

π 2VEC: POLICY REPRESENTATION WITH SUCCESSOR FEATURES

Gianluca Scarpellini* †
Istituto Italiano di Tecnologia

Ksenia Konyushkova
Google DeepMind

Claudio Fantacci
Google DeepMind

Tom Le Paine
Google DeepMind

Yutian Chen
Google DeepMind

Misha Denil
Google DeepMind

†: Work done during an internship at Google DeepMind

*: Corresponding author gianluca.scarpellini@iit.it

ABSTRACT

This paper introduces π 2vec, a method for representing black box policies as comparable feature vectors. Our method combines the strengths of foundation models that serve as generic and powerful state representations and successor features that can model the future occurrence of the states for a policy. π 2vec represents the behaviors of policies by capturing statistics of how the behavior evolves the features from a pretrained model, using a successor feature framework. We focus on the offline setting where both policies and their representations are trained on a fixed dataset of trajectories. Finally, we employ linear regression on π 2vec vector representations to predict the performance of held out policies. The synergy of these techniques results in a method for efficient policy evaluation in resource constrained environments.

1 INTRODUCTION

Robot time is an important bottleneck in applying reinforcement learning in real life robotics applications. Constraints on robot time have driven progress in sim2real, offline reinforcement learning (offline RL), and data efficient learning. However, these approaches do not address the problem of policy evaluation which is often time intensive as well. Various proxy metrics were introduced to eliminate the need for real robots in the evaluation. For example, in sim2real we measure the performance in simulation (Lee et al., 2021). In offline RL we rely on Off-policy Evaluation (OPE) methods (Gulcehre et al., 2020; Fu et al., 2021). For the purpose of deploying a policy in the real world, recent works focused on Offline Policy Selection (OPS), where the goal is to select the best performing policy relying only on offline data. While these methods are useful for determining coarse relative performance of policies, one still needs time on real robot for more reliable estimates (Levine et al., 2020).

Our proposed π 2vec aims at making efficient use of the evaluation time. Efficient offline policy evaluation and selection is relevant in reinforcement learning projects, where researchers often face the challenge of validating improvements. π 2vec enables researchers to make more informed decisions regarding which new policy iterations to prioritize for real-world testing or to identify and discard less promising options early in the development process. In particular, we predict the values of unknown policies from a set of policies with known values in an offline setting, where a large dataset of historical trajectories from other policies and human demonstrations is provided. The last step requires policies to be represented as vectors which are comparable and thus can serve as an input to the objective function. Prior work from Konyushkova et al. (2021) represents policies by the actions that they take on a set of canonical states, under the assumption that similar actions in similar states imply similar behaviour. However, this assumption is sometimes violated in practice. This work aims at finding more suitable representation by characterizing the policies based on how they change the environment.

To represent policies, our method π 2vec combines two components: successor features and foundation models. We adapt the framework of Q-learning of successor features (Barreto et al., 2017) to the

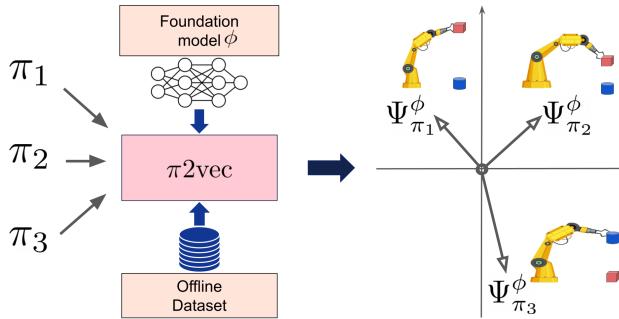


Figure 1: π 2vec method relies on the successor feature framework, that we adopt in combination with a dataset of offline demonstrations and a visual foundation model ϕ . π 2vec represents each policy π_i as a feature vector $\Psi_{\pi_i}^\phi \in \mathbb{R}^n$. $\Psi_{\pi_i}^\phi$ encodes the expected behavior of a policy when deployed on an agent.

offline setting by applying the Fitted Q evaluation (FQE) algorithm (Le et al., 2019) which is typically used for off-policy evaluation (OPE). In this work the features for individual states are provided by a general purpose pretrained visual foundation model (Bommasani et al., 2021). The resulting representations can be used as a drop in replacement for the action-based representation used by Konyushova et al. (2021).

Our experiments show that π 2vec achieves solid results in different tasks and across different settings. To summarize, our main contributions are the following:

- We propose π 2vec, a novel policy representation of how the policies change the environment, which combines successor features, foundation models, and offline data;
- We evaluate our proposal through extensive experiments predicting return values of held out policies in 3 simulated and 2 real environments. Our approach outperforms the baseline and achieves solid results even in challenging real robotic settings and out-of-distribution scenarios;
- We investigate various feature encoders, ranging from semantic to geometrical visual foundation models, to show strengths and weaknesses of various representations for the task at hand.

2 RELATED WORK

Representation of black-box policies. In this paper, our objective is to create vector representations for policies to predict their performance. We treat policies as black-boxes (i.e., no access to internal state, parameters, or architectures) that yield actions for a given observation. It is important to emphasize that our objective differs from representation learning for RL (Schwarzer et al., 2020; Jaderberg et al., 2016; Laskin et al., 2020), as we focus on representing policies rather than training feature encoders for downstream tasks.

Konyushova et al. (2021) studied a setting where the goal is to identify the best policy from a set of policies with a dataset of offline experience and limited access to the environment. Each policy is represented by a vector of actions at a fixed set of states. While this representation performs well in certain applications, it may not be the most effective for predicting policy performance. For instance, consider two policies that generate random actions at each state. These policies do not exhibit meaningfully different behaviour, so for policy evaluation purposes, we expect them to be similar. However, the action policy representation categorizes these policies as different. This paper proposes a method to address this limitation by measuring trajectory-level changes in the environment.

In BCRL (Chang et al., 2022), a state-action feature representation is proposed for estimating policy performance. However, the representation of each policy is independent of other policies and thus cannot be employed to regress the performance of new policies given a set of evaluated policies.

Offline Policy Evaluation. Off-policy Evaluation (OPE) aims to evaluate a policy given access to trajectories generated by another policy. It has been extensively studied across many domains (Li et al., 2010; Theocharous et al., 2015; Kalashnikov et al., 2018; Nie et al., 2019). Broad categories of OPE methods include methods that use importance sampling (Precup, 2000), binary classification (Irpan et al., 2019), stationary state distribution (Liu et al., 2018), value functions (Sutton et al., 2016;

Le et al., 2019), and learned transition models (Zhang et al., 2021), as well as methods that combine two or more approaches (Farajtabar et al., 2018). The main focus of the OPEs approaches is on approximating the return values function for a trained policy, while π 2vec goes beyond classical OPE and focuses on encoding the behavior of the policy as vectors, in such a way that those vectors are comparable, to fit a performance predictor.

Foundation Models for Robotics. Foundation models are large, self-supervised models (Bommasani et al., 2021) known for their adaptability in various tasks (Sharma et al., 2023). We compare three representative foundation models (Radford et al., 2021; Dosovitskiy et al., 2021; Doersch et al., 2022). Our proposal, π 2vec, is independent of the feature encoder of choice. Better or domain-specific foundation models may improve results but are not the focus of this study.

