Finer Behavioral Foundation Models via Auto-Regressive Features and Advantage Weighting

Anonymous authors
Paper under double-blind review

Keywords: Unsupervised RL, offline training, autoregressive features, successor measures

Summary

The forward-backward representation (FB) is a recently proposed framework (Touati et al., 2023; Touati & Ollivier, 2021) to train *behavior foundation models* (BFMs) that aim at providing zero-shot efficient policies for any new task specified in a given reinforcement learning (RL) environment, without training for each new task. Here we address two core limitations of FB model training.

First, FB, like all successor-feature-based methods, relies on a *linear* encoding of tasks: at test time, each new reward function is linearly projected onto a fixed set of pre-trained features. This limits expressivity as well as precision of the task representation. We break the linearity limitation by introducing *auto-regressive features* for FB, which let fine-grained task features depend on coarser-grained task information. This can represent arbitrary nonlinear task encodings, thus significantly increasing expressivity of the FB framework.

Second, it is well-known that training RL agents from offline datasets often requires specific techniques.We show that FB works well together with such offline RL techniques, by adapting techniques from (Nair et al., 2020b; Cetin et al., 2024) for FB. This is necessary to get non-flatlining performance in some datasets, such as DMC Humanoid.

As a result, we produce efficient FB BFMs for a number of new environments. Notably, in the D4RL locomotion benchmark, the generic FB agent matches the performance of standard single-task offline agents (IQL, XQL). In many setups, the offline techniques are needed to get any decent performance at all. The auto-regressive features have a positive but moderate impact, concentrated on tasks requiring spatial precision and task generalization beyond the behaviors represented in the trainset.

Together, these results establish that generic, reward-free FB BFMs can be competitive with single-task agents on standard benchmarks, while suggesting that expressivity of the BFM is not a key limiting factor in the environments tested.

Contribution(s)

1. Provide a succinct but precise list of the contribution(s) of the paper. Use contextual notes to avoid implications of contributions more significant than intended and to clarify and situate the contribution relative to prior work (see the examples below). If there is no additional context, enter "None". Try to keep each contribution to a single sentence, although multiple sentences are allowed when necessary. If using complete sentences, include punctuation. If using a single sentence fragment, you may omit the concluding period. A single contribution can be sufficient, and there is no limit on the number of contributions. Submissions will be judged mostly on the contributions claimed on their cover pages and the evidence provided to support them. Major contributions should not be claimed in the main text if they do not appear on the cover page. Overclaiming can lead to a submission being rejected, so it is important to have well-scoped contribution statements on the cover page.
   Context: None

Finer Behavioral Foundation Models via Auto-Regressive Features and Advantage Weighting

Anonymous authors
Paper under double-blind review

Abstract

The forward-backward representation (FB) is a recently proposed framework (Touati et al., 2023; Touati & Ollivier, 2021) to train *behavior foundation models* (BFMs) that aim at providing zero-shot efficient policies for any new task specified in a given reinforcement learning (RL) environment, without training for each new task. Here we address two core limitations of FB model training.

First, FB, like all successor-feature-based methods, relies on a *linear* encoding of tasks: at test time, each new reward function is linearly projected onto a fixed set of pre-trained features. This limits expressivity as well as precision of the task representation. We break the linearity limitation by introducing *auto-regressive features* for FB, which let fine-grained task features depend on coarser-grained task information. This can represent arbitrary nonlinear task encodings, thus significantly increasing expressivity of the FB framework.

Second, it is well-known that training RL agents from offline datasets often requires specific techniques.We show that FB works well together with such offline RL techniques, by adapting techniques from (Nair et al., 2020b; Cetin et al., 2024) for FB. This is necessary to get non-flatlining performance in some datasets, such as DMC Humanoid.

As a result, we produce efficient FB BFMs for a number of new environments. Notably, in the D4RL locomotion benchmark, the generic FB agent matches the performance of standard single-task offline agents (IQL, XQL). In many setups, the offline techniques are needed to get any decent performance at all. The auto-regressive features have a positive but moderate impact, concentrated on tasks requiring spatial precision and task generalization beyond the behaviors represented in the trainset.

Together, these results establish that generic, reward-free FB BFMs can be competitive with single-task agents on standard benchmarks, while suggesting that expressivity of the BFM is not a key limiting factor in the environments tested.

1 Introduction

The forward-backward representation (FB) is a recently proposed framework (Touati et al., 2023; Touati & Ollivier, 2021) to train behavior foundation models (BFMs) from offline data. BFMs promise to provide zero-shot efficient policies for any new task specified in a given reinforcement learning (RL) environment, beyond the tasks and behaviors in the training set. This contrasts with traditional offline RL and imitation learning approaches, which are trained to accomplish individual target tasks, with no mechanism to tackle new tasks without repeating the full training procedure.

The FB approach strives to learn an agent that recovers many possible behaviors in a given environment, based on learning *successor measure* representations, without any reward signal. After training, an FB agent can be prompted via several kinds of task description: an explicit reward function, a goal state, or even a single demonstration (Pirotta et al., 2023).

However, in its current formulation, FB has been shown effective only for toy problems and relatively simple locomotion tasks and when trained on undirected datasets collected via unsupervised exploration (Burda et al., 2019).

Here, we tackle two core limitations of the "vanilla" FB framework, namely, the difficulty to learn from complex offline datasets, and the linear correspondence between tasks and features. As a result, we can build high-performing FB BFMs for a series of new environments. Our main contributions are the following:

1. We show that the vanilla FB policy optimization leads to poor performance when learning from datasets made of a few near-optimal examples for a few specific tasks. This failure is exactly analogous to naively using online RL algorithms in the offline setting, a well-studied problem (Fujimoto et al., 2019; Wu et al., 2019; Levine et al., 2020). This explains the poor performance of vanilla FB on the *D4RL benchmark*, as reported in recent unsupervised RL works (Park et al., 2024; Frans et al., 2024).
2. Accordingly, we introduce a new policy optimization step for FB, to improve learning from offline datasets demonstrating complex behaviors. In particular, we integrate an improved version of *advantage-weighted regression* (Nair et al., 2020a), together with recent advancements from the offline RL literature (Cetin et al., 2024) and additional algorithmic refinements to FB (Section 3.2). We show these changes are crucial to train FB with common offline datasets beyond pure RND exploration and scale to more challenging environments, often making the difference between near-zero and satisfactory performance (Section 4.3).
3. We overcome a core theoretical limitation of FB and, more generally, of all *successor features* frameworks (Barreto et al., 2017; Borsa et al., 2018): their linear correspondence between reward functions and task representation vectors. Indeed, in these frameworks, at test time, the reward function is linearly projected onto a fixed set of pre-trained features. This results in "reward blurring" and limits spatial precision in the task representation (Touati & Ollivier, 2021). We introduce a new *auto-regressive* encoding of task features (Section 3.1), that breaks the linearity constraint by letting fine-grained task features depend on coarser-grained task information. This allows for universal approximation of any arbitrary task space (Appendix, Theorem B.3). We show that auto-regressive features make a moderate but systematic difference when learning *new* test tasks far from ones considered to build the datasets, or for tasks requiring precise goal-reaching (eg, 15% relative increase for goal-reaching in the Jaco arm environment).
4. With these improvements, we show that advantage-weighted autoregressive FB (FB-AWARE) extends FB performance to new environments such as Humanoid and the locomotion environments in the canonical *D4RL benchmark* (Fu et al., 2020). On the latter, FB-AWARE matches the performance of standard offline RL agents trained on a single task with full access to rewards (Section 4.3.3), further vindicating the use of behavior foundation models for zero-shot RL.

2 Preliminaries

Markov decision process, notation. We consider a reward-free Markov decision process (MDP) $\mathcal{M} = (S, A, P, \gamma)$ with state space S, action space A, transition probabilities $P(s'|s,a)$ from state s to s' given action a, and discount factor $0 < \gamma < 1$ (Sutton & Barto, 2018). A policy π is a function $\pi\colon S \to \mathrm{Prob}(A)$ mapping a state s to the probabilities of actions in A. Given $(s_0, a_0) \in S \times A$ and a policy π, we denote $\Pr(\cdot|s_0, a_0, \pi)$ and $\mathbb{E}[\cdot|s_0, a_0, \pi]$ the probabilities and expectations under state-action sequences $(s_t, a_t)_{t\geq 0}$ starting at (s_0, a_0) and following policy π in the environment, defined by sampling $s_t \sim P(s_t|s_{t-1}, a_{t-1})$ and $a_t \sim \pi(a_t|s_t)$. Given any reward function $r\colon S \to \mathbb{R}$, the Q-function of π for r is $Q^\pi_r(s_0, a_0) := \sum_{t\geq 0} \gamma^t \mathbb{E}[r(s_t)|s_0, a_0, \pi]$. The *value function* of π for r is $V^\pi_r(s) := \mathbb{E}_{a\sim\pi(s)} Q^\pi_r(s, a)$, and the *advantage function* is is $A^\pi_r(s, a) := Q^\pi_r(s, a) - V^\pi_r(s)$.

