcal{O}_v$ differ from the ones from $\mathcal{O}_u$ just for a visual variation (e.g., different color for the grass in CarRacing), and a trained policy π_v^j. Given two corresponding observations $o_u \in O_u$ and $o_v \in O_v$, the latent representations produced by the respective encoders are different:

$$\phi_u^i(o_u) \neq \phi_v^j(o_v) \quad \text{and therefore} \quad \mathbf{x}_u^i \neq \mathbf{x}_v^j. \tag{2}$$

In the following, we describe how to leverage the emerging similarities in these learned latent representations to unify them, enabling zero-shot stitching between encoders and controllers. This can be achieved either by projecting latent spaces into a common space (Section 3.1), or by directly mapping from one space to another (Section 3.2).

3.1 Relative Representations

The main idea of Relative Representations [23] is to represent latent space elements not in terms of their original embeddings (*absolute* embeddings), but encoding them w.r.t. some selected samples $\mathbf{A}$ called *anchors*. Namely, given an observation o_u and its corresponding latent space produced by the encoder, $\phi_u^i(o_u) = \mathbf{x}_u^i$, the *relative representations* $\mathbf{z}_u^i$ are produced by computing the similarity between the encoded frame and the selected anchors:

$$\begin{aligned} \mathbf{z}_u^i &= sim(\phi_u^i(o_u), \phi_u^i(\mathbf{A}_u)) && (3)\\ &= sim(\mathbf{x}_u^i, \phi_u^i(\mathbf{A}_u)) && (4)\\ &= [sim(\mathbf{x}_u^i, \phi_u^i(\mathbf{A}_u^{(0)})), sim(\mathbf{x}_u^i, \phi_u^i(\mathbf{A}_u^{(1)})), \dots sim(\mathbf{x}_u^i, \phi_u^i(\mathbf{A}_u^{(d)}))], && (5) \end{aligned}$$

where d is the dimension of the latent space and one of the controller's inputs. Per the original paper, we select the cosine similarity as sim function. However, the original anchor selection process assumes the availability of an offline dataset to sample $\mathbf{A}$ from, we generalize this assumption to the online RL setting in Section 3.3.

Intuition This relative representation disregards the latent points' absolute positions, which are heavily influenced by the agent's training process, and instead focuses on the relationships between observations. We expect encoders to produce different absolute latent representations due to environment variations, yet yield *roughly similar relative latent representations*:

$$\begin{aligned} sim(\phi_u^i(o_u), \phi_u^i(\mathbf{A}_u)) &\approx sim(\phi_v^j(o_v), \phi_v^j(\mathbf{A}_v)) && (6)\\ sim(\mathbf{x}_u^i, \phi_u^i(\mathbf{A}_u)) &\approx sim(\mathbf{x}_v^j, \phi_v^j(\mathbf{A}_v)) && (7)\\ \mathbf{z}_u^i &\approx \mathbf{z}_v^j && (8) \end{aligned}$$

Thus, we *train the controller directly on the relative spaces* to produce a universal controller, reusable across a variety of settings.

Module stitching We exploit such quasi-equivalence between the latent relative representation of different agents to perform zero-shot stitching. For example, if we are given policies π_u^i and π_v^j trained with relative representations to act on $\mathcal{M}_u^i = (\mathcal{O}_u, \mathcal{T}_i)$ and $\mathcal{M}_v^j = (\mathcal{O}_v, \mathcal{T}_j)$ respectively, we can combine the encoder ϕ_u^i from π_u^i and the controller ψ_v^j from π_v^j to create a new policy $\hat{\pi}_u^j$ that can act on $\mathcal{M}_u^j = (\mathcal{O}_u, \mathcal{T}_j)$:

$$\hat{\pi}_u^j(o_u) = \psi_v^j(\mathbf{z}_u^i). \tag{9}$$

This can be done because encoders ϕ_u^i and ϕ_v^j produce similar latent spaces, therefore controllers ψ_u^i and ψ_v^j are trained on similar representations. Throughout this paper, we will use encoders and policies trained using PPO.