3 METHODOLOGY

3.1 OVERVIEW

Our setting is the following. We start with a large dataset of historical trajectories \mathbb{D} , and a policy-agnostic state-feature encoder $\phi : \mathbb{S} \rightarrow \mathbb{R}^N$. Given a policy π , our objective is to use these ingredients to create a policy embedding $\Psi_\pi^\phi \in \mathbb{R}^N$ that represents the behavior of π (and can be used to predict its performance).

We aim to create this embedding offline, without running the policy π in the environment. Although we can evaluate π for any state in our historical dataset \mathbb{D} , we emphasize that we do not have access to any on policy trajectories from π , which significantly complicates the process of creating an embedding that captures the behavior of π .

Our method π 2vec has three steps:

1. Choose a policy-agnostic state-feature encoder ϕ . We discuss several options for ϕ below and in the experiments; however, π 2vec treats the policy-agnostic state-feature encoder as a black box, allowing us to leverage generic state-feature representations in our work.
2. Train a policy-specific state-feature encoder $\psi_\pi^\phi : (\mathbb{S}, \mathcal{A}) \rightarrow \mathbb{R}^N$. In this step we combine the policy-agnostic state-feature encoder ϕ , and the policy π , to create policy-specific state-feature encoder by training on the historical dataset \mathbb{D} . The policy-specific state features $\psi_\pi^\phi(s)$ capture statistics of how π would change the environment were it to be run starting from the state s .
3. Aggregate the policy-specific state-features to create state-agnostic policy features Ψ_π^ϕ that represent the behavior of π in a state-independent way.

Using the steps outlined above we can collect a dataset of policy-specific state-independent features paired with measured policy performance. This dataset can be used to train a model that predicts the performance of a policy from its features using supervised learning. Because we compute features for a policy using only offline data, when we receive a new policy we can compute its policy-specific state-independent features and apply the performance model to predict its performance **before running it in the environment**. In the following sections we expand on each step.

3.2 POLICY-AGNOSTIC STATE FEATURES

The role of the state-feature encoder ϕ is to produce an embedding that represents an individual state of the environment. In this paper we focus on state encoders $\phi : I \rightarrow \mathbb{R}^N$ that consume single images I . Generically our method is agnostic to the input space of the state-feature encoder, but practically speaking it is convenient to work with image encoders because that gives us access to a wide range of pretrained generic image encoders that are available in the literature.

We also consider a few simple ways to construct more complex features from single image features. When each state provides multiple images we embed each image separately and sum the result to create a state embedding. We also consider creating embeddings for transitions (s, s') by computing $\Delta\phi(s, s') \triangleq \phi(s') - \phi(s)$. Both cases allow us to leverage features from pretrained models.

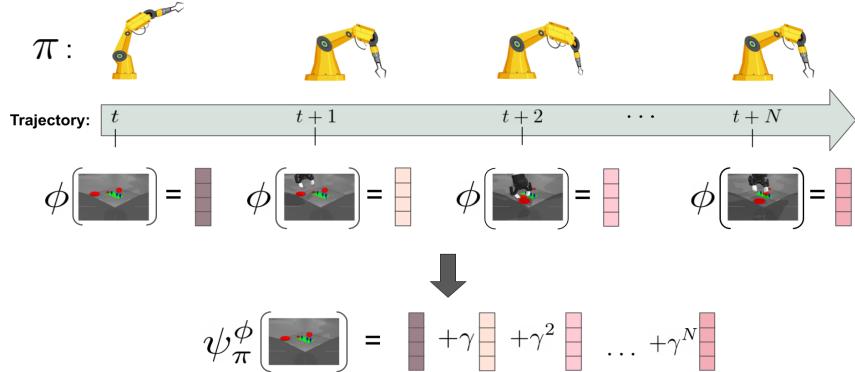


Figure 2: Given a trajectory from the dataset of offline demonstrations, we train successor feature $\psi_\pi^\phi(s_t)$ to predict the discounted sum of features $\sum_i \gamma^i \phi(s_{t+i})$, where ϕ is a visual feature extractor and π is a policy. Intuitively, $\phi(s_t)$ represents semantic changes in the current state of the environment s_t , while successor feature $\psi_\pi^\phi(s_t)$ summarizes all *future* features encoded by ϕ if actions came from policy π .

3.3 POLICY-SPECIFIC STATE FEATURES

The next step is to use the policy-agnostic state-feature encoder ϕ that provides a generic representation for individual states to train a policy-specific state-feature encoder $\psi_\pi^\phi : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}^N$ that represents the effect that π would have on the environment if it were run starting from the given state.

The work of Dayan (1993); Barreto et al. (2017) on successor features provides a basis for our approach to policy representation. We briefly review successor features here, and comment below on how we make use of them. We refer the reader to recent literature covering successor features Lehnert & Littman (2020); Brantley et al. (2021); Reinke & Alameda-Pineda (2021).

Suppose that the reward function for a task can be written as a linear function

$$r(s, a, s') = \langle \phi(s, a, s'), \mathbf{w}_{\text{task}} \rangle, \quad (1)$$

where $\phi(s, a, s') \in R^N$ encodes the state-transition as a feature vector and $\mathbf{w}_{\text{task}} \in R^N$ are weights. Barreto et al. (2017) observe that if the reward can be factored as above, then the state-action-value function for a policy π can be written as

$$Q^\pi(s, a) = \mathbb{E}_{(s'|s) \sim D, a \sim \pi(s)} \left[\sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i, s_{i+1}) \right] = \langle \psi_\pi^\phi(s, a), \mathbf{w}_{\text{task}} \rangle, \quad (2)$$

where

$$\psi_\pi^\phi(s, a) = \mathbb{E}_{(s'|s) \sim D, a \sim \pi(s)} \left[\sum_{i=t}^{\infty} \gamma^{i-t} \phi(s_i, a_i, s_{i+1}) \right], \quad (3)$$

$(s|s') \sim D$ is a transition from the environment, and γ is the discount factor. The corresponding state-value function is $V^\pi(s) \triangleq Q^\pi(s, \pi(s)) = \langle \psi_\pi^\phi(s, \pi(s)), \mathbf{w}_{\text{task}} \rangle \triangleq \langle \psi_\pi^\phi(s), \mathbf{w}_{\text{task}} \rangle$. We will use the notation $\psi_\pi^\phi(s) \triangleq \psi_\pi^\phi(s, \pi(s))$ frequently throughout the remainder of the paper.

The value of $\psi_\pi^\phi(s)$ is known as the *successor features* of the state s under the policy π . Successor features were originally motivated through the above derivation as a way of factoring the value function of a policy into a task-independent behavior component (the successor features) that is independent of the task, and a task-dependent reward component that is independent of behavior.

For our purposes we will mostly ignore the reward component (although we return to it in one of the experiments) and focus on the behavior term shown in Equation 3. This term is interesting to us for two reasons. First, we can see by inspection of the RHS that the value of $\psi_\pi^\phi(s) = \psi_\pi^\phi(s, \pi(s))$ represents the behavior of π as a future discounted sum of state features along a trajectory obtained by running π beginning from the state s . In other words, ψ_π^ϕ represents the behavior of π in terms of

the features of the states that π will encounter, where the state features are themselves given by the policy-agnostic state-feature encoder from the previous section. Figure 2 summarizes the relationship between successor features ψ and state encoders ϕ .