We assume access to a dataset consisting of *reward-free* observed transitions (s_t, a_t, s_{t+1}) in the environment. We denote by ρ the distribution of states s_{t+1} in the training set.

Behavioral foundation models, zero-shot RL. A *behavioral foundation model* for a given reward-free MDP, is an agent that can produce approximately optimal policies for any reward function r specified at test time in the environment, without performing additional learning or fine-tuning for each new reward function. An early example of such a model includes universal successor features (SFs) (Borsa et al., 2018), which depend on a set of (sometimes handcrafted) features: at test time, the reward is linearly projected onto the features, and a pre-trained policy is applied. Forward-backward representations (defined below) are another one, mathematically related to SFs. Touati et al. (2023) compares a number of variants of SFs and FB on a number of empirical problems.

The forward-backward framework. The FB framework (Touati & Ollivier, 2021; Touati et al., 2023) is a theoretically and empirically well-supported way to train BFMs, based on learning an efficient representation of the *successor measures* M^π for various policies π. For each state-action $(s_0, a_0) \in S \times A$, this is a measure over states, describing the distribution of future states visited by starting at (s_0, a_0) and following policy π. It is defined as

$$M^\pi(s_0, a_0, X) := \sum_{t \geq 1} \gamma^t \Pr(s_t \in X \mid s_0, a_0, \pi) \tag{1}$$

for any subset $X \subset S$. M^π satisfies a measure-valued Bellman equation (Blier et al., 2021), which can be used to learn approximate parametric models of M.

Touati & Ollivier (2021) propose to learn a finite-rank parametric model of M, as follows:

$$M^{\pi_z}(s_0, a_0, X) \approx \int_{s \in X} F(s_0, a_0, z)^\top B(s) \rho(\mathrm{d}s) \tag{2}$$

where ρ is the data distribution, where F and B take values in $\mathbb{R}^d$, where $z \in \mathbb{R}^d$ is a task encoding vector, and where

$$\pi_z(s) = \arg\max_a F(s, a, z)^\top z \tag{3}$$

is a parametric policy depending on z. F, B and π_z are learned at train time. At test time, given a reward function r, one estimates the task representation vector

$$z = \mathbb{E}_{s \sim \rho}[r(s) B(s)] \tag{4}$$

and then the policy π_z is applied.

The main result of Touati & Ollivier (2021) is that when (2)–(3) hold, then for *any* reward function r, the policy π_z so obtained is optimal. At test time, reward functions for FB may also be specified through an expert demonstration (Pirotta et al., 2023). The full algorithm for FB training is provided in Algo. 1 (Appendix B.3).

3 Breaking Some Key Limitations of the Forward-Backward Framework

3.1 Auto-Regressive Features for Non-Linear Task Encoding

Intuition for auto-regressive FB: nonlinear task encoding. Forward-backward (FB) representations and their predecessor, universal successor features (SFs), attempt to solve zero-shot RL by linearly projecting new tasks (reward functions r) onto a set of features $B \colon S \to \mathbb{R}^d$. At test time, when facing a new reward function r, a task encoding $z \in \mathbb{R}^d$ is computed by $z = \mathbb{E}[r(s)B(s)]$ (FB) or $z = (\mathbb{E}[\phi(s)\phi(s)^\top)^{-1}\mathbb{E}[r(s)\phi(s)]$ (SFs). Then a pretrained policy π_z is applied.

FB aims at learning the features B that minimize the error from this process: B is obtained by a finite-rank approximation of the operator that sends a reward r to its Q-function. Bringing the FB loss to 0 (which requires infinitely many features) guarantees successful zero-shot RL for any reward function r. Theoretically, the features B in FB "most linearize" the computation of Q-functions, and empirically this brings better performance than other feature choices (Touati et al., 2023).

Still, even with the best features B, the task encoding z is *linear*, because $z = \mathbb{E}[r(s)B(s)]$ is linear in r: tasks are identified by the size-d vector of their correlations with a fixed set of d pre-trained features B.

The standard FB framework learns a rank d approximation by focusing on the main eigenvectors of the environment dynamics (Blier et al., 2021). Projecting the reward onto these eigenvectors can remove spatial precision, creating short-term reward blurring (Touati & Ollivier, 2021).

This is clearly suboptimal. Intuitively, if we first acquire information that the rewards are located in the top-left corner of S, we would like to use more precise features located in the top-left corner to better identify the reward function there.

Auto-regressive features make this possible, while still keeping most of the theoretical properties of plain FB. The idea is to compute the task encoding $z = \mathbb{E}[r(s)B(s)]$ progressively, and let the later-computed features B depend on the early components of z. We decompose z and B into K blocks $z = (z_1, z_2, \dots, z_K)$ and $B = (B_1, B_2, \dots, B_K)$. We first compute $z_1 = \mathbb{E}[r(s)B_1(s)]$ as in plain FB. But then we compute $z_2 = \mathbb{E}[r(s)B_2(s, z_1)]$ where the second block of features B_2 is allowed to depend on z_1, thus conditioning the features on the task information provided by z_1. This can be iterated: $z_3 = \mathbb{E}[r(s)B_3(s, z_1, z_2)]$, etc. The resulting vector $= (z_1, z_2, \dots, z_K)$ encodes the task in an auto-regressive manner, where the meaning of z_i depends on $z_{1:i-1}$.

Intuitively, z_1 provides a "coarse" task encoding by linear features. Then we compute a further, finer task encoding z_2 by computing the correlation of r with features B_2 that depend on the coarse task encoding z_1. Hopefully the features B_2 can become more specialized and provide a better task encoding.

In practice, the main change with respect to plain FB training is that B depends on z. We now represent the successor measures M^{π_z} by $F(z)^\top B(z)$, instead of simply $F(z)^\top B$ which shares the same B for all policies. This allows for a better fit of the FB model. This is formalized in the next section and in Appendix B.

Contrary to plain FB, the task encoding $r \mapsto z$ becomes fully nonlinear: the set of tasks r represented exactly becomes a nonlinear submanifold of all possible tasks, instead of a d-dimensional subspace. Even with just two levels of features, this model is able to represent an arbitrary nonlinear mapping between reward functions r and task representations z (Appendix, Theorem B.3), instead of just linearly projecting the reward onto a fixed basis of features. This greatly extends the theoretical expressivity of the FB and successor feature frameworks.

This model also encodes a hierarchical prior on tasks, favoring tasks that can be described through a cascade of more and more specialized task features.

FB with auto-regressive task encoding: formal description. Auto-regressive features extend plain FB by letting B depend on z. In ordinary FB, this would be problematic, since the task encoding $z = \mathbb{E}[r(s)B(s)]$ used at test time becomes a fixed point equation if B depends on z. However, this fixed point equation can be handled easily if B has a hierarchical or auto-regressive structure.

Definition 3.1. A feature map $B\colon S \times \mathbb{R}^d \to \mathbb{R}^d$ is called *auto-regressive* if, for any $1 \le i \le d$ and any $(s, z) \in S \times \mathbb{R}^d$, the i-th component of $B(s, z)$ only depends on $(z_1, \dots, z_{i-1})$ and not on $(z_i, \dots, z_d)$.

For such models, we can easily compute fixed-points values of the type $z = B(s, z)$, by first computing the component B_1 of the output (which does not dependent on z), which determines z_1, which allows us to compute the component B_2 of the output, which determines z_2, etc.

In practice, auto-regressive models $B(s, z)$ can be built by splitting both the representation vector $z \in \mathbb{R}^d$ and the output $B(s, z) \in \mathbb{R}^d$ into k "auto-regressive groups" of dimension d/k. The first group $B_1(s, z)$ in the output of B is actually independent of z, and the i-th group $B_i(s, z)$ of the output of B only takes as inputs the previous groups $z_1, \dots, z_{i-1}$ of z. At test time, this allows us to compute the fixed point $z = \mathbb{E}_{s\sim\rho}[r(s)B(s, z)]$ by first estimating the first group, $z_1 = \mathbb{E}_{s\sim\rho}[r(s)B_1(s)]$ similar

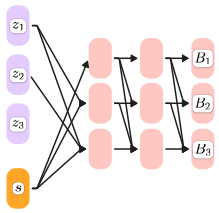

Figure 1: An auto-regressive architecture for $B(s, z)$. The i-th block of the output B only depends on blocks $z_1, \dots, z_{i-1}$ of the input z. In each layer, the weights from each block to the lower-ranking blocks of the next layer have been removed. The state s is still fed to every block on the input layer.

to plain FB. Then the other groups are computed iteratively: $z_{i+1} = \mathbb{E}_{s\sim\rho}[r(s)B_{i+1}(s, z_1, \dots, z_i)]$. In the experiments, we focus on $k = 4$ or $k = 8$ auto-regressive groups.

We employ a network architecture (Fig. 1) in which each layer of B_i has access to the previous layers of all previous blocks $B_{1\dots i}$: this ensures good expressivity while preserving the auto-regressive property. This allows for efficient evaluation: this is implemented as masks on fully-connected layers for the full model B.