3.2 Latent Alignment

Consider an environment $\mathcal{M}_u^j$ for which we do not have a trained policy, but we have an encoder ϕ_u^i and a controller ψ_v^j trained for the policies π_u^i and π_v^j, respectively. Relative Representations involves mapping the output of each encoder to a shared latent space, enabling the subsequent training of a universal policy. Instead, employing latent alignment techniques allows us to obtain a direct mapping τ_u^v: $\mathcal{X}_u^i \mapsto \mathcal{X}_v^j$, from the latent space produced by the encoder ϕ_u^i to the one resulting from the encoder ϕ_v^j. This translated space is compatible with the existing ψ_v^j, so that:

$$\tau_u^v(\phi_u^i(\mathbf{o}_u)) \approx \phi_v^j(\mathbf{o}_v) \tag{10}$$

$$\tau_u^v(\mathbf{x}_u^i) \approx \mathbf{x}_v^j \tag{11}$$

Thus, we can reuse encoders and controllers from π_u^i and π_v^j, respectively, to obtain a new policy $\tilde{\pi}_u^j$, without additional training:

$$\tilde{\pi}_u^j(o_u) = \psi_v^j[\tau_u^v(\phi_u^i(\mathbf{o}_u))] \tag{12}$$

Estimating τ The *latent translation* method [20] allows us to estimate τ_u^v. Furthermore, this work suggests that, given two latent spaces $\mathbf{X} \in \mathbb{R}^{n \times d1}$ and $\mathbf{Y} \in \mathbb{R}^{m \times d2}$ from independently trained deep neural networks, the transformation τ that directly maps $\mathbf{X}$ to $\mathbf{Y}$: (i) is mostly orthogonal and (ii) can be estimated from a few corresponding elements between the two spaces. In our work, $\mathbf{X}$ and $\mathbf{Y}$ are produced by ϕ_u and ϕ_v, respectively. As in [20], we use *Singular Value Decomposition* (SVD) to estimate the optimal orthogonal transformation. We leverage this method in the context of online reinforcement learning to zero-shot create policies that can perform in environments with visual-task combinations never seen during training.

3.3 Data Collection

Previous research on latent space translations and their methods [23, 14, 20] relies on supervision provided by samples in partial correspondence between the two domains (*anchors*), which are subsets of the training data. In online reinforcement learning (RL), there is no associated training data to sample from. To address this challenge, we assume the existence of a mapping function that translates observations from one environment $\mathcal{O}u$ to another target environment $\mathcal{O}v$. This mapping can be estimated in various ways, such as through manual annotation, replaying sequences of actions in the environments, or directly estimating the transformation in the observation space (e.g., pixel space). Furthermore, we assume that an agent trained end-to-end to solve a specific task in a specific environment will generate a comprehensive set of observations, providing a reasonable approximation of the entire latent space. Nevertheless, forcing the agent to explore more could be beneficial in this context.

In our experiments, we gather parallel samples either by directly translating the observation in pixel space, when there is a well-defined known visual variation between the environments, or by replaying the same sequence of actions in both environments, that in this case must be deterministic and initialized with the same random seed. We leave to future research other possible approximation techniques for translating observations between different environments.

4 Experiments

In this Section, we assess the zero-shot performance of stitched policies on novel visual-task variations combinations. In Section 4.1 we employ relative representations to study the latent space similarity, providing a qualitative analysis of the projected spaces and showing that aligned frames with different visual variations exhibit similar latent representations. In Section 4.2 we first show how relative models train when compared to standard (absolute models) and then compare their performance when no stitching is applied. Finally, in Section 4.3 we perform a quantitative analysis comparing zero-shot stitching performance for all of our models. This involves: stitching absolute models in a *naive* approach (no latent communication technique applied), and stitching models that employ relative representations and latent translation approaches.

Notation We refer to standard, end-to-end policies using *absolute* representations as *E. Abs*, and to *E. Rel* for end-to-end policies trained using relative representations. We will use *S. Abs*, and to *S. Rel* to instead refer to policies created with zero-shot-stitching through relative representations, where encoders and controllers variations can include seed, background colors and tasks. Finally, we will refer to *S. Transl* for stitched policies obtained by mapping absolute latent spaces via τ projection. Unlike end-to-end models, stitched agent modules come from encoders and controllers trained independently and assembled as detailed in Section 3.1 and Section 3.2.