Second, Equation 3 satisfies the Bellman equation meaning that the function $\psi_\pi^\phi(s, a)$ can be estimated from off-policy data in a task-agnostic way using a modified version of Q-learning, where the scalar value reward in ordinary Q-learning is replaced with the vector valued transition features $\phi(s, a, s')$. We rely on Fitted Q Evaluation (FQE, Le et al. (2019)), an *offline* Q-learning based algorithm, and thus, we obtain a representation of policy behavior purely from data without executing the policy in the environment. Given a dataset \mathbb{D} and a policy π , FQE estimates its state-action-value function $Q^\pi(s, a)$ according to the following bootstrap loss:

$$L(\theta) = \mathbb{E}_{(s, a, r, s') \sim \mathbb{D}, a' \sim \pi(s')} [\|\psi_\theta^\pi(s, a) - (\phi(s, a, s') + \psi_\theta^\pi(s', a'))\|^2]. \quad (4)$$

FQE is simple to implement and it performs competitively with other OPE algorithms in a variety of settings (Fu et al., 2021) including simulated and real robotics domains (Paine et al., 2020; Konyushova et al., 2021). We use FQE with our historical dataset \mathbb{D} to train a policy-specific state-action-feature network $\psi_\pi^\phi(s, a)$, which we then use as the policy-specific state-feature encoder $\psi_\pi^\phi(s) \triangleq \psi_\pi^\phi(s, \pi(s))$ by plugging in the policy action.

3.4 STATE-AGNOSTIC POLICY FEATURES

We obtain a single representation Ψ_π^ϕ of a policy π from the state-dependent successor features $\psi_\pi^\phi(s)$ for that policy by averaging the successor features over a set of canonical states:

$$\Psi_\pi^\phi = \mathbb{E}_{s \sim D_{\text{can}}} [\psi_\pi^\phi(s)], \quad (5)$$

where D_{can} is a set of states sampled from historical trajectories. We sample the canonical states set $D_{\text{can}} \subset \mathbb{D}$ uniformly from our historical dataset, as in Konyushova et al. (2021), ensuring that each canonical state comes from a different trajectory for better coverage. We average successor features over the same set D_{can} for every policy. The intuition behind this representation is that $\psi_\pi^\phi(s)$ represents the expected change that π induces in the environment by starting in the state s ; by averaging over D_{can} , Ψ_π^ϕ represents an aggregated average effect of the behavior of π .

3.5 PERFORMANCE PREDICTION

We aim at predicting the performance of novel, unseen policies. We begin with a dataset of historical policies for which we have measured performance $\Pi = \{\dots, (\pi_i, R_i), \dots\}$. For each policy in this dataset we create an embedding using the above procedure to obtain a new dataset $\{\dots, (\Psi_{\pi_i}^\phi, R_i), \dots\}$ and then train a performance model $\hat{R}_i = f(\Psi_{\pi_i}^\phi)$ using supervised learning. Given a new policy π_* we can then predict its performance before running it in the environment by computing the π 2vec features for the new policy using the above procedure and applying the performance model to obtain $\hat{R}_* = f(\Psi_{\pi_*}^\phi)$.

4 EXPERIMENTAL SETUP

In this section we describe the feature encoders, domains, and evaluation procedures, followed by details about our baselines. More details about our architecture, domains, and training procedure can be found in the Appendix.

Feature encoder. Firstly, the *Random* feature encoder employs a randomly-initialized ResNet-50 (He et al., 2016). Random features are trivial to implement, and achieve surprisingly strong performance in many settings (Rahimi & Recht, 2007). Here they serve as a simple baseline.

Next, we explore with *CLIP* (Radford et al., 2021). CLIP-network is trained to match image and text embeddings on a large-scale dataset of image caption pairs. Intuitively, by aligning image and text features, CLIP network is trained to encode high-level semantic information.

Visual Transformers (ViT) (Dosovitskiy et al., 2021) treat images as a 1D sequence of patches and learn visual features via an attention mechanism. In our experiments the visual transformer is pre-trained on imagenet classification.

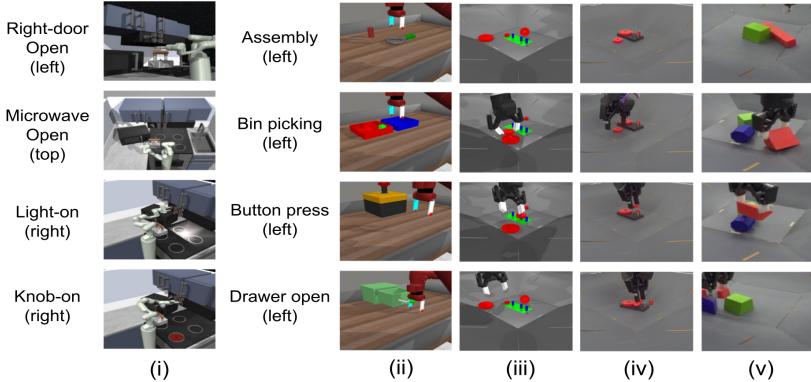


Figure 3: We adopt 5 environments. (i) Kitchen: 5 tasks (Knob-on, Left door open, light on, microwave open, and right door open) and 3 points of views. (ii) Metaworld: 4 tasks (assembly, button press, bin picking, and drawer open) and 3 points of views. (iii) Insert gear in simulation (iii) and (iv) on a real robot. (v) RGB stacking on a real robot.

Lastly, we explore *Track-any-point (TAP)* (Doersch et al., 2022), a general-purpose network for point tracking in videos. The network is pre-trained to track arbitrary points over video sequences and as a result it learns to understand the low-level geometric features in a scene. We use an attention layer trained to select task-relevant features from the TAP model to reduce dimensionality.

This set of feature encoders spans a spectrum of properties as they are created by optimising different objectives. At one extreme CLIP features are trained to align image features with a text description, and encode the semantics of the image. At the other extreme TAP features are trained to track points in videos, and capture low level geometric and texture information. ViT features are in the middle, they need to encode both semantics and local texture to accomplish classification tasks. Depending on the environment and task at hand, better state representation is likely to result in better prediction properties of π 2vec. We leave the question of finding the best representation as future work.

Domains. We present extensive experiments to support π 2vec’s capabilities across three simulated domains—Insert Gear (Sim), Metaworld, and Franka-Kitchen, and two real domains—Insert Gear (Real) and RGB Stacking (Figure 3). In each domain we use a dataset of offline human demonstrations (Metaworld and Kitchen) and held out policies trajectories (RGBStacking and Insert Gear) for training policy representations. Each policy is treated as a black-box where we do not have any prior knowledge about the architecture or training parameters. We provide further details in Supplementary.

Evaluation. We assess the quality of the policy representations by measuring the ability of the model f to predict the performance of held out policies (see Section 3.5). We adopt k-fold cross validation over the set Π and report results averaged over cross validation folds. Following previous works on offline policy evaluation (Paine et al., 2020; Fu et al., 2021), we adopt the following three complementary metrics. We report further details in the Supplementary.

- **Normalized Mean Absolute Error (NMAE)** measures the accuracy of the prediction w.r.t. the ground-truth. We adopt MAE instead of MSE to be robust to outliers and we normalize the error to be in range between the return values for each environment. Lower is better.
- **Rank Correlation** measures how the estimated values correlate with the ground-truth. Correlation focuses on how many evaluations on the robot are required to find the best policy. Higher is better.
- **Regret@1** measures the performance difference between the best policy and the predicted best policy, normalized w.r.t. the range of returns values for each environment. Lower is better.