The following result extends the theorem from Touati & Ollivier (2021) for vanilla FB, to allow B to depend on z.

Theorem 3.2. *Assume we have learned representations $F\colon S \times A \times \mathbb{R}^d \to \mathbb{R}^d$ and $B\colon S \times \mathbb{R}^d \to \mathbb{R}^d$, as well as a parametric family of policies π_z depending on $z \in \mathbb{R}^d$, satisfying $\forall s \in S,\ a \in A, X \subset S, z \in \mathbb{R}^d$*

$$\begin{cases} M^{\pi_z}(s, a, X) = \int_X F(s, a, z)^\top B(s', z)\, \rho(\mathrm{d}s'), \\ \pi_z(s) = \arg\max_a F(s, a, z)^\top z \end{cases} \tag{5}$$

Then the following holds. For any reward function r, if we can find a value $z_r \in \mathbb{R}^d$ such that

$$z_r = \mathbb{E}_{s\sim\rho}[r(s)B(s, z_r)] \tag{6}$$

then π_{z_r} is an optimal policy for reward r, and the optimal Q-function is $Q^\star_r(s, a) = F(s, a, z_r)^\top z_r$.

Moreover, if B is auto-regressive, then the fixed point (6) always exists, and can be computed directly, by iteratively computing each component $z_i = \mathbb{E}[r(s)B_i(s, z_1, \dots, z_{i-1})]$ for $i = 1, \dots, d$.

Further theoretical properties and proofs are given in Appendix B. In particular, Theorem B.3 establishes that autoregressive FB with two blocks is enough to represent *any* task encoding $r \mapsto z_r$, while vanilla FB is contrained to a linear task encoding $r \mapsto z_r$. Thus, autoregressive FB is inherently more expressive.

Training F and B for this setup is similar to Touati & Ollivier (2021), except that B depends on z, which has some consequences for minibatch sampling, and results in higher variance. The details are given in Appendix B.2 and Algorithm 2. Training is based on the measure-valued Bellman equation satisfied by M^{π_z}: we plug in the model $M^{\pi_z} \approx F(z)^\top B(z)\rho$ in this equation and minimize the Bellman gaps.

There is little computational overhead compared to vanilla FB. In practice, we enforce the auto-regressive property via a single neural network B, by dropping a specific subset of the layer connections across neurons (Fig. 1). This implementation allows for efficient training: given access to any specific z and s, the output $B(s, z)$ can be computed in a single forward pass. On the other hand, the computation of the fixed point z_r from (6) requires several forward passes through the network B in Fig. 1, but this occurs only at test time when the reward function is known.

Auto-regressive FB models the successor features M^{π_z} via a model $M^{\pi_z} \approx F(z)^\top B(z)\rho$ with full dependency on z, versus $M^{\pi_z} \approx F(z)^\top B\rho$ for vanilla FB. This is more natural, especially for large

γ. Indeed, for $\gamma \to 1$ we have $M^{\pi_z}(s_0, a_0, \mathrm{d}s) = \frac{1}{1-\gamma}\mu_z(\mathrm{d}s) + o(1/(1-\gamma))$ with μ_z the stationary distribution of π_z, namely, approximately rank-one with z dependency on the s part. The vanilla FB model has no z dependency on the s part, only on the (s_0, a_0) part, which means all stationary distributions μ_z must be approximated using the shared features $B(s)$.

3.2 Better Offline Optimization for FB: Advantage Weighting and Other Improvements

Like off-policy algorithms designed for the offline RL setting, vanilla FB training appears prone to distribution shift, hindering its ability to scale and to learn on datasets exhibiting mixtures of behaviors for various tasks. Inspired by the recent analysis (Cetin et al., 2024), we propose a set of modifications to FB training to overcome these limitations.

Improved advantage weighting objective We introduce an alternative policy optimization step for FB, based on recent analysis and advancements in offline RL algorithms with policy constraints. We use an improved version of the *advantage-weighted* (AW) regression loss (Peng et al., 2019), commonly used in popular recent algorithms (Nair et al., 2020a; Kostrikov et al., 2022; Wang et al., 2020; Garg et al., 2023). Following Nair et al. (2020a), a first version starts with sampling a batch of n transitions from the data, and updates the parametric policy π_θ to optimize

$$\arg\max_\theta \mathbb{E}_{(a_{1:n}, s_{1:n})\sim\mathcal{B}}\left[\sum_{i=1}^{n} w(s_i, a_i)\log \pi_\theta(a_i|s_i)\right], \tag{7}$$
$$\text{where} \quad w(s_i, a_i) = \frac{\exp(A_\phi(s_i, a_i)/\beta)}{\sum_{j=0}^{n}\exp(A_\phi(s_j, a_j)/\beta)}, \quad A_\phi(s, a) = Q_\phi(s, a) - \mathbb{E}_{a'\sim\pi}[Q_\phi(s, a')]$$

is the advantage function as estimated by the critic model Q_ϕ. The weights $w(s, a)$ are a weighted importance sampling (WIS) approximation of $\exp(A_\phi(s, a)/\beta)/Z$, where $Z = \mathbb{E}_{s,a\sim\mathcal{B}}[\exp(A_\phi(s, a)/\beta)]$.

In the FB framework, the policies are conditioned by the latent variable z, and the Q-function estimate is $Q_\phi(s, a, z) = F_\phi(s, a, z)^T z$. Therefore, a direct transposition of (7) to the FB framework leads to the following objective for training the policy:

$$\arg\max_\theta \mathbb{E}_{\substack{(a_{1:n}, s_{1:n})\sim\mathcal{B},\\ z_{1:n}\sim Z}}\left[\sum_{i=1}^{n} w(s_i, a_i.z_i)\log \pi_\theta(a_i|s_i, z_i)\right], \tag{8}$$
$$w(s_i, a_i, z_i) = \frac{\exp(A_\phi(s_i, a_i, z_i)/\beta)}{\sum_{j=0}^{n}\exp(A_\phi(s_j, a_j, z_j)/\beta)}, A_\phi(s, a, z) = F_\phi(s, a, z)^T z - \mathbb{E}_{a'\sim\pi_z}[F_\phi(s, a', z)^T z],$$

However, the variance and bias of this weighted importance sampling approach have a linear inverse relationship with the batch size n. Instead, we propose to use modified weights $w'(s, a, z)$ that implement *improved weighted importance sampling* (IWIS), a simple change to WIS proposed by Skare et al. (2003), shown to reduce the bias of WIS from $O(n^{-1})$ to $O(n^{-2})$. Integrating IWIS yields our final *policy improvement objective*:

$$\arg\max_\theta \mathbb{E}_{\substack{(a_{1:n}, s_{1:n})\sim\mathcal{B},\\ z_{1:n}\sim Z}}\left[\sum_{i=1}^{n} w'(s_i, a_i.z_i)\log \pi_\theta(a_i|s_i, z_i)\right], \tag{9}$$
$$\text{where} \quad w'(s_i, a_i, z_i) \propto \frac{w(s_i, a_i, z_i)}{\sum_{j\neq i} w(s_j, a_j, z_j)}, \sum_{j=0}^{n} w'(s_j, a_j, z_j) = 1,$$

and $w(s_i, a_i, z_i)$ is as in (8).

We validate the effect of IWIS in Table 10 (Appendix E): FB-AW, which uses the IWIS weights (9), performs slightly but consistently better than with the WIS weights (8).

Evaluation-based sampling. Furthermore, following Cetin et al. (2024), we integrate *evaluation-based sampling* (ES), an additional component to mitigate the undesirable consequences of learning

a Gaussian policy, which is generally insufficient to capture the distribution from the exponentiated advantages. Namely, when deploying π_θ at test time, we approximate the argmax in (3) by sampling M actions $a_1, \ldots, a_M$ from the trained policy $\pi(s)$, and perform the one with the largest Q-value as predicted by $F_\phi(s, a_i, z)^T z$. The specific impact of this change is illustrated in Fig. 7 (Appendix E).

Uncertainty representation. To represent uncertainty in the model, we train two different networks F_1 and F_2 for the forward embedding, inspired by (Fujimoto et al., 2018; Touati et al., 2023). However, we introduce two changes.

In the Bellman equation, we use the *average* of the resulting two estimates of the target successor measures, namely, $\frac{1}{2}(F_1(s_{t+1}, a_{t+1}, z)^\top B(s') + F_2(s_{t+1}, a_{t+1}, z)^\top B(s'))$. This departs from Touati et al. (2023), which used the min between the target successor measures, namely, $\min\{F_1(s_{t+1}, a_{t+1}, z)^\top B(s'), F_2(s_{t+1}, a_{t+1}, z)^\top B(s')\}$, in line with (Fujimoto et al., 2018). Indeed, for Q-function estimates, a min might encode some form of conservatism, but for successor measures, the interpretation of a min is less direct. [1]

Finally, we use two fully parallel networks for F_1 and F_2, while Touati et al. (2023) opted for a shared processing network with two separate shallow heads for F_1 and F_2. The specific impact of these changes is reported in Table 10 (Appendix E).