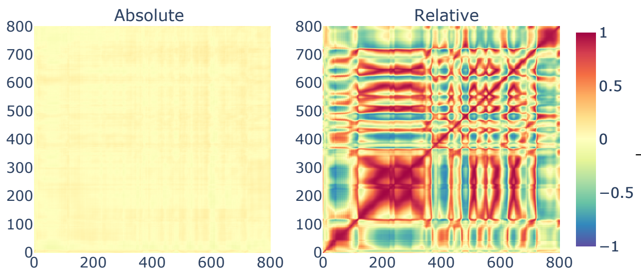
Environment For the following experiments, we consider the CarRacing [17] environment, which simulates driving a car from a 2D top-down perspective around a randomly generated track, focusing on speed while staying on the track. It provides RGB image observations, and uses a discretized action space with five options: steer left, steer right, accelerate, brake, and idle. The agent earns rewards for progress and penalties for going off-track. We modified the environment to enable visual changes (e.g, grass color or camera zoom) and task alterations (e.g., speed limits or different action spaces). The possible visual variations are: background (grass) colors *green*, *red*, *blue* and *far camera zoom*, while tasks are divided in: *standard* and *slow* car dynamics and different action spaces, such as *scrambled*, which use a different action space and therefore a different output order for the car commands, and *no idle*, which removes the “idle” action. Please refer to Appendix A.3 for the implementation and tests with another environment.

Training procedures We train policies using the PPO implementation provided in the CleanRL library [12] with default hypermarameters for both absolute and relative representations.

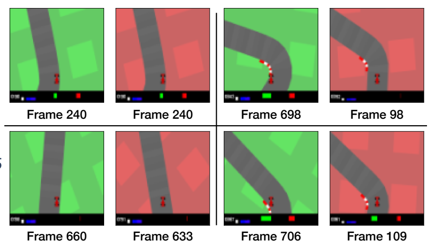
Zero-Shot Stitching Procedure. In Section 3.1 and Section 3.2, we outlined the methodologies for stitching modules together using relative representations and semantic alignment, respectively. We consider the *encoder* to be the group of convolutional layers up to the first flatten layer, while the *controller* is everything that comes immediately after, that is a succession of linear layers and activation functions. Once diverse policies are trained under various conditions, we can generate new policies by assembling independently trained encoders and controllers through zero-shot stitching. The training variations for these individual components are tailored to the requirements of each experimental section; nevertheless, the zero-shot stitching performance evaluation is always on visual-task variation not seen during training. It is crucial to select encoders and controllers that correspond to the specific visual or task variations they were trained on. For instance, when operating within an environment featuring a green background, an encoder trained on that specific visual variation should be utilized. Similarly, for tasks that involve driving a car at low speeds, a controller trained for that specific driving condition must be employed.

4.1 Latent space analysis

We analyze the latent spaces produced by the encoders trained in the CarRacing environment with different grass colors. As described in Section 3.3, we collect parallel observation between the two environment variations by rolling out a policy on an environment and replaying the same sequence of actions in the other environment, setting a fixed seed to ensure reproducibility. In Figure 2a, we report the pairwise cosine similarities of the first ~ 800 frames between the two latent spaces. Therefore, the diagonal shows the similarity between two perfectly aligned frames where the only difference is the grass color. As anticipated, the absolute similarities are consistently low, even for the aligned frames along the diagonal. This is expected because the two spaces are not directly comparable. However, when we unify the spaces using relative representations, we observe a strong similarity on the diagonal, along with some spurious similarities off-diagonal. In Figure 2b, we present the frames associated with high similarity points in the relative space. Although these points are off-diagonal and



(a)



(b)

Figure 2: (a) Comparison between absolute (left) and relative (right) representations produced by the same model. Rows and columns show the cosine similarity between the latent spaces coming from frames of the CarRacing environment with different visual variations (i.e., green and red grass color). Relative representations let similarities emerge not only along the diagonal, where frames are aligned, but also off-diagonal, highlighting similarities between different parts of the track. (b) We report qualitative examples by visualizing frame pairs associated to high similarity regions in (a) (denoted by the frame number). Each pair is semantically similar, even though not in direct correspondence.