Correlation and Regret@1 are the most relevant metric for evaluating π 2vec on OPS. On the other hand, NMAE refers to the accuracy of the estimated return value and is suited for OPE.

Baselines. The problem in this paper is to represent policies in such a way that the representations can be used to predict the performance of other policies given the performance of a subset of policies. Importantly, to address this problem the representation should 1) encode the behavior of the policy, 2) in a way that is comparable with the representations of other policies, and 3) does not require online

Table 1: We compare π 2vec and *Actions* representations for Insert-gear (real) and Insert-gear (sim) tasks, as well as for the RGB stacking environment. The table shows the performance and confidence intervals for different feature representations and encoders.

Representation	NMAE \downarrow	Correlation \uparrow	Regret@1 \downarrow
RGB Stacking			
<i>Actions</i>	0.261 ± 0.045	0.785 ± 0.177	0.074 ± 0.083
VIT	0.224 ± 0.063	0.775 ± 0.146	0.036 ± 0.116
Δ VIT	0.344 ± 0.050	0.030 ± 0.332	0.375 ± 0.206
CLIP	0.330 ± 0.042	0.342 ± 0.293	0.325 ± 0.180
Δ CLIP	0.287 ± 0.048	0.583 ± 0.126	0.079 ± 0.126
Random	0.304 ± 0.066	0.330 ± 0.334	0.226 ± 0.177
Δ Random	0.325 ± 0.109	0.352 ± 0.348	0.190 ± 0.180
Insert gear (real)			
<i>Actions</i>	0.252 ± 0.028	-0.545 ± 0.185	0.578 ± 0.148
Random	0.275 ± 0.027	-0.207 ± 0.267	0.360 ± 0.162
CLIP	0.198 ± 0.030	0.618 ± 0.136	0.267 ± 0.131
Δ CLIP	0.253 ± 0.228	-0.109 ± 0.100	0.429 ± 0.100
Insert gear (sim)			
<i>Actions</i>	0.174 ± 0.015	0.650 ± 0.056	0.427 ± 0.172
Random	0.215 ± 0.026	0.555 ± 0.104	0.422 ± 0.143
TAP	0.164 ± 0.022	0.680 ± 0.095	0.359 ± 0.184
VIT	0.224 ± 0.025	0.402 ± 0.129	0.448 ± 0.195
Δ VIT	0.255 ± 0.024	0.218 ± 0.139	0.457 ± 0.153
CLIP	0.180 ± 0.031	0.502 ± 0.068	0.298 ± 0.126
Δ CLIP	0.189 ± 0.020	0.586 ± 0.077	0.314 ± 0.147

data. Active Offline Policy Selection (AOPS) (Konyushova et al., 2021) stands alone as a notable work that delves into policy representation from offline data with the task of deciding which policies should be evaluated in priority to gain the most information about the system. AOPS showed that representing policies according to its algorithm leads to faster identification of the best policy. In AOPS’s representation, which we call “*Actions*”, policies are represented through the actions that the policies take on a fixed set of canonical states. We build *Actions* representation as follows. We run each policy π on the set of states D_{can} sampled from historical trajectories. Next, we concatenate the resulting set of actions $\{\pi(s)\}_{s \in D_{\text{can}}}$ into a vector.

To the best of our knowledge, the *Actions* representation is the only applicable baseline in the setting that we adopt in this paper. Nevertheless, OPE methods that estimate policy performance from a fixed offline dataset are standard methodology in offline RL literature. Although these methods do not take the full advantage of the problem setting in this paper (the performance of some of the policies is known) they can still serve for comparison. In this paper, we compared against FQE which is a recommended OPE method that strikes a good balance between performance (it is among the top methods) and complexity (it does not require a world model) (Fu et al., 2021).

5 RESULTS

We report results for various feature encoders for Insert gear (sim and real) and RGBStacking. Similarly, we report averaged results for over 4 tasks and 3 point of view for Metaworld and over 5 tasks and 3 point of view for Kitchen. Along with results for each feature encoder, we report the average results of picking the **best** feature encoder for each task (BEST- ϕ). Similarly, we report as BEST-CLIP and BEST-VIT the average results when adopting the best feature encoder between CLIP/VIT and Δ CLIP/ Δ VIT. We identify the **best** feature encoder for a task by conducting cross-validation on previously evaluated policies and pick the best encoder in terms of regret@1.

Our results demonstrate that (i) π 2vec outperforms the *Actions* baseline models consistently across real and simulated robotics environments and multiple tasks, showcasing the framework’s effectiveness in representing policies. Furthermore, we demonstrate the applicability to real-world robotic settings, specifically in the challenging Insert Gear (Real) environment, where even underperforming policies contribute to improved policy evaluation. We show that choosing the best model as

Table 2: We evaluate π 2vec on Metaworld and Kitchen. The results are averaged over all settings and confidence intervals are reported. BEST- ϕ is π 2vec average performance assuming that we adopt the best ϕ in terms of regret@1 for each task-POV setting. Similarly, BEST-CLIP and BEST-VIT are the best feature encoder between CLIP/VIT and Δ CLIP/ Δ VIT.

Representation	NMAE \downarrow	Correlation \uparrow	Regret@1 \downarrow
Metaworld			
<i>Actions</i>	0.424 ± 0.058	0.347 ± 0.152	0.232 ± 0.078
CLIP	0.340 ± 0.035	0.254 ± 0.143	0.250 ± 0.076
Δ CLIP	0.325 ± 0.092	0.286 ± 0.154	0.232 ± 0.086
BEST-CLIP	0.309 ± 0.027	0.351 ± 0.130	0.194 ± 0.076
VIT	0.303 ± 0.030	0.280 ± 0.146	0.263 ± 0.091
Δ VIT	0.315 ± 0.026	0.162 ± 0.169	0.325 ± 0.084
BEST-VIT	0.298 ± 0.029	0.300 ± 0.147	0.244 ± 0.092
Random	0.366 ± 0.086	0.043 ± 0.150	0.375 ± 0.108
BEST- ϕ	0.289 ± 0.018	0.460 ± 0.099	0.153 ± 0.060
Kitchen			
<i>Actions</i>	0.857 ± 0.128	0.326 ± 0.128	0.221 ± 0.089
CLIP	0.417 ± 0.032	0.021 ± 0.219	0.317 ± 0.081
Δ CLIP	0.352 ± 0.026	0.260 ± 0.216	0.244 ± 0.081
BEST-CLIP	0.333 ± 0.025	0.346 ± 0.200	0.197 ± 0.076
VIT	0.385 ± 0.030	0.030 ± 0.244	0.322 ± 0.095
Δ VIT	0.344 ± 0.025	0.155 ± 0.234	0.251 ± 0.082
BEST-VIT	0.321 ± 0.024	0.412 ± 0.228	0.151 ± 0.068
Random	0.382 ± 0.033	-0.017 ± 0.225	0.334 ± 0.080
BEST- ϕ	0.392 ± 0.053	0.591 ± 0.203	0.070 ± 0.045

a feature-extractor greatly improves results (ii). Finally, we adopt π 2vec to solve Equation 2 and estimate policies’ return values in the Metaworld’s **assembly** environment, without relying on any ground-truth data (iii). Although the successor feature assumption of linearity of rewards is violated, π 2vec still ranks policies competitively in the offline setting when compared to FQE. In the Appendix, we provide an intuition for choosing the best ϕ based on the correlation between task difficulty (iv), and we study the effect of different dataset types, such as demonstrations and trajectories from held out policies (v). We investigate π 2vec’s generalization capabilities (vi), including out-of-distribution scenarios (vii). We also demonstrate that π 2vec represents random policies close in the feature space (viii), and that π 2vec is robust to canonical state coverage (ix) and effective with online data (x).