4 Evaluation

4.1 Algorithms and Baselines

We mainly compare the following algorithms:

1. "Vanilla" FB: the classical implementation of FB from Touati et al. (2023) that employs TD3 policy improvement loss.
2. FB-AW (FB with advantage weighting): The FB method using the advantage weighting components described in Section 3.2.
3. FB-AWARE (FB with advantage weighting and auto-regressive encoding): The FB method using both AW and the auto-regressive component from Section 3.1. For the auto-regressive part, we test either 4 or 8 consecutive auto-regressive blocks for B.
4. On some environments (those where AW is not necessary to reach good performance) we also include FB-ARE without the AW component.

In addition to the vanilla FB baseline, we include universal successor features (Borsa et al., 2018) based on *Laplacian eigenfunctions* as the base features (Touati et al., 2023). This version of successor features was found to perform best in Touati et al. (2023). We denote it by LAP-AW, since we use the advantage weighting as in FB-AW.

All the aforementioned variants of FB use the same architecture and consistent hyperparameters.

4.2 Datasets and Benchmarks

We consider a series of environments, tasks, and datasets for these environments, as follows.

We start with the *Jaco arm* domain (Laskin et al., 2021), a simple robotic arm model. The tasks consist in reaching various target positions (Section 4.3.1). This provides a test of spatial precision. For this domain, we build a training dataset via the *RND* unsupervised exploration method from Yarats et al. (2022), which provides good data diversity if exploration is not too difficult in an environment.

Next, we consider four standard domains from the *DeepMind Control (DMC) Suite* (Tassa et al., 2018): Walker, Cheetah, Quadruped, and Humanoid. Since we want to build behavior foundation models and not task-specific agents, on top of the classical tasks for these environments (walk, run...) we introduce a number of additional tasks such as bounce, flip, pullup..., described in Appendix C.3. For these domains, we consider two training datasets:

[1] For instance, since the Q-function for reward r is $Q = M.r$, taking the min of M might encode a min for a reward r but a max for the reward $-r$.

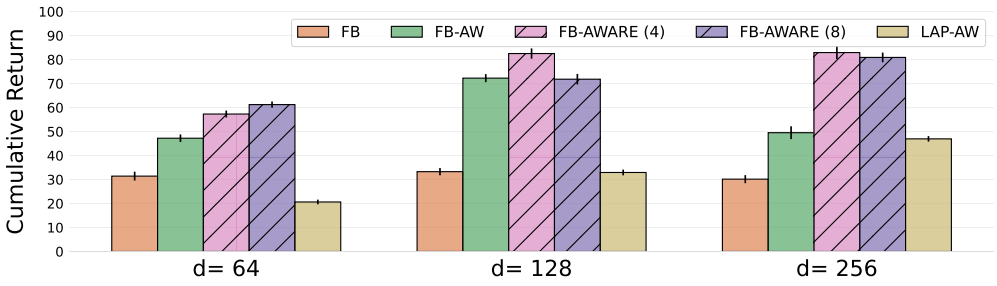

Figure 2: Average cumulative reward achieved by the algorithms, trained on RND dataset for different representation dimensions when aiming to reach goals (four randomly selected goals and four corner goals), in the Jaco arm environment.

1. We build a first dataset using RND, as for Jaco. However, RND appears to provide insufficient exploration (particularly on Humanoid); moreover, the RND trainset does not contain any purposeful trajectories.
2. Therefore, we also train on the *MOOD* datasets from Cetin et al. (2024). MOOD contains a mixture of behaviors, obtained as follows. For each environment, a small number of "classical" tasks are selected. Then an online TD3 algorithm is used to train a classical agent for each of these tasks. The set of trajectories produced by these agents during training are then pooled and merged into a single dataset for the environment. Thus, the mixed-objective MOOD datasets include high-quality examples for a few tasks in each environment.
   Evaluation on the MOOD dataset must distinguish between tasks that contributed to the dataset (*in-dataset* tasks), and tasks that did not (*out-of-dataset* tasks). [2] A priori, one would expect the former to be easier, as information from the original single-task agents is present in the data. To evaluate the ability of the FB models to generalize beyond in-dataset tasks, we used the new tasks from Appendix C.3 as out-of-dataset tasks.

Finally, to test the generality of the approach, we also train FB, FB-AW and FB-AWARE agents on the locomotion tasks of the *D4RL benchmark* (Fu et al., 2020). Here we stick to the original tasks in the benchmark, and compare FB performance to the best task-specific offline RL agents in the literature. Since those are single-task while FB is a generalist agent, this is a natural *topline* for FB, so we expect FB to reach a good fraction of the performance of the best task-specific agents, in line with the methodology of Touati et al. (2023).

4.3 Empirical Evaluation

We train FB, FB-AW, FB-AWARE (4 and 8 blocks) and LAP-AW on the four DMC locomotion environments (Walker, Cheetah, Quadruped and Humanoid), as well as the Jaco arm domain. We pretrain each model on both offline datasets (MOOD and RND), and repeat each training 5 times (with different random seeds).

We evaluate each model on several downstream tasks per environment. For each model and task, we sample 100,000 states $\{s\}$ from the offline dataset and compute their corresponding task reward $\{r(s)\}$ in order to infer the task encoding vector z_r ((4) or (6)). Then we compute the cumulated reward achieved by the policy π_{z_r}, computed using the task-specific reward, and averaged over 100 episodes. Finally, we report the average and variance of the cumulative reward over the 5 pre-trained models (with different seeds).

4.3.1 Jaco Arm Results

We train each algorithm in the Jaco environment using the RND dataset, for three choices of representation dimension: $d = 64, 128,$ and 256. The tasks involve reaching a goal within an episode

[2] We avoid "in-distribution" and "out-of-distribution", since FB is not trained on a distribution of tasks but in an unsupervised way given the data.

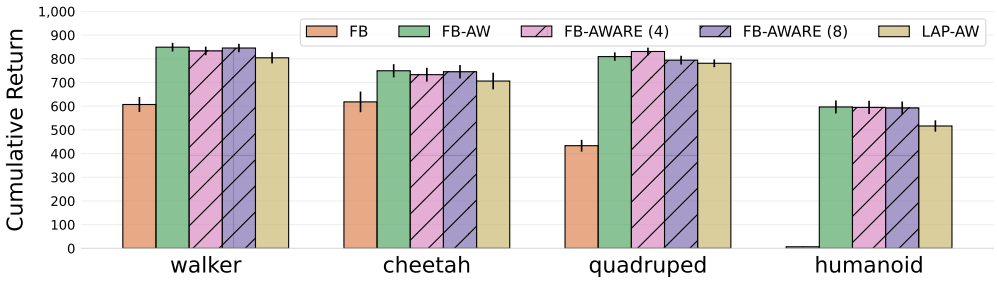

Figure 3: Averaged cumulative reward achieved by the algorithms on *in-dataset tasks*, trained on MOOD dataset for DMC Locomotion.

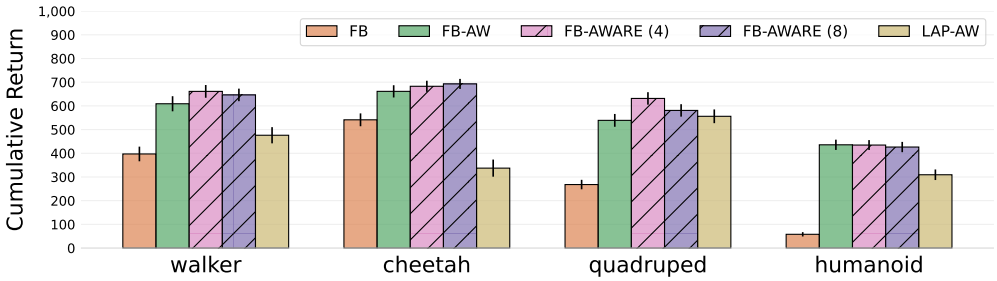

Figure 4: Average cumulative reward achieved by the algorithms on *out-of-dataset* tasks, trained on MOOD dataset for DMC Locomotion.

length of 250 time steps, with the agent receiving a reward of approximately 1 when the arm's gripper is close to the target goal specified by its (x, y, z) coordinates. For the goals, we included the four corners of the environment, plus four goals selected at random (once and for all, common to all the algorithms tested).

In Figure 2, we depict the average rewards attained by each algorithm for reaching this mixture of goals. The resulting goal-reaching rewards for dimensions 64, 128, and 256 are presented in Tables 5, 6, and 7, respectively.

FB-AW significantly outperforms FB, more than doubling the score for the best dimension $d = 128$. FB-AWARE with 4 autoregressive blocks further enhances performance by a relative margin of about 15%.