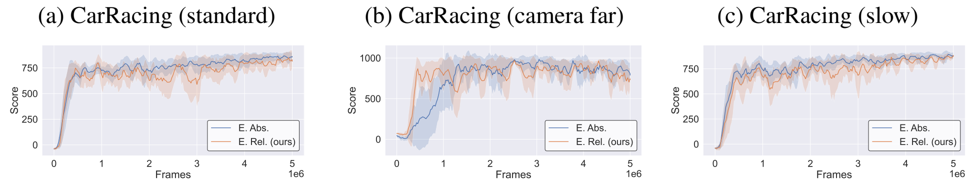


Figure 3: Comparison of E. Abs and E. Rel. training curves. We report three different Car Racing environment variations, noting that in all the training the convergence follows similar tendencies for the two methods and that the relative encoding is not cause of training instability.

Table 1: Mean scores for models trained end-to-end, without stitching. Models trained using relative representations (*Rel*) have comparable performance, with small performance loss on average. Scores are computed over four training seeds and, for each combination, over ten distinct track seeds.

	Visual variations				Task variations		
	green	red	blue	far (green)	slow	scrambled	no idle
E. Abs	829 ± 54	854 ± 26	852 ± 48	872 ± 35	996 ± 6	879 ± 42	889 ± 19
E. Rel (ours)	832 ± 54	797 ± 86	811 ± 21	820 ± 22	624 ± 125	874 ± 20	862 ± 69

not in direct correspondence, they are semantically similar. This shows the effectiveness of relative representations in capturing semantic relationships across different spaces.

In summary, this analysis demonstrates that different policies trained in the Visual RL context exhibit emerging similarities in their latent representations.

4.2 End-to-end performance

Relative representations Figure 3 shows the training curves for CarRacing variations, comparing *E. Abs.* to *E. Rel.* (**ours**) under different conditions. The curves are generated using evaluation scores obtained during training, averaged over four different seeds. Solid lines represent the mean values, and shaded areas indicate the standard deviation. The training stability with relative representations is comparable to that of standard (absolute) training. Furthermore, the results in Table 1 demonstrate that the end-to-end performance of agents is generally comparable to those of absolute models, except for the model trained on the slow task. This table also includes a reference baseline of a model trained on all color variations simultaneously.

These results show that relative representations do not impact training stability and that evaluation performance remains generally comparable. Refer to Appendix A.3.1 for end-to-end tests for some games in the Atari game suite.

Latent Alignment When using latent alignment with relative representations, it is unnecessary to train new models to project the latent variables into a new space. Instead, by mapping directly from one space to another, we can utilize any previously trained model to perform latent alignment. Therefore, we use our Absolute models as a base.

4.3 Zero-shot stitching

The advantages of latent communication techniques become most apparent when performing zero-shot stitching, which allows assembling agents capable of operating in environments they have never encountered during training. Indeed, as shown in Section 4.1, models trained with different seeds or under different settings develop distinct latent representations, making it impossible to naively stitch together independently trained encoders and controllers.

Table 2 presents the zero-shot stitching performance between encoders and controllers across seed, visual, and task variations. Each component is trained with a unique seed. When **Encoder** and **Controller** variations are the same (e.g., green-green), we only consider the performance of stitching between different seeds. Visual and task variations are analyzed independently; hence, the *Task Variations (green)* columns only consider controllers originally trained on a green background, while they are stitched to encoders trained on different background colors.

Both latent communication methodologies significantly outperform the naive baseline, which, to our knowledge, is the only baseline capable of zero-shot generalization to novel environments without further fine-tuning, changes to the agents' architecture or the observations seen during training. Interestingly, agents trained with relative representations maintain high performance when visual variations are the only source of variation. However, there is a marked performance decline with the *slow* task variation. Surprisingly, agents stitched using latent translation exhibit performance comparable to the original ad-hoc end-to-end models across all visual variations and tasks.

In summary, these findings indicate that latent translation is a promising technique for assembling agents capable of operating in novel environment variations. Again, stitching results for the Atari suite can be seen in Appendix A.4.

4.3.1 Computational Advantage

The proposed methods offer a significant computational advantage by reducing the training time required to develop new policies. Indeed, they enable the assembly of new policies from existing ones, making it possible to adapt to novel environments more efficiently.