(i) π 2vec consistently outperforms *Actions*. We compare π 2vec and *Actions* across all scenarios. Our method outperforms *Actions* representation when predicting values of unseen policies in both real robotics scenarios–RGB stacking and insert-gear (real)–as shown in Table 1. In the former, Ψ^{VIT} achieves regret@1 of 0.036 compared to *Actions*’ 0.074, with a relative improvement of 51%. In the latter, Ψ^{CLIP} improves over *Actions* by achieving regret@1 0.267 compared to *Actions*’ 0.578 and drastically outperform *Actions* in terms of correlation by achieving +0.618 compared to *Actions*’ –0.545. π 2vec performs robustly on insert gear (real) despite policies’ performances for this task vary greatly (see supplementary for per-task policies performances).

We also evaluate our approach in the simulated counterpart *Insert Gear (Sim)*. In this environment, Ψ^{CLIP} and Ψ^{TAP} achieve regret@1 of 0.314 and 0.359 respectively, compared to *Actions* 0.427. We underline the dichotomy between geometrical and semantic features: Ψ^{TAP} performs best in terms of NMAE and Correlation, while Ψ^{CLIP} outperforms in Regret@1. These results highlight how various ϕ compare depending on setting, type of task, and policy performance.

(ii) When evaluating across multiple settings, selecting ϕ leads to better results. We compare π 2vec with different foundation models across 12 Metaworld settings and 15 Kitchen settings. Table 2 reports the average results across all settings for Metaworld and Kitchen. In Metaworld, we notice that *Actions* performs on par with Ψ^{CLIP} , Ψ^{VIT} , and their respective variations Δ CLIP and Δ VIT, in terms of correlation and regret@1, while our approach consistently outperforms *Actions* in terms of NMAE. As these domains have less state variability, *Actions* represent policies robustly. We test CLIP/ Δ CLIP

Table 3: We extend π 2vec to the fully-offline setting and test it on Metaworld assembly task (left, right, and top). We report results and confidence intervals. In this setting, performances of all policies are unknown.

Representation	NMAE \downarrow	Correlation \uparrow	Regret@1 \downarrow
Assembly (left)			
FQE	0.338 \pm 0.062	0.125 \pm 0.218	0.424 \pm 0.260
π 2vec	8.306 \pm 0.155	0.360 \pm 0.097	0.215 \pm 0.079
Assembly (right)			
FQE	0.270 \pm 0.093	-0.029 \pm 0.351	0.504 \pm 0.071
π 2vec	2.116 \pm 0.056	0.154 \pm 0.115	0.319 \pm 0.080
Assembly (top)			
FQE	0.322 \pm 0.012	-0.251 \pm 0.516	0.609 \pm 0.228
π 2vec	0.492 \pm 0.006	0.555 \pm 0.106	0.149 \pm 0.071

and VIT/ Δ VIT on previously evaluated policies for each task through cross-validation to identify the best feature encoder for the task in terms of regret@1. We report $\Psi^{\text{BEST-CLIP}}$ and $\Psi^{\text{BEST-VIT}}$ as the average results over the best among CLIP/VIT and Δ CLIP/ Δ VIT. $\Psi^{\text{BEST-CLIP}}$ achieves regret@1 0.194 and correlation 0.351, outperforming *Actions* representation. We highlight that the choice of ϕ is critical, since Ψ^{random} —using a randomly-initialized ResNet50 as feature extractor—underperforms. Moreover, π 2vec with the best ϕ drastically improves, achieving regret@1 of 0.153 compared to *Actions* 0.232. We notice similar improvements when evaluating on Kitchen’s 15 settings. Table 2 compares choosing the *BEST* ϕ w.r.t. to VIT and CLIP, and against *Actions*. In Kitchen, Ψ^{VIT} outperforms Ψ^{CLIP} and *Actions*, while $\Psi^{\text{BEST}-\phi}$ achieves the overall best results.

(iii) π 2vec enables fully-offline policy selection. By directly modelling the relationship between successor features and returns, we avoid the linear reward assumption of the original successor features work. This is preferable since rewards are generally not linearly related to state features. However, this restricts our method to settings where some policies’ performance is known. To evaluate performance in a fully-offline setting, we fit a linear model the task **reward** $\hat{r} = \langle \phi(s), \mathbf{w}_{\text{task}} \rangle$ given the state’s feature representation $\phi(s)$, as in Equation 2 from the original successor features work. Next we predict policies returns as $\hat{R}_i = \langle \Psi_{\pi_i}^\phi, \mathbf{w}_{\text{task}} \rangle$. We compare our approach to FQE in Table 3 and find that while our method’s return predictions are inaccurate (as evidenced by the high NMAE), it still performs well in ranking policies (higher Correlation and lower Regret@1).

6 CONCLUSION

We presented π 2vec, a framework for offline policy representation via successor features. Our method treats the policy as a black box, and creates a representation that captures statistics of how the policy changes the environment rather than its idiosyncrasies. The representations can be trained from offline data, and leverage the pretrained features of visual foundation models to represent individual states of the environment. In our experiments, we represented policies by relying on visual features from semantic (CLIP), geometric (TAP), and visual (VIT) foundation models. We showed that π 2vec outperforms previously used *Actions* based representations and generalizes to fully-offline settings. Overall, our experiments showcase the effectiveness and versatility of π 2vec in representing policies and its potential for various applications in reinforcement learning.

Moving forward, we acknowledge that finding the optimal combination of these elements remains an ongoing challenge. Future work should explore diverse foundation models, offline learning algorithms for successor feature training, and dataset choices. Fine-tuning the feature encoder ϕ along with ψ_ϕ^π is interesting but pose challenges, as each feature encoder would specialize to predict features for a specific policy, resulting in policy representations that are independent and not comparable. We leave end-to-end fine-tuning as future work. Integrating π 2vec into AOPS framework (Konyushova et al., 2021) for enhanced offline policy selection is another intriguing avenue. Additionally, extending π 2vec to augment the Generalized Policy Improvement (Barreto et al., 2017) in offline settings presents exciting research opportunities.

REFERENCES

- 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.
- Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.
- Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- Kianté Brantley, Soroush Mehri, and Geoff J Gordon. Successor feature sets: Generalizing successor representations across policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11774–11781, 2021.
- Jonathan Chang, Kaiwen Wang, Nathan Kallus, and Wen Sun. Learning bellman complete representations for offline policy evaluation. In *International Conference on Machine Learning*, pp. 2938–2971. PMLR, 2022.
- Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624, 1993.
- Carl Doersch, Ankush Gupta, Larisa Markeeva, Adrià Recasens, Lucas Smaira, Yusuf Aytar, João Carreira, Andrew Zisserman, and Yi Yang. Tap-vid: A benchmark for tracking any point in a video. *arXiv preprint arXiv:2211.03726*, 2022.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=YicbFdNTTy>.
- Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.
- Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. pp. 1447–1456, 2018.
- Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.
- Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, Ziyu Wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, et al. Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*, 2021.
- Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:7248–7259, 2020.
- Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- Alexander Irpan, Kanishka Rao, Konstantinos Bousmalis, Chris Harris, Julian Ibarz, and Sergey Levine. Off-policy evaluation via off-policy classification. *Advances in Neural Information Processing Systems*, 32, 2019.

- Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.
- Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *ICLR (Poster)*, 2015. URL <http://dblp.uni-trier.de/db/conf/iclr/iclr2015.html#KingmaB14>.
- Ksenia Konyushova, Yutian Chen, Thomas Paine, Caglar Gulcehre, Cosmin Paduraru, Daniel J Mankowitz, Misha Denil, and Nando de Freitas. Active offline policy selection. *Advances in Neural Information Processing Systems*, 34:24631–24644, 2021.
- Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, Jul 2020.
- Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, pp. 3703–3712. PMLR, 2019.
- Alex X Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, et al. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *5th Annual Conference on Robot Learning*, 2021.
- Lucas Lehnert and Michael L Littman. Successor features combine elements of model-free and model-based reinforcement learning. *The Journal of Machine Learning Research*, 21(1):8030–8082, 2020.
- Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. pp. 661–670, 2010.
- Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. *Advances in Neural Information Processing Systems*, 31, 2018.
- Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.
- Xinkun Nie, Emma Brunskill, and Stefan Wager. Learning when-to-treat policies. *arXiv preprint arXiv:1905.09751*, 2019.
- Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*, 2020.
- Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pp. 80, 2000.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.
- Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

- Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *Transactions on Machine Learning Research (TMLR)*, 2022.
- Chris Reinke and Xavier Alameda-Pineda. Successor feature representations. *arXiv preprint arXiv:2110.15701*, 2021.
- Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.
- Mohit Sharma, Claudio Fantacci, Yuxiang Zhou, Skanda Koppula, Nicolas Heess, Jon Scholz, and Yusuf Aytar. Lossless adaptation of pretrained vision models for robotic manipulation. 2023.
- Richard S Sutton, A Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17(1): 2603–2631, 2016.
- Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. 2015.
- Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020.
- Michael R Zhang, Tom Le Paine, Ofir Nachum, Cosmin Paduraru, George Tucker, Ziyu Wang, and Mohammad Norouzi. Autoregressive dynamics models for offline policy evaluation and optimization. *arXiv preprint arXiv:2104.13877*, 2021.

7 APPENDIX

7.1 DOMAINS

The *Metaworld* Yu et al. (2020) and *Kitchen* Gupta et al. (2019) domains are widely known in the literature. They contain many tasks from multiple views, however, the variability among tasks is low. For example, the robotic arm in Metaworld is initialized within a narrow set of positions, while in Kitchen the object positions are fixed. The task in real robot *RGB Stacking* Lee et al. (2021); Reed et al. (2022) is to stack the red object on top of the blue object with green object as a distractor, where objects are of various geometric shapes. This task is difficult because the objects have unusual shapes and may be positioned at any point in the workspace. We also consider a challenging *gear insertion* task in sim and real where the objective is to pick up a gear of certain size (from an arbitrary position in the workspace) in the presence of other gears and insert it onto a specific shaft on the gear assembly base (arbitrary position in real, fixed in sim). We describe data and policies for each domain below.

Insert Gear (Sim). We use 18 policies for Insert Gear (Sim) task in the simulated environment. We take an intermediate and the last checkpoint for each policy. We collect trajectories with $T = 300$ steps from a single π and train all ψ_{π_i} on those trajectories. The state s consists of two images, one from a left camera and one from a right camera, and proprioception sensing. All the policies in this domain have the following architecture. Image observations are encoded using a (shared) ResNet, and proprioception is embedded using an MLP. Then, two embeddings are concatenated and further processed by an MLP, followed by an action head.

Insert Gear (Real). The observable state consists of three points of view: a camera on the left of the basket, a camera on the right of the basket, and an egocentric camera on the gripper. The state also includes proprioception information about the arm position. The setup corresponds to the medium gear insertion task described in the work of Du et al. (2023). We conduct experiments on the Insert Gear (Real) task on a real robotic platform by evaluating 18 policies. We collect a dataset of trajectories with a hold-out policy trained on human demonstrations. Next, we train our set of policies on this dataset and we evaluate π 2vec. The state and he policy architecture are the same as in Insert Gear (Sim).

RGB stacking. We use 12 policies trained with behavior cloning on a previously collected dataset of demonstrations for RGB stacking task with a real robotic arm. Each policy is trained with a variety of hyperparameters. The state s consists of images from the basket cameras, one on the left and one on the right, and a first person camera mounted on the end-effector, and proprioception sensing. For training π 2vec, we adopted an offline dataset of trajectories. We collected the trajectories by running a policy trained on human demonstrations. Trajectory length varies between 800 and 1000 steps. We built the evaluation dataset D_{can} by sampling 5,000 trajectories and then sampling one state from each of them. Policy architecture follows the description in Lee et al. (2021).

Metaworld. For Metaworld, we consider 4 tasks: assembly, button press, bin picking, and drawer open. We use 3 points of views (left, right, and top), as specified in Sharma et al. (2023); Nair et al. (2022). For each task-camera pair, we adopt 12 policies as proposed by Sharma et al. (2023) for the particular task and point of view. The policies vary by the hyperparameters used during training (learning rate, seed, and feature extractor among NFnet, ViT, and ResNet). Next, we train a successor feature network ψ_{π}^* for each policy π on a cumulative dataset of demonstrations for all tasks and views. At evaluation, we build D_{can} by uniformly sampling one state from each demonstration.

Franka-Kitchen. For Kitchen, we consider 5 tasks: Knob-on, Left door open, light on, microwave open, and door open with 3 points of views: left, right, and top, as provided by previous works (Sharma et al., 2023; Nair et al., 2022). For each task and point of view, we use human demonstrations provided by Fu et al. (2020). We also adopt policies $\{\pi_i\}$ proposed by Sharma et al. (2023). Each policy solves each task using proprioception and image information from a single point of view, and the policies vary by the hyperparameters used during training (learning rate, seed, and feature extractor among NFnet, ViT, and ResNet). Additional details can be found in Sharma et al. (2023).

Table 4 reports mean and standard deviation of the expected return values for the policies under consideration. We highlight that Metaworld and Insert gear have high standard-deviation (standard

Table 4: We report the mean return value and its standard-deviation for the policies for each environment. For Metaworld and Kitchen, we take the average mean and standard-deviation across all tasks and points of views. \dagger Number of policies per task-camera pair.