4.3.2 DMC Locomotion Results

For Cheetah, Quadruped, Walker and Humanoid, the MOOD dataset results in substantially better models than the RND dataset, whatever the algorithm (Appendix D, Table 9 vs Table 8). This is especially striking for Humanoid, where RND does not explore enough and even a classical single-task TD3 agent is hard to train. Therefore, we focus the discussion on MOOD, with full RND results in Appendix D.

Figures 3 and 4 report the results for the Cheetah, Quadruped, Walker and Humanoid environments, using MOOD data for training, evaluated on in-dataset tasks and out-of-dataset tasks respectively. We used a fixed representation dimension for all algorithms ($d = 64$ for Walker, Cheetah and Quadruped, $d = 128$ for Humanoid).

In this setup, the advantage weighting component proves crucial for achieving satisfactory performance, particularly in the Humanoid environment, where vanilla FB performance is near-zero. However, on the lower-quality RND dataset, the advantage weighting component appears to hurt performance (Appendix D, Table 8). This is consistent with observations in Yarats et al. (2022) for the single-task setting, where conservative offline-RL methods hurt performance on RND data.

Table 1: Performance on the popular locomotion-v2 and FrankaKitchen datasets from the D4RL benchmark, comparing with recent offline RL algorithms (with performance as reported in the literature). FB-AWARE uses 8 AR groups. Other hyper-parameters are tuned per-environment, consistently with the offline baselines.

Dataset/Algorithm	BC	10%BC	DT	1StepRL	AWAC	TD3+BC	CQL	IQL	XQL	FB	FB-AW	FB-AWARE
	Reward-based learning									Reward free learning		
halfcheetah-medium-v2	42.6	42.5	42.6	48.4	43.5	48.3	44.0	47.4	48.3	49.0±1.93	60.0±0.9	62.7±0.9
hopper-medium-v2	52.9	56.9	67.6	59.6	57.0	59.3	58.5	66.3	74.2	0.9±0.69	59.1±5.0	59.9±21.4
walker2d-medium-v2	75.3	75.0	74.0	81.8	72.4	83.7	72.5	78.3	84.2	0.5±0.9	80.5±11.7	89.6±0.8
halfcheetah-medium-replay-v2	36.6	40.6	36.6	38.1	40.5	44.6	45.5	44.2	45.2	30.8±23.4	52.7±1.2	50.8±1.2
hopper-medium-replay-v2	18.1	75.9	82.7	97.5	37.2	60.9	95.0	94.7	100.7	16.4±2.9	87.1±3.7	89.6±4.0
walker2d-medium-replay-v2	26.0	62.5	66.6	49.5	27.0	81.8	77.2	73.9	82.2	9.9±4.9	91.7±6.3	98.8±0.5
halfcheetah-medium-expert-v2	55.2	92.9	86.8	93.4	42.8	90.7	91.6	86.7	94.2	91.7±6.7	99.6±0.8	100.1±0.7
hopper-medium-expert-v2	52.5	110.9	107.6	103.3	55.8	98.0	105.4	91.5	111.2	1.8±1.2	55.9±11.6	62.2±9.2
walker2d-medium-expert-v2	107.5	109.0	108.1	113.0	74.5	110.1	108.8	109.6	112.7	0.3±0.8	109.6±1.3	105.8±0.6
Locomotion-v2 total	466.7	666.2	672.6	684.6	450.7	677.4	698.5	692.6	752.9	22.3	696.2	719.5
kitchen-partial-v0	38.0	-	-	-	-	-	49.8	46.3	73.7	4±4	47.0±4.5	52.5±9.4
kitchen-mixed-v0	51.5	-	-	-	-	-	51.0	51.0	62.5	5±5	48.5±7.2	53.5±3.8

LAP-AW does well at in-dataset tasks, but lags behind FB for out-of-dataset tasks. This may be because LAP-AW's learned features are closely tied to the in-dataset tasks, derived from the eigenfunctions of the Laplacian of the behavior policy present in the dataset.

FB-AW and FB-AWARE exhibit the most favorable and consistent overall performance. The autoregressive component provides a slight enhancement in out-of-dataset tasks across all environments except from Humanoid. This slight difference is not observed on the RND dataset (Table 8). The best-performing model overall is obtained with the MOOD dataset and FB-AWARE algorithm.

4.3.3 D4RL Performance

Finally, to test the generality and robustness of these methods, we test performance on the D4RL benchmark after reward-free training. D4RL is a ubiquitous benchmark, used by many recent offline RL research for evaluation and comparison.

Here, we are comparing the multitask, unsupervised FB-AWARE agent to task-specifics agents, so the performance of the latter are a natural *topline* for FB-AW and FB-AWARE. In line with Touati et al. (2023), we expect FB-AWARE to reach a good fraction of the performance of the agents trained specifically on each task.

So we compare FB-AWARE's performance with the results available for a large pool of offline RL algorithms optimizing for an individual objective with full access to rewards. This setting is quite different from Section 4.3.2 and especially the MOOD datasets, since D4RL datasets mostly comprise trajectories from agents trying to accomplish a single task.

FB-AW and FB-AWARE's overall performance matches the task-specific recent state-of-the-art from XQL and IQL. There is a slight advantage to FB-AWARE over FB-AW, although this falls within the overall margin of error.

This establishes that zero-shot, task-agnostic behavior foundation models trained via FB can compete with top task-specific agents for offline RL on standard benchmarks.

4.4 Ablations and Discussion

Appendix E contains additional tests and ablations concerning each of the components introduced, such as the impact of dimension d, the number of auto-regressive blocks for FB-AWARE, specific design choices for FB training (B normalization, z sampling), and the offline RL methods introduced in Section 3.2 (advantage weighting, improved weighted importance sampling, evaluation-based sampling, and uncertainty representation).

The impact of training data for learning behavior foundation models. Perhaps unsurprisingly, the dataset has a large impact on the performance of behavior foundation models, as exemplified by the higher scores of FB methods trained on MOOD vs RND for the Locomotion tasks. On Humanoid, the combination of both MOOD and advantage weighting appears necessary to reach any reasonable performance at all.

On the other hand, the best algorithm to train a behavior foundation model also depends on the data available: with only RND data, advantage weighting actually hurts performance (Table 8), although with these data, performance is relatively poor anyway. This is consistent with existing observations for classical single-task agents: on RND data, TD3 is better than conservative offline-RL approaches (Yarats et al., 2022).

Limitations. The environments considered here are all noise-free, Markovian (history-free) continuous control environments.

The effect of autoregressive FB is relatively modest in these experiments. This is surprising given the huge theoretical change in expressivity compared to vanilla FB. This suggests that the main limiting factor in our suite of experiments may not be the expressivity of the behavior foundation model, perhaps due to limited exploration in the training datasets, or from the relative simplicity of the environments tested.

5 Conclusions

Specific offline RL training techniques are necessary to build efficient FB behavior foundation models in environments such as DMC Humanoid, and can make the difference between near-zero and good performance. Employing auto-regressive features greatly enhances the theoretical expressivity of these foundation models, and improves spatial precision and task generalization. The improvement is moderate in our setup, perhaps indicating that BFM expressivity is not a key limiting factor for these tasks. These improvements bring zero-shot, reward-free FB BFMs on par with single-task, reward-trained offline agents on a number of locomotion environments.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

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.

Léonard Blier, Corentin Tallec, and Yann Ollivier. Learning successor states and goal-dependent values: A mathematical viewpoint. *arXiv preprint arXiv:2101.07123*, 2021.

Diana Borsa, Andre Barreto, John Quan, Daniel J Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2018.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Edoardo Cetin, Andrea Tirinzoni, Matteo Pirotta, Alessandro Lazaric, Yann Ollivier, and Ahmed Touati. Asac: Simple ingredients for offline reinforcement learning with diverse data. *ICML*, 2024.

Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pp. 1430–1440. PMLR, 2021.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural computation*, 5(4):613–624, 1993.

Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=tc5qisoB-C.

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Russ R Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 35:35603–35620, 2022.

Kevin Frans, Seohong Park, Pieter Abbeel, and Sergey Levine. Unsupervised zero-shot reinforcement learning via functional reward encodings. *arXiv preprint arXiv:2402.17135*, 2024.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, pp. 1582–1591, 2018. URL http://proceedings.mlr.press/v80/fujimoto18a.html.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Divyansh Garg, Joey Hejna, Matthieu Geist, and Stefano Ermon. Extreme q-learning: Maxent RL without entropy. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SJ0Lde3tRL.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

Christopher Grimm, Irina Higgins, Andre Barreto, Denis Teplyashin, Markus Wulfmeier, Tim Hertweck, Raia Hadsell, and Satinder Singh. Disentangled cumulants help successor representations transfer to new tasks. *arXiv preprint arXiv:1911.10866*, 2019.

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.

Philippe Hansen-Estruch, Amy Zhang, Ashvin Nair, Patrick Yin, and Sergey Levine. Bisimulation makes analogies in goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, pp. 8407–8426. PMLR, 2022.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. *arXiv preprint arXiv:2110.15191*, 2021.