By reusing policy components, we can create new policies without starting training from scratch. Table 3 illustrates the amount of time saved through zero-shot stitching for the CarRacing models. This table shows the training time required for agents across all visual-task combinations, as previously detailed in Table 2. Normally, training models for every visual-task combination would require 110 hours. However, our approach significantly reduces this time, needing only the highlighted cells in light blue, representing a fraction of the total training time. Indeed, it is sufficient to have at least one encoder and one controller for each variation. This enables the creation of all other agents, saving 88 hours of training. Importantly, this time-saving benefit scales quadratically with the number of visual variations and tasks considered, providing substantial efficiency gains as the complexity of the environment increases.

In summary, latent communication significantly reduces training time in RL by allowing the reuse of policy components to assemble novel policies without the need to train from scratch.

5 Conclusion and limitations

Limitations and Future Works We believe our work opens to several compelling research directions. While we restricted our analysis to controlled environments in favor of interpretability, extending our methodology to more complex and real environments would provide further insights

Table 2: Mean score of new agents created via zero-shot stitching, combining encoders and controller trained with different visual and task variations or training seeds. The original domains for the encoders and the controllers are listed in the columns and rows, respectively. The table compares policies obtained via zero-shot stitching in a naive way (absolute) and with relative and translation methods. Visual variations wise, using any of the stitching methods allow to retain performance that are close to those of the end-to-end models. Task wise, instead, translation generally outperforms the relative approach, especially in the slow task. Both methods greatly outperforms the naive baseline. Each cell reports the mean score and standard deviation calculated across ten seeds for the track, four encoders, and four controllers

			Controller					
			Visual Variations (task standard)			**Task Variations** (green)		
			`green`	`red`	`blue`	`slow`	`scrambled`	`no idle`
Encoder	`green`	*S. Abs*	175 ± 304	167 ± 226	-4 ± 79	148 ± 328	106 ± 217	213 ± 201
		S. Rel	781 ± 108	787 ± 62	794 ± 61	268 ± 14	781 ± 126	824 ± 82
		S. Transl	822 ± 62	786 ± 82	829 ± 49	764 ± 287	846 ± 66	781 ± 72
	`red`	*S. Abs*	157 ± 248	43 ± 205	22 ± 112	83 ± 191	138 ± 244	252 ± 228
		S. Rel	810 ± 52	776 ± 92	803 ± 58	476 ± 430	790 ± 72	817 ± 69
		S. Transl	859 ± 41	807 ± 52	809 ± 60	824 ± 192	838 ± 52	853 ± 50
	`blue`	*S. Abs*	137 ± 225	130 ± 274	11 ± 122	95 ± 128	138 ± 224	144 ± 206
		S. Rel	791 ± 64	793 ± 40	792 ± 48	564 ± 440	804 ± 41	828 ± 50
		S. Transl	839 ± 57	808 ± 70	814 ± 52	746 ± 319	832 ± 60	808 ± 62
	`far`	*S. Abs*	152 ± 204	65 ± 180	2 ± 152	-49 ± 9	351 ± 97	349 ± 66
		S. Rel	527 ± 142	605 ± 118	592 ± 86	303 ± 100	594 ± 39	673 ± 91
		S. Transl	714 ± 45	712 ± 71	727 ± 52	762 ± 131	738 ± 44	626 ± 77

Table 3: Table of training times. We only need to train combinations for the cells highlighted in blue and then perform zero-shot stitching to assemble all the other agents (*totaling 13 hrs*). Normally, it would be required to train agents for each domain-task combination (*totaling 46 hrs*). Visual variations *V1*: green, *V2*: red, *V3*: blue, *V4*: far (green) and task variations *T1*: standard, *T2*: slow, *T3*: no idle, *T4*: scrambled.

	V1	V2	V3	V4
T1	3h	3h	3h	3h
T2	4h	4h	4h	4h
T3	3h	3h	3h	3h
T4	3h	3h	3h	3h

into its scalability. Similarly, all our agents are trained from scratch on specific tasks for consistency. Relying instead on large pre-trained vision models and stitching to different controllers would provide even further computational savings and flexibility.