Environment	N. policies	Return values
Insert-gear (sim)	18	0.60 ± 0.51
Insert-gear (real)	18	2.29 ± 1.71
RGB Stacking	12	179.41 ± 8.21
Metaworld	12 \dagger	215.88 ± 164.99
Kitchen	12 \dagger	-47.86 ± 20.05

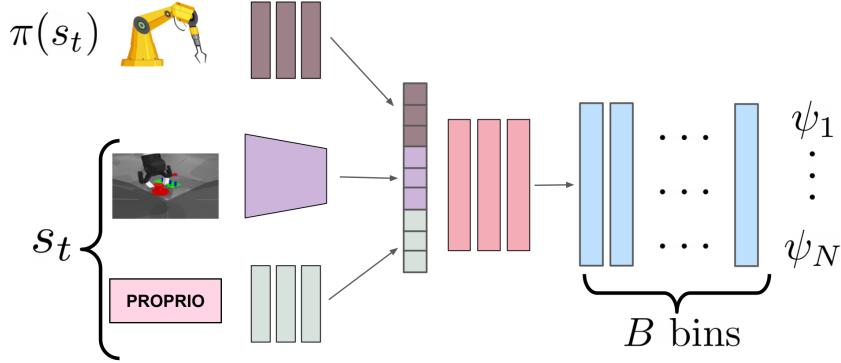


Figure 4: We implement ψ_π^ϕ as a neural-network. First, we encode state s_t —consisting observations and proprioception—and policy actions $\pi(s_t)$ into feature vectors. Next, we concatenate the features and input the resulting vector to a multi-layer perceptron. ψ_π^ϕ outputs a vector of $B \times N$ dimensions, where B number of bins of the distribution and N is the dimension of the feature vector $\phi(s_t)$. We reshape the output into a matrix, where each row i represents a histogram of probabilities of size B for the successor feature ψ_i .

deviation is 75% or more of the mean return value), as we have extremely good and extremely bad policies. On the contrary, return values for RGB Stacking and Kitchen vary less, i.e., most of the policies for these environments achieve similar performance.

7.2 ARCHITECTURE AND TRAINING DETAILS

The architecture of successor features network $\psi_\pi^\phi(s)$ for a policy π is illustrated in Figure 4. The network takes state-action pairs as input; it encodes actions and proprioception with a multi-layer perceptron, and visual observations using a ResNet50. When the environment observation consists of multiple images (e.g., multiple camera views of the same scene), we embed each image separately and average the resulting vectors. We concatenate the state and action encodings and process the resulting feature vector with a MLP. Finally, the network $\psi_\pi^\phi(s)$ outputs a vector of dimension $R^{N \times B}$, where N is the dimension of the feature vector $\phi(s)$ represented as a distribution over B bins¹. $\psi_\pi^\phi(s)$ returns N histograms, where each histogram ψ_i approximates the distribution of the discounted sum of feature ϕ_i over policy π . For each environment, we inspect the range of values assumed by ϕ to find the min (lower) and max (upper) bound of the histogram. At inference time, we take the expected value of each histogram to compute the successor features vector.

We train $\psi_\pi^\phi(s, a)$ using an FQE with a distributional objective (Le et al., 2019; Bellemare et al., 2017). Training the successor features network only requires offline data (separate for each domain) and does not require any online interactions. We train the network for 50,000 steps for Metaworld and Kitchen and 100,000 steps for RGB Stacking, Insert Gear (Sim), and Insert Gear (Real). We adopt

¹We use $B = 51$ bins in all experiments.

Table 5: We conduct additional experiments on Metaworld environment when using a dataset of trajectories for training π 2vec. As expected, enhancing the dataset leads to better performances. We report as BEST-CLIP and BEST-VIT the average results when adopting the best feature encoder between CLIP/VIT and Δ CLIP/ Δ VIT in terms of regret@1.

Representation	Dataset	NMAE	Correlation	Regret@1
<i>Actions</i>	-	0.424 ± 0.058	0.347 ± 0.152	0.232 ± 0.078
Δ VIT	trajectories	0.296 ± 0.024	0.399 ± 0.128	0.214 ± 0.064
Δ CLIP	trajectories	0.278 ± 0.014	0.469 ± 0.096	0.189 ± 0.075
BEST- ϕ	trajectories	0.269 ± 0.017	0.507 ± 0.105	0.187 ± 0.073
Δ VIT	best	0.322 ± 0.029	0.447 ± 0.126	0.191 ± 0.074
Δ CLIP	best	0.274 ± 0.026	0.537 ± 0.166	0.177 ± 0.175
BEST- ϕ	best	0.231 ± 0.016	0.615 ± 0.086	0.135 ± 0.052

different training steps because Metaworld and Kitchen are more predictable than RGB Stacking and Insert gear and less demonstrations are provided. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of $3e^{-5}$ and a discount factor of $\gamma = 0.99$. For evaluation, we adopt 3-fold cross-validation in all experiments.

7.3 METRICS

We adopt three common metrics Fu et al. (2021): NMAE, correlation, regret@1.

- **Normalized Mean Absolute Error (NMAE)** is defined as the difference between the value and estimated value of a policy:

$$\text{NMAE} = |V^\pi - \hat{V}^\pi|, \quad (6)$$

where V^π is the true value of the policy, and \hat{V}^π is the estimated value of the policy.

- **Regret@1** is the difference between the value of the best policy in the entire set, and the value of the best predicted policy. It is defined as:

$$\text{Regret@1} = \max_{i \in 1:N} V_i^\pi - \max_{j \in (1:N)} V_j^\pi. \quad (7)$$

- **Rank Correlation** (also Spearman’s ρ) measures the correlation between the estimated rankings of the policies’ value estimates and their true values. It can be written as:

$$\text{Corr} = \frac{\text{Cov}(V_{1:N}^\pi, \hat{V}_{1:N}^\pi)}{\sigma(V_{1:N}^\pi)\sigma(\hat{V}_{1:N}^\pi)}. \quad (8)$$

Correlation and regret@1 are the most relevant metrics for evaluating π 2vec on Offline Policy Selection (OPS), where the focus is on ranking policies based on values and selecting the best policy.

Regret@1 is commonly adopted in assessing performances for Offline Policy Selection, as it directly measures how far off the best-estimated policy is to the actual best policy.

Correlation is relevant for measuring how the method ranks policies by their expected return value. By relying on methods that achieve higher correlation and thus are consistent in estimating policy values, researchers and practitioners can prioritize more promising policies for online evaluation.

On the other hand, NMAE refers to the accuracy of the estimated return value. NMAE is especially significant when aiming at estimating the true value of a policy and is suited for Offline Policy Evaluation (OPE), where we are interested to know the values of each policy. We assess π 2vec’s representation in both settings, showing that π 2vec consistently outperforms the baseline in both metrics. We improve the discussion on metrics in the Appendix of the manuscript.

7.4 ADDITIONAL EXPERIMENTS

(iv) **Correlation between task difficulty and ϕ .** We notice that policy performance varies widely in Insert Gear (Sim) and Insert Gear (Real), as most of the policies fail to solve the task (see

Table 6: We evaluate π 2vec and *Actions* baseline when comparing intermediate checkpoints for a set of policies for Metaworld assembly task and left point-of-view. We highlight that *Actions* representations are similar in such scenario, leading to poor generalization when evaluating on unseen and potentially different policies.

Representation	NMAE	Correlation	Regret@1
<i>Actions</i>	0.724 ± 0.268	-0.189 ± 0.137	0.034 ± 0.014
Δ CLIP	0.570 ± 0.173	0.190 ± 0.361	0.029 ± 0.034

Table 7: We conduct an experiment in an out-of-domain setting. We represent a set of policies that were trained for Metaworld assembly task from *right* point-of-view with *Actions* and $\Psi^{\Delta\text{CLIP}}$. Next, we evaluate how those representations predict the return values if policies are fed with images from a different point-of-view (left camera in our experiment). $\Psi^{\Delta\text{CLIP}}$ outperforms *Actions* in terms of regret@1 and NMAE, supporting our intuition that π 2vec is robust in an out-of-distribution setting.