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.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 5502–5511. International Joint Conferences on Artificial Intelligence Organization, 7 2022. DOI: 10.24963/ijcai.2022/770. URL https://doi.org/10.24963/ijcai.2022/770. Survey Track.

Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020a.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020b.

Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation policies with Hilbert representations. In *Forty-first International Conference on Machine Learning*, 2024.

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

Matteo Pirotta, Andrea Tirinzoni, Ahmed Touati, Alessandro Lazaric, and Yann Ollivier. Fast imitation via behavior foundation models. In *The Twelfth International Conference on Learning Representations*, 2023.

Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Rapid exploration for open-world navigation with latent goal models. *arXiv preprint arXiv:2104.05859*, 2021.

Øivind Skare, Erik Bølviken, and Lars Holden. Improved sampling-importance resampling and reduced bias importance sampling. *Scandinavian Journal of Statistics*, 30(4):719–737, 2003.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 2nd edition.

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

Ahmed Touati and Yann Ollivier. Learning one representation to optimize all rewards. *Advances in Neural Information Processing Systems*, 34:13–23, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does zero-shot reinforcement learning exist? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MYEap_OcQI.

Ziyu Wang, Alexander Novikov, Konrad Zolna, Josh S Merel, Jost Tobias Springenberg, Scott E Reed, Bobak Shahriari, Noah Siegel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7768–7778. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/588cb956d6bbe67078f29f8de420a13d-Paper.pdf.

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

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

Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2371–2378. IEEE, 2017.

Chongyi Zheng, Benjamin Eysenbach, Homer Walke, Patrick Yin, Kuan Fang, Ruslan Salakhutdinov, and Sergey Levine. Stabilizing contrastive rl: Techniques for offline goal reaching. *arXiv preprint arXiv:2306.03346*, 2023a.

Chongyi Zheng, Ruslan Salakhutdinov, and Benjamin Eysenbach. Contrastive difference predictive coding. *arXiv preprint arXiv:2310.20141*, 2023b.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Further Related Work

The forward-backward framework (Touati & Ollivier, 2021) builds upon the principles of successor features (Barreto et al., 2017; Zhang et al., 2017; Grimm et al., 2019), the continuous extension to the canonical successor representation Dayan (1993) and its continuous state-space extension. However, in contrast to FB, this line of work has mostly focused on constructing a set of features, linear w.r.t. downstream rewards, using apriori knowledge and heuristic measures such as Laplacian eigenfunctions (Borsa et al., 2018). To this end Touati et al. (2023) showed, empirically, the superiority of end-to-end learning with FB as compared to many such heuristics, in line with results in this same work. The proposed autoregressive extension to FB could be also applied to this broader class of methods, a noteworthy direction for future work.

Goal-conditioned RL (GCRL) (Liu et al., 2022) is another area of research closely related to FB, which has seen notable successful applications in real-world robotics (Shah et al., 2021; Ma et al., 2022; Zheng et al., 2023a). As with successor features, GCRL has traditionally relied on a priori knowledge taking the form of explicit demonstrations (Ding et al., 2019; Gupta et al., 2019), handcrafted subgoals (Andrychowicz et al., 2017; Nachum et al., 2018; Chane-Sane et al., 2021), together with other coverage heuristics (Ghosh et al., 2019; Hansen-Estruch et al., 2022) – which have been used both to construct the space of goals and learn its relative multi-task policy. Moreover, in a similar fashion to FB, recent work also strived to model and tackle the GCRL problem with a principled contrastive-like end-to-end objective (Eysenbach et al., 2021; 2022; Zheng et al., 2023b). However, GCRL is, by design, more restrictive than successor features and FB as it cannot capture tasks that go beyond reaching individual points in the space of goals.

Other works avoid the linearity constraint of successor features and vanilla FB by explicitly relying on a prior over tasks. For instance, in an approach akin to meta-RL but without hand-crafted tasks, Frans et al. (2024) use a mixture of random MLPs, random linear functions, and random goal-reaching to pre-train a set of policies together with a encoder that quickly identifies a reward function from a few reward samples. Contrary to FB, the dynamics of the environment plays no role in building the set of features.

B Auto-Regressive FB: Extensions, Proofs, Algorithmic Considerations

B.1 Proof and Extension of Theorem 3.2

Here we prove Theorem 3.2 and extend it in two directions similar to Touati & Ollivier (2021).

The first extension concerns *goal spaces*, and is useful when we know in advance that the rewards functions of interest will not depend on the whole state. For instance, in a multi-agent setting, the reward of an agent may depend only on its own state, but it must still observe the whole state to make decisions. This is formalized by assuming that the reward function only depends on some variable $g = \phi(s)$ rather than the whole state s. Then we can learn with $B(g)$ instead of $B(s)$ (while F and policies still require the full state).

The second extension only uses the finite-rank $F^\top B$ model for the *advantage* functions, namely, the model is $M^{\pi_z}(s_0, a_0, \mathrm{d}s) \approx F(s_0, a_0, z)^\top B(s', z) + \bar{m}(s_0, z, s)$ where $\bar{m}$ is any function independent of the actions. This lifts part of the finite-rank restriction, since $\bar{m}$ itself is unconstrained: the finite-

rank FB model is only applied to the differential effect of actions on top of the baseline model $\bar{m}$.

Definition B.1 (Extended forward-backward representation of an MDP)**.** Consider an MDP with state space S and action space A. Let $\phi\colon S \times A \to G$ be a function from state-actions to some goal space $G = \mathbb{R}^k$.

Let $Z = \mathbb{R}^d$ be some representation space. Let

$$F\colon S \times A \times Z \to Z, \qquad B\colon G \times Z \to Z, \qquad \bar{m}\colon S \times Z \times G \to \mathbb{R} \tag{10}$$

be three functions. For each $z \in Z$, define the policy

$$\pi_z(a|s) := \arg\max_a F(s,a,z)^\top z. \tag{11}$$

Let ρ be any measure over the goal space G.

We say that F, B, and $\bar{m}$ are an *extended forward-backward representation* of the MDP with respect to ρ, if the following holds: for any $z \in Z$, any state-action (s,a) the successor measure M^{π_z} of policy π_z is given by

$$M^{\pi_z}(s,a,X) = \int_{g \in X} \left(F(s,a,z)^\top B(g,z) + \bar{m}(s,z,g)\right) \rho(\mathrm{d}g) \tag{12}$$

for any goal subset $X \subset G$.

Theorem B.2 (Forward-backward representation of an MDP, with features as goals)**.** *Consider an MDP with state space S and action space A. Let $\phi\colon S \times A \to G$ be a function from state-actions to some goal space $G = \mathbb{R}^k$.*

Let F, B, and $\bar{m}$ be an extended forward-backward representation of the MDP with respect to some measure ρ over G.

Then the following holds. Let $R\colon S \times A \to \mathbb{R}$ be any bounded reward function, and assume that this reward function depends only on $g = \phi(s,a)$, namely, that there exists a function $r\colon G \to \mathbb{R}$ such that $R(s,a) = r(\phi(s,a))$.

Assume that there exists $z_R \in \mathbb{R}^d$ satisfying

$$z_R = \int_{g \in G} r(g) B(g, z_R)\, \rho(\mathrm{d}g). \tag{13}$$

This is always the case if B is auto-regressive.

Then:

1. *π_{z_R} is an optimal policy for reward R in the MDP.*
2. *The optimal Q-function $Q^\star_R$ for reward R is*

$$Q^\star_R(s,a) = F(s,a,z_R)^\top z_R + \int_{g \in G} \bar{m}(s,z_R,g) r(g)\, \rho(\mathrm{d}g). \tag{14}$$

The last term does not depend on the action a, so computing the $\bar{m}$ term is not necessary to obtain the advantages $Q^*(s,a) - Q^*(s,a')$ or the optimal policies.

Theorem B.2 implies Theorem 3.2, by taking $\phi = \mathrm{Id}$ and $\bar{m} = 0$.

There is an important difference between this result and the corresponding statement in non-auto-regressive FB (Theorem 4 in Touati & Ollivier (2021)). For non-autoregressive FB, the FB model provides the Q-functions of all policies π_z for all rewards R, even if $z \neq z_R$. Namely, $Q^{\pi_z}_R(s,a) = F(s,a,z)^\top z_R$ for all pairs (z,R) (in the case $\bar{m} = 0$). Here, this only holds when $z = z_R$. Classical, non-auto-regressive FB provides more Q-functions than strictly needed to obtain the policies: it models the Q-functions of π_z even for rewards unrelated to z. With auto-regressive FB, the additional expressivity of the model comes at the price of getting less information about the other Q-functions.

Proof of Theorem B.2. Let M^π be the successor measure of policy π. Let m^π be the density of M^π with respect to ρ. Let $R(s,a) = r(\phi(s,a))$ be a reward function as in the statement of the theorem.