Conclusions In this work, we propose an analysis of simple yet effective techniques in visual reinforcement learning to stitch encoders and controllers coming from different training regimes. Relying on recent representation learning advancements, we demonstrate how to unify representations learned in different training and compose new agents solving on visual-task pairs never seen at training time. Our extensive experiments highlight that despite RL's training instability, models from different random seeds, visuals, and tasks learn a consistent representation of the world. Our approach provides a paradigm that makes RL more accessible, provides more efficient training, and significantly reduces computational costs.

References

[1] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 225–236, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/01ded4259d101feb739b06c399e9cd9c-Abstract.html`.

[2] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[3] Adrian Csiszarik, Peter Korosi-Szabo, Akos K. Matszangosz, Gergely Papp, and Daniel Varga. Similarity and matching of neural network representations. *ArXiv preprint*, abs/2110.14633, 2021. URL `https://arxiv.org/abs/2110.14633`.

[4] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2169–2176. IEEE, 2017.

[5] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[7] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

[8] Michael Gygli, Jasper Uijlings, and Vittorio Ferrari. Towards reusable network components by learning compatible representations. *AAAI*, 35(9):7620–7629, 2021.

[9] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *International Conference on Robotics and Automation*, 2021.

[10] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[11] Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. In *Conference on Neural Information Processing Systems*, 2021.

[12] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

[13] Pingcheng Jian, Chao Yang, Di Guo, Huaping Liu, and Fuchun Sun. Adversarial skill learning for robust manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2555–2561. IEEE, 2021.

[14] Pingcheng Jian, Easop Lee, Zachary Bell, Michael M Zavlanos, and Boyuan Chen. Policy stitching: Learning transferable robot policies. *arXiv preprint arXiv:2309.13753*, 2023.

[15] Peter Karkus, Mehdi Mirza, Arthur Guez, Andrew Jaegle, Timothy Lillicrap, Lars Buesing, Nicolas Heess, and Theophane Weber. Beyond tabula-rasa: a modular reinforcement learning approach for physically embedded 3d sokoban. *arXiv preprint arXiv:2010.01298*, 2020.

[16] Taylor W Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. *Advances in neural information processing systems*, 30, 2017.

[17] Oleg Klimov. Carracing-v0. *URL https://gym. openai. com/envs/CarRacing-v0*, 2016.

[18] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 991–999. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298701. URL `https://doi.org/10.1109/CVPR.2015.7298701`.

[19] Chenyu Liu, Yan Zhang, Yi Shen, and Michael M Zavlanos. Learning without knowing: Unobserved context in continuous transfer reinforcement learning. In *Learning for Dynamics and Control*, pages 791–802. PMLR, 2021.

[20] Valentino Maiorca, Luca Moschella, Antonio Norelli, Marco Fumero, Francesco Locatello, and Emanuele Rodolà. Latent space translation via semantic alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=pBa70rGHlr`.

[21] Jorge A Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition. *arXiv preprint arXiv:2207.00429*, 2022.

[22] Sharada Mohanty, Jyotish Poonganam, Adrien Gaidon, Andrey Kolobov, Blake Wulfe, Dipam Chakraborty, Gražvydas Šemetulskis, João Schapke, Jonas Kubilius, Jurgis Pašukonis, et al. Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark. *arXiv preprint arXiv:2103.15332*, 2021.

[23] Luca Moschella, Valentino Maiorca, Marco Fumero, Antonio Norelli, Francesco Locatello, and Emanuele Rodolà. Relative representations enable zero-shot latent space communication. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=SrC-nwieGJ`.

[24] Antonio Norelli, Marco Fumero, Valentino Maiorca, Luca Moschella, Emanuele Rodolà, and Francesco Locatello. ASIF: Coupled Data Turns Unimodal Models to Multimodal Without Training. *ArXiv preprint*, abs/2210.01738, 2022. URL `https://arxiv.org/abs/2210.01738`.

[25] Stuart J Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 656–663, 2003.

[26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[28] Christopher Simpkins and Charles Isbell. Composable modular reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4975–4982, 2019.

[29] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

[30] Andrea Tirinzoni, Rafael Rodriguez Sanchez, and Marcello Restelli. Transfer of value functions via variational methods. *Advances in Neural Information Processing Systems*, 31, 2018.

[31] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[32] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[33] Lorenz Wolf and Mirco Musolesi. Augmented modular reinforcement learning based on heterogeneous knowledge. *arXiv preprint arXiv:2306.01158*, 2023.