Representation	NMAE	Regret@1
<i>Actions</i>	0.363 ± 0.055	0.475 ± 0.100
Δ CLIP	0.227 ± 0.078	0.300 ± 0.106

Table 8: We intuitively expect that π 2vec represents random policies close in feature space w.r.t. *Actions* representations, as random policies do not change the environment in any meaningful way. We conduct an experiment with random policies for Metaworld assembly-left to provide quantitative evidence supporting our interpretation.

Representation	Average distance	Max distance
<i>Actions</i>	0.39	0.62
Random	0.11	0.17
Δ CLIP	0.03	0.22

supplementary for per-task policies performances). The gap is evident when compared to the average return value for RGB Stacking, where standard deviation is negligible. Our intuition is that in hard-to-solve scenarios the actions are often imperfect and noisy. This interpretation would explain poor performance of the *Actions* baseline. The comparison of Ψ^{CLIP} and Ψ^{ViT} across environments suggests a correlation between the choice of ϕ and policies return values. Ψ^{CLIP} performs better than Ψ^{ViT} in Insert Gear (Sim), Insert Gear (Real), and Metaworld, where we report the highest standard deviation among policies performances. Ψ^{CLIP} is robust when the task is hard and most of the policies fail to solve it. On the other hand, Ψ^{ViT} is the best option in Kitchen and RGB stacking, where the standard deviation of policies returns is low or negligible.

(v) Studying the performance of π 2vec with different datasets. We investigate how modifications of the dataset for training π 2vec improves performance in Metaworld. Intuitively, if the training set for π 2vec closely resembles the set of reachable states for a policy π , solving Equation 2 leads to a more close approximation of the real successor feature of π in (s, a) . We empirically test this claim as follows. We collect 1,000 trajectories for each task-view setting using a pre-trained policy. Next, we train successor features network ψ_π^ϕ for each policy π and feature encoder ϕ , and represent each policy as Ψ_π^ϕ . Table 5 reports results on Metaworld when training π 2vec with the aforementioned dataset (named *trajectory* in the Table). In this setting, Ψ^{CLIP} and Ψ^{ViT} outperform both their counterpart trained on demonstrations and *Actions* representation, reaching respectively regret@1 of 0.189 and 0.187. These results slightly improve if we assume to opt for the best dataset for each task and take the average, as reported under *best* dataset in Table 5. Overall, choosing the best feature encoder ϕ and best dataset for any given task leads to the best performing $\Psi^{\text{BEST}-\phi}$ with correlation 0.615 and regret@1 0.135—improving over *Actions* by 0.26 and 0.10 respectively.

(vi) π 2vec generalizes better while *Actions* works with policies are very similar. We explore how *Actions* and π 2vec compare in the special scenario where all policies are similar. We take 4 intermediate checkpoints at the end of the training for each policy as a set of policies to represent.

Table 9: We evaluate π 2vec when trained on online data for *Insert Medium (gear)*. Results for π 2vec from offline data on this task are reported in Table 1.

Representation	NMAE	Correlation	Regret@1
<i>Actions</i>	0.174	0.650	0.427
CLIP	0.158	0.627	0.288
Random	0.348	0.425	0.302
TAP	0.172	0.686	0.205

Our intuition is that intermediate checkpoints for a single policy are similar to each other in how they behave. Next, we represent each checkpoint with Ψ^{CLIP} and *Actions*. We compare cross-validating the results across all checkpoints w.r.t. training on checkpoints for 3 policies and testing on checkpoints for the holdout policy. Table 6 reports results of this comparison on Metaworld’s assembly-left task. We notice that *Actions* representations fail to generalize to policies that greatly differ from the policies in the training set. Fitting the linear regressor with *Actions* achieves a negative correlation of -0.189 and regret@1 0.034. On the other hand, Ψ^{CLIP} is robust to unseen policies and outperforms Actions with positive correlation 0.190 and lower regret of 0.029.

(vii) **π 2vec performs in out-of-distribution scenarios.** We conduct another investigation to explore π 2vec performances in a out-of-distribution setting. We hypothesize that π 2vec represents policies in meaningful ways even when those policies are deployed in settings that differ from the training set, thanks to the generalisation power of foundation models. Table 7 compares Δ CLIP and *Actions* in evaluating policies trained for Metaworld’s assembly-right and tested in Metaworld’s assembly-left. π 2vec achieves reget@1 of 0.300 and NMAE of 0.227, outperforming *Actions* by 0.175 and 0.136 respectively. We leave further exploration of π 2vec in out-of-distribution settings for further works.

(viii) **π 2vec represents random policies close in the representation space.** Intuitively, we expect that random policies do not modify the environment in a meaningful way. Therefore, their representations should be closer to each other compared to the similarity between the more meaningful trained policies. We investigate this claim as follows. We provide a set of 6 trained policies and a set of 6 random policies for Metaworld assembly-left. We compute the average and max distance among the random policies representations, normalized by the average intraset distance between trained policies representations. We compare our Ψ^{CLIP} and Ψ^{Random} with *Actions*. Table 8 reports the results that clearly support our intuition. Both $\Psi^{\Delta\text{CLIP}}$ and Ψ^{Random} represent random policies close to each other, as the average distance of their representation is respectively 0.03 and 0.11 and the maximum distance 0.22 and 0.17 respectively. On the other hand, if we represent policies with *Actions*, the representations average and maximum distances are 0.39 and 0.62, meaning that random policies are represented far apart from each other.

(ix) **Canonical state coverage.** We sample canonical states uniformly from the dataset that was used for training offline RL policies. Even though there is some intuition that selecting canonical states to represent the environment better can be beneficial, even simple sampling at random worked well in our experiments. We conduct further experiments to ablate the state coverage. We adopt demonstrations from Metaworld Assembly task and adopt the initial state of each trajectory for π 2vec’s and *Actions* representation. By adopting the initial state of a trajectory, π 2vec cannot rely on the state coverage. We report the results in Table 10. We show that π 2vec is robust to state coverage, showing SOTA performances even when the canonical states coverage is limited to the first state of each demonstration.

(x) **π 2vec from online data.** We ideally want to evaluate policies without deployment on a real robot, which is often time-consuming and can lead to faults and damages. Nonetheless, we explore π 2vec capabilities for representing policies from online data. For each policy π , we collect a dataset of trajectories by deploying the policy on the agent. Next, we train ψ_π on the dataset of π ’s trajectories and compute its π 2vec’s representation. Table 9 reports results when training π 2vec on online data on Insert Gear (sim) task. We show that π 2vec’s performances improve with respect to the offline counterpart. This result is expected: a better dataset coverage leads to improved results, as we also showed in (v).

Table 10: We test π 2vec robustness to canonical states coverage. We estimate π 2vec’s representations for Metaworld’s assembly task (left, right, and top point-of-views) by using only the first state of the demonstrations for each task.

Representation	NMAE ↓	Correlation ↑	Regret@1 ↓
Assembly (left)			
CLIP	0.381	0.359	0.287
Δ CLIP	0.260	0.592	0.087
Random	0.422	0.252	0.46
VIT	0.366	0.26	0.347
Actions	0.356	0.503	0.222
Assembly (right)			
CLIP	0.363	0.023	0.365
Δ CLIP	0.242	0.582	0.096
Random	0.334	0.313	0.212
VIT	0.27	0.345	0.304
Actions	0.405	0.369	0.263
Assembly (top)			
CLIP	0.463	0.270	0.330
Δ CLIP	0.305	0.594	0.078
Random	0.394	0.277	0.328
VIT	0.418	0.020	0.417
Actions	0.414	0.554	0.106