By Proposition 16 in Touati & Ollivier (2021), The Q-function of π for the reward R is

$$Q_R^\pi(s,a) = \int_g r(g) M^\pi(s,a,\mathrm{d}g) \tag{15}$$

Let z_R satisfy the fixed point property (13), and let us take $\pi = \pi_{z_R}$. By definition of an extended FB representation, we have

$$\begin{aligned} Q_R^{\pi_{z_R}}(s,a) &= \int_g r(g) M^{\pi_{z_R}}(s,a,\mathrm{d}g) && (16)\\ &= \int_g r(g)(F(s,a,z_R)^\top B(g,z_R) + \bar{m}(s,z_R,g))\rho(\mathrm{d}g). && (17)\\ &= F(s,a,z_R)^\top \int_g r(g) B(g,z_R)\rho(\mathrm{d}g) + \int_g r(g)\bar{m}(s,z_R,g))\rho(\mathrm{d}g). && (18) \end{aligned}$$

But thanks to the fixed point property (13), we have $\int_g r(g)B(g,z_R)\rho(\mathrm{d}g) = z_R$. Therefore, the Q-function of π_{z_R} for reward R is

$$Q_R^{\pi_{z_R}}(s,a) = F(s,a,z_R)^\top z_R + \int_g r(g)\bar{m}(s,z_R,g))\rho(\mathrm{d}g). \tag{19}$$

We have to prove that this is the optimal Q-function for R. A pair of a Q-function and policy π are optimal for R if and only if simultaneously $\pi(a|s) = \arg\max_a Q(s,a)$ and $Q = Q_R^\pi$. Here, by definition of the policies π_z, we have

$$\begin{aligned} \pi_{z_R}(a|s) &= \arg\max_a F(s,a,z_R)^\top z_R && (20)\\ &= \arg\max_a Q_R^{\pi_{z_R}}(s,a) && (21) \end{aligned}$$

since the additional term $\int_g r(g)\bar{m}(s,z_R,g))\rho(\mathrm{d}g)$ in $Q_R^{\pi_{z_R}}$ does not depend on a.

Therefore, $Q_R^{\pi_{z_r}}$ and π_{z_R} are optimal for reward R, which ends the proof. □

Theorem B.3 (Auto-regressive features with two levels are a universal approximator for task encoding)**.** *Assume the state space is finite, so that a reward function is an element of $\mathbb{R}^{\#S}$. Then, for any continuous task encoding function $\zeta\colon \mathbb{R}^{\#S} \to \mathbb{R}^d$ mapping rewards r to task encodings $z = \zeta(r)$, such that $z_r = 0$ for $r = 0$, there exist neural networks $B_1(s)$ and $B_2(s,z)$ approximating ζ, namely, for any r,*

$$\zeta(r) \approx \mathbb{E}_{s\sim\rho}[r(s)B_2(s,z_1)], \qquad z_1 = \mathbb{E}_{s\sim\rho}[r(s)B_1(s)] \tag{22}$$

up to an arbitrary precision.

Lemma B.4. *Let $f\colon \mathbb{R}^n \to \mathbb{R}^d$ be a C^3 function such that $f(0) = 0$. Then there exists a continuous matrix-valued function $g\colon \mathbb{R}^n \to \mathbb{R}^{n\times d}$ such that for any $x \in \mathbb{R}^n$,*

$$f(x) = g(x)\cdot x. \tag{23}$$

Proof of the lemma. By working on each output component of f separately, we can assume that $d = 1$. Thus, we have to prove that for any C^3 function $f\colon \mathbb{R}^n \to \mathbb{R}$ with $f(0) = 0$, there exists a vector-valued function $g\colon \mathbb{R}^n \to \mathbb{R}$ such that $f(x) = g(x)^\top x$.

Since f is C^3 with $f(0) = 0$, we can write its Taylor expansion

$$f(x) = D^\top x + \frac{1}{2}x^\top H x + R(x) \tag{24}$$

where $D = \partial_x f(0)$ is the gradient of f at $x = 0$, $H = \partial_x^2 f(0)$ its Hessian, and where the remainder R is $O(\|x\|^3)$.

A priori, the function $R(x)/\|x\|^2$ is defined everywhere except $x = 0$. But since R is $O(\|x\|^3)$, $R(x)/\|x\|^2$ tends to 0 for $x = 0$, so it is a well-defined continuous function on the whole domain.

Thus, let us set

$$g(x) := D + \frac{1}{2}Hx + \frac{R(x)}{\|x\|^2}\, x. \tag{25}$$

Then, by construction, $g(x)^\top x = f(x)$ as needed. □

Proof. Proof of Theorem B.3 By the lemma, there exists a matrix-valued function g such that $\zeta(r) = g(r) \cdot r$ for $r \in \mathbb{R}^{\#S}$.

Define $B_1(s)$ to be the one-hot encoding $B_1(s) = \mathbb{1}_s/\rho(s) \in \mathbb{R}^{\#S}$, where $\mathbb{1}_i$ denotes the vector with all zeroes except a 1 at position i. By universal approximation theorems for neural networks, this choice of B_1 can be realized by a neural network with arbitrary good approximation (actually a one-layer neural network with identity weights and no activation function).

Then

$$z_1 = \mathbb{E}_{s\sim\rho}[r(s)B_1(s)] = \sum_s \rho(s)r(s)\mathbb{1}_s/\rho(s) = r. \tag{26}$$

Then in turn, for any B_2,

$$z_2 = \mathbb{E}_{s\sim\rho}[r(s)B_2(s, z_1)] = \sum_s \rho(s)r(s)B_2(s, r). \tag{27}$$

For each s and r, $B_2(s, r)$ is an element of $\mathbb{R}^d$. For each component $1 \leq i \leq d$, define

$$B_2(s, r)_i := g_{is}(r)/\rho(s) \tag{28}$$

where $g(r)$ is the matrix defined above. Then we have, by construction

$$\sum_s \rho(s)r(s)B_2(s, r)_i = \sum_s r(s)g_{is}(r) \tag{29}$$

namely

$$B_2(s, r) = g(r) \cdot r = \zeta(r) \tag{30}$$

as needed.

By universal approximation theorems for neural networks, it is possible to realize this choice of B_2 by a neural network, with arbitrarily good approximation.

Note: the principle of this proof extends to continuous state spaces, by taking a partition of unity for B_1 instead of a one-hot encoding, though this results in further approximation errors. □

B.2 Training Loss and Algorithmic Considerations for Auto-Regressive FB

Loss for FB-AR; sampling. While plain FB represents the successor measures as $M^{\pi_z}(s, a, \mathrm{d}s') \approx F(s, a, z)^\top B(s')\rho(\mathrm{d}s')$, auto-regressive FB uses $M^{\pi_z}(s, a, \mathrm{d}s') \approx F(s, a, z)^\top B(s', z)\rho(\mathrm{d}s')$. The training principle remains the same: learn F and B to minimize the error in this approximation. However, this leads to several changes in practice.

The error of the FB model can be measured as in plain FB, based on the Bellman equation satisfies by M^{π_z} (see Appendix B in Touati et al. (2023)). For each value of z, the Bellman loss on

$M^{\pi_z}(s, a, \mathrm{d}s') - F(s, a, z)^\top B(s')\rho(\mathrm{d}s')$ is

$$\mathcal{L}(F, B) = \mathbb{E}_{\substack{(s_t, a_t, s_{t+1}) \sim \rho \\ s' \sim \rho}} \left[\left(F(s_t, a_t, z)^\top B(s', z) - \gamma \bar{F}(s_{t+1}, \pi_z(s_{t+1}), z)^\top \bar{B}(s', z) \right)^2 \right] - 2\mathbb{E}_{(s_t, a_t, s_{t+1}) \sim \rho} \left[F(s_t, a_t, z)^\top B(s_{t+1}, z) \right] + \texttt{Const} \tag{31}$$

where `Const` is a constant term that we can discard since it does not depend on F and B. The only difference with plain FB is that now B depends on z.

For training, the variable z is sampled as in Touati et al. (2023): namely, z is sampled 50% of the time from a standard d-dimensional Gaussian (and later normalized) and 50% of the time from the B representation of a state ramdomly sampled from the training data (computed every step for normal FB and computed every 32 steps for AR FB for speed). Fig. 13 (Appendix E) tests the effect of only sampling z from a Gaussian.

For reasons discussed below, for FB we follow the original strategy of sampling a different z for each sampled transition, but for FB-AR we sample a unique z for the whole batch.

Algorithm 2 implements this loss, together with sampling of z.

Minibatch handling, and increased variance for auto-regressive FB. Having B depend on z has practical consequences for variance and minibatch sampling. In vanilla FB, starting from (31), it is possible to sample a minibatch of N transitions (s_t, a_t, s_{t+1}), a minibatch of N values of s', choose a different value of z for each s_t in the minibatch, compute the N values of $F(s_t, a_t, z)$ and $B(s')$, and compute the N^2 dot products $F(s_t, a_t, z)^\top B(s')$ involved in the loss, at a cost of N forward passes through F and B (Appendix A in Touati & Ollivier (2021)).

In auto-regressive FB, the value of z must be the same for F and B in the loss (31). Therefore, contrary to plain FB, we only sample a *single* value of z for the minibatch. Then we can compute the N values $F(s_t, a_t, z)$ and $B(s', z)$, and the N^2 dot products $F(s_t, a_t, z)^\top B(s', z)$ for the loss (31). This appears in Algorithm 2.

Using a single value of z per minibatch results in increased variance of auto-regressive FB with respect to vanilla FB and longer training times, which we observe in practice.

Indeed, if we want to keep using different values of z for each $F(s_t, a_t, z)$, we must either compute many more values $B(s', z)$ (one for each pair s' and z), or use fewer dot products by computing fewer values of $B(s', z)$ and only using them with $F(s_t, a_t, z)$ with the same z.

More precisely, in general we can proceed as follows: Let k be a hyperparameter controlling the number of distinct values of z we will use in the minibatch. We sample N transitions (s_t, a_t, s_{t+1}), N values of s', and split these samples into k groups of size N/k. For each group, we sample a value of z, we compute the values $F(s_t, a_t, z)$ and $B(s', z)$ for the states in that group, and we use all dot products $F(s_t, a_t, z)^\top B(s', z)$ within that group.

Thus, the solution we used just has $k = 1$: the same value of z is used throughout the minibatch, so we can compute N^2 dot products $F(s_t, a_t, z)^\top B(s', z)$ with only N forward passes through F and B. The other extreme would be $k = N$: for each (s_t, a_t, s_{t+1}), we pick a value of z and a value of s', compute $F(s_t, a_t, z)$ and $B(s', z)$, and form the dot product $F(s_t, a_t, z)^\top B(s', z)$. This uses more distinct values of z, but has only N dot products. In practice this would mean a reduced variance from sampling z, but an increased variance from sampling s'. Our chosen option $(k = 1)$ has the opposite trade-off. This choice was based on preliminary results showing that, while using a higher k appeared to learn slightly faster at the beginning, the fewer dot products led to convergence to slightly lower performance.

To some extent, this effect may represent a hindrance for autoregressive FB compared with vanilla FB. However, Fig. 12 (Appendix E) shows that this effect is limited: indeed, vanilla FB with a single z per minibatch performs only slightly worse than vanilla FB with many z's.

Normalization of B. As in Touati et al. (2023), to improve numerical conditioning on B, we use an auxiliary orthonormalization loss which ensures that B is approximately an orthognoal matrix. Indeed, one can change F and B without changing the B model, by $F \leftarrow FC$ and $B \leftarrow B(C^\top)^{-1}$ for any invertible matrix B, because the FB model only depends on the values of $F^\top B$ (Touati & Ollivier, 2021). In the case of auto-regressive features, since B depends on z, we can do this normalization separately for each z, without impacting the FB model. Explicitly, the orthonormalization loss is

$$\mathcal{L}_{\text{norm}}(B) := \mathbb{E}_z \left\| \mathbb{E}_{s\sim\rho}[B(s,z)B(s,z)^\top] - \text{Id} \right\|^2_{\text{Frobenius}} \tag{32}$$

$$= \mathbb{E}_z \mathbb{E}_{s\sim\rho,\, s'\sim\rho} \left[(B(s,z)^\top B(s',z))^2 - \|B(s,z)\|_2^2 - \|B(s',z)\|_2^2 \right] + \texttt{Const.} \tag{33}$$

where z is sampled as for the main loss.

Moreover, we further normalize the output of B: for each input (s, z) and each auto-regressive block in the output of B, we set $B(s,z) \leftarrow B(s,z)\frac{\sqrt{d_k}}{\|B(s,z)\|}$ with d_k the size of the block k, so that each output component is of size approximately 1. Fig. 13 (Appendix E) tests the effect of not using this output normalization.

Scale invariance in the reward, and normalization of z. In reinforcement learning, the optimal behavior is the same for reward r or reward $\alpha.r$ with any $\alpha > 0$. This inherent invariance property can be directly enforced within the architecture of FB to help with learning.

In non-autoregressive FB models, since $z = \mathbb{E}_s[r \times B(s)]$, the scaling $r \leftarrow \alpha \times r$ directly translates to a scaling in the task representation $z \leftarrow \alpha z$. Thus, one simple way to enforce invariance with respect to the rewards' scale is to always normalize z to a fixed norm (in practice, norm $\sqrt{d}$, so each of z value value expectedly has a magnitude close to 1). We denote this operation by the preprocessing function $fz = \bar{z} = \sqrt{d} \times \frac{z}{||z||}$, which we apply when feeding any z to π_θ and F_ϕ. Hence, this ensures that $F(s,a,z) = F(s,a,\alpha \times z)$ and $\pi_z = \pi_{\alpha \times z}$ by construction. As a result, the predictions and behaviors made when facing rewards r and $\alpha \times r$ are exactly the same.

However, in our auto-regressive FB models where $z = \mathbb{E}_s[r \times B(s,z)]$, the same strategy cannot be applied when feeding z as input to B. This is because we never have access to the full z until the very end of the inference procedure. Hence, given a reward r, we still want to have scale-invariance in B but have no means of performing a standard input normalization, as we have no access to the magnitude of the resulting z.

A first strategy to counteract this limitation could be to normalize z only within each auto-regressive group, $f_g(z) = \{f_g(z)_1, \dots, f_g(z)_n\} = \{\bar{z}_1, \dots, \bar{z}_n\}$. While this would enforce a fixed scale and not pose test-time issues it would non-trivially affect the information available to B about the actual task z, losing any notion of relative magnitude between different groups. For instance, in case for $n = d$, each normalized component $\bar{z}_k$ would be a binary scalar only preserving the *sign* from the corresponding task representation z_k.

We overcome these limitations by designing a new '*residual autoregressive* normalization strategy, f_{ar} compatible with the requirements of B's inference while still fully and exclusively preserving the information about z's direction as with traditional normalization.

Our strategy achieves these properties by using an iterative normalization scheme for each autoregressive component in its output $f_r(z) = \{f_r(z)_1, \dots, f_r(z)_n\}$. As with the aforementioned naive approach, we start with normalizing the first component z_1 within itself: $f_r(z)_1 = \bar{z}_1$. Then, we proceed to *residually normalize* all other z_k, also making use of all previous autoregressive components $z_1, \dots z_{k-1}$:

$$f_r(z)_k = \frac{z_k}{||z_{1:k}||},$$

where $z_{1:k}$ simply corresponds to the concatenated first k auto-regressive components of z. Finally, we also rescale each kth component by a constant factor $\sqrt{\sum_{i=0}^{k} d_i}$ to avoid biasing later components to have a smaller magnitude at initialization and incentivize $||f_r(z)|| \approx \sqrt{d}$. We note there is a

bijective map between traditional normalization and this auto-regressive scheme, thus, preserving the full information of the direction component of z without requiring the full vector.

B.3 Algorithms

Algorithm 1 FB

Inputs Offline dataset $\mathcal{D}$, number of ensemble networks M, randomly initialized network $\{F_{\theta_m}\}_{m\in[M]}$, B_ω and π_ϕ, transition mini-batch size I mixing probability τ_{mix},Polyak coefficient ζ, orthonormality regularisation coefficient λ.

for $t = 1, \dots$ **do**

  /* Sampling

  Sample I latent vectors

  $$z \sim \begin{cases} \mathcal{N}(0, I_d) & \text{with prob } 1-\tau_{\text{mix}} \\ B(s) \quad \text{where } s \sim \mathcal{D}, & \text{with prob } \tau_{\text{mix}} \end{cases}$$

  $z \leftarrow \sqrt{d}\frac{z}{\|z\|}$

  Sample a mini-batch of I transitions $\{(s_i, a_i, s_i')\}_{i\in[I]}$ from $\mathcal{D}$

  /* Compute FB loss

  Sample $a_i' \sim \pi_\phi(s_i', z_i)$ for all $i \in [I]$

  $$\mathscr{L}_{\text{FB}}(\theta_m, \omega) = \frac{1}{2I(I-1)} \sum_{j\neq k} \left(F_{\theta_k}(s_i, a_i, z_i)^\top B_\omega(s_k') - \gamma \frac{1}{M} \sum_{m\in[M]} F_{\theta_m^-}(s_i', a_i', z_i)^\top B_{\omega^-}(s_k') \right)^2$$
  $$- \frac{1}{I} \sum_i F_{\theta_k}(s_i, a_i, z_i)^\top B_\omega(s_i'), \quad \forall m \in [M]$$

  /* Compute orthonormality regularization loss

  $$\mathscr{L}_{\text{ortho}}(\omega) = \frac{1}{2I(I-1)} \sum_{i\neq k} (B_\omega(s_i')^\top B_\omega(s_k')^2 - \frac{1}{I} \sum_i B_\omega(s_i')^\top B_\omega(s_i')$$

  /* Compute actor loss

  Sample $a_i^\phi \sim \pi_\phi(s_i, z_i)$ for all $i \in [I]$

  $$\mathscr{L}_{\text{actor}