[34] Muammer Y. Yaman, Sergei V. Kalinin, Kathryn N. Guye, David Ginger, and Maxim Ziatdinov. Learning and predicting photonic responses of plasmonic nanoparticle assemblies via dual variational autoencoders. *ArXiv preprint*, abs/2208.03861, 2022. URL `https://arxiv.org/abs/2208.03861`.

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

[36] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=GY6-6sTvGaf`.

[37] Takuma Yoneda, Ge Yang, Matthew R Walter, and Bradly Stadie. Invariance through latent alignment. *arXiv preprint arXiv:2112.08526*, 2021.

[38] Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032*, 2018.

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

[40] Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

A Appendix

A.1 Environments

CarRacing We use the discretized version of CarRacing, which has the following Action space: **0:** left, **1:** right, **2:** accelerate, **3:** brake, **4:** do nothing. We use the standard dynamics, with small rewards for passing through checkpoints and a small penalty every step otherwise, -100, and end episode if the car goes out of the boundaries. Normal car density (the "weight" of the car) is 1.0. Zoom = 2.7. During our trials, we perform visual (grass color, camera with zoom=1) and behavior variations. The latter are performed via reward constraints (car speed), or by modifying the action space (scrambled actions, smaller action space).

Table 4: Complete list of variations applied for the CarRacing environment.

Variation	Type	ep. length	Other Details
color	visual	1000	colors: green, red, blue
camera	visual	1000	far view (zoom = 1)
slow	task	3000	no negative reward per step, -100 if speed > 35.
scrambled	task	1000	action space: shuffled
no idle	task	1000	action space: noop action removed

Models for all visual and task variations are trained with the same set of hyperparameters

A.2 Atari Games

We base our environment suite on NaturalEnv [38], which originally allowed to replace default game background with images or solid colors. Our selection of games is: *Breakout, Boxing, Pong*. For all we select the *NoFrameskip-v4* version, with default action space. Models are trained for 10000000 steps and default training parameters as in [12].

A.3 Latent communication on the Atari game suite

To test the generalizability of our method we also perform stitching tests with the following Atari games: Breakout, Boxing, Pong from the NaturalEnv collection [38], which allow background customization with solid colors; the actions are game-specific. In the Breakout environment, scores typically range between 0 and 200-300, representing a satisfactory final score. In Boxing, scores fall within the range of [-100, 100], where a score of 100 indicates that the agent defeats the opponent without sustaining any hits. For Pong, scores range from [-21, 21], with 21 signifying victory over the opponent without conceding a single point. As we did for the CarRacing environment, we first evaluate end-to-end performance of standard and relative models followed by the stitching evaluation.

Training with relative representations The training procedure is the same described in section 4.2. Training curves comparing E. Abs. to E. Rel. (**ours**) for the Atari suite with visual variations are shown in 4. Also in this case, training using relative representations is comparable to standard training, with the exception of the Breakout environment where the curve tends to grow more slowly.

(a) Breakout (b) Boxing (c) Pong

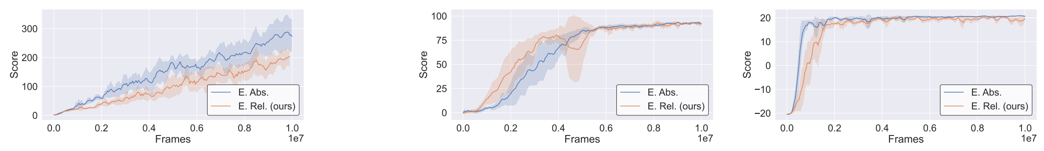


Figure 4: Comparison of E. Abs and E. Rel. training curves for some games in the Atari suite. We see that in all the training the convergence follows similar tendencies for the two methods, and the relative encoding is not cause of training instability.

A.3.1 End-to-end performance

Results can be seen in Table 5. Performance of relative models for Pong and Boxing are comparable to those of the absolute models. Breakout, however, has much lower scores. We attribute this to the higher visual complexity caused by the numerous bricks in the level.

Table 5: Episode mean scores for models trained end-to-end, therefore when no stitching is performed. Models trained using relative representations (*Rel*) have comparable performance, with small performance loss on the average.

		Visual variations		
		plain	green	red
Pong	*E. Abs*	21 ± 0	21 ± 0	21 ± 0
	E. Rel (ours)	21 ± 0	20 ± 1	21 ± 0
Boxing	*E. Abs*	95 ± 2	95 ± 3	96 ± 2
	E. Rel (ours)	95 ± 3	93 ± 4	88 ± 6
Breakout	*E. Abs*	298 ± 63	262 ± 61	132 ± 19
	E. Rel (ours)	146 ± 60	77 ± 25	119 ± 135

A.4 Zero-shot stitching

Zero-shot stitching performance Table 6 presents zero-shot stitching evaluation. Although relative models outperform absolute ones in all the environments, there is a significant performance drop in

Table 6: Mean score of new agents created via zero-shot stitching, combining encoders and controller trained with different visual and task variations or training seeds. The original domains for the encoders and the controllers are listed in the columns and rows, respectively. The table compares policies obtained via zero-shot stitching in a naive way (absolute) and with relative and translation methods. Visual variations wise, using any of the stitching methods allow to retain performance that are close to those of the end-to-end models. Task wise, instead, translation generally outperforms the relative approach, especially in the slow task. Both methods greatly outperforms the naive baseline. Each cell reports the mean score and standard deviation calculated across ten seeds for the track, four encoders, and four controllers

			Controller								
			Pong			**Boxing**			**Breakout**		
			`plain`	`green`	`red`	`plain`	`green`	`red`	`plain`	`green`	`red`
Encoder	`plain`	*S. Abs*	-21 ± 0	-21 ± 0	-21 ± 1	-29 ± 10	-25 ± 22	-33 ± 5	8 ± 6	12 ± 5	8 ± 5
		S. Rel	0 ± 20	-2 ± 19	7 ± 17	65 ± 39	11 ± 54	46 ± 40	71 ± 73	16 ± 18	17 ± 11
		S. Transl	-16 ± 6	-10 ± 11	-14 ± 10	89 ± 5	49 ± 42	46 ± 53	265 ± 59	217 ± 59	172 ± 100
	`green`	*S. Abs*	-20 ± 1	-21 ± 0	-21 ± 0	-18 ± 17	-21 ± 15	-33 ± 10	9 ± 7	15 ± 4	7 ± 6
		S. Rel	-1 ± 20	-7 ± 19	8 ± 18	66 ± 38	42 ± 36	58 ± 27	28 ± 19	64 ± 100	44 ± 59
		S. Transl	-3 ± 16	13 ± 12	-13 ± 10	86 ± 6	47 ± 49	56 ± 58	169 ± 99	229 ± 98	165 ± 93
	`red`	*S. Abs*	-21 ± 0	-21 ± 0	-21 ± 0	-27 ± 17	-20 ± 16	-38 ± 13	9 ± 6	13 ± 5	8 ± 5
		S. Rel	1 ± 18	-3 ± -18	6 ± 20	62 ± 40	20 ± 56	49 ± 39	17 ± 9	25 ± 28	14 ± 10
		S. Transl	-7 ± 16	-7 ± 13	-20 ± 1	80 ± 22	52 ± 32	38 ± 52	31 ± 26	75 ± 58	145 ± 92

the mean performance indicated by the lower mean scores. The high the standard deviation, however, signifies that some of the models are still able to perform well in some cases. We argue that the high precision required by Atari games might be the reason for the performance drop in stitching, as even small noise in the encoders' latent spaces can bring to minor mistakes in action predictions, which in turn can bring to a losing condition in the game. Meanwhile, the CarRacing environment is far more accommodating, and in the event of a mistake, the policy can compensate in subsequent actions.

A.5 Computing

We trained all of our models on an RTX 3080 with an Intel i7-9700K CPU @ 3.60GHz and 64 GB of RAM. All of our models were trained with the CleanRL implementation of PPO, running 16 environments in parallel.

CarRacing We trained all of our CarRacing models for a total of 5 million steps. Training took around 3h for every model, except for the slow task, which took 4, mainly because episodes were set to be longer to allow the car to complete the track, and therefore evaluation episodes took longer to complete.

Atari Suite Games in the Atari suite were trained for 10 million steps. Pong and Breakout required around 2h30m, boxing 2h40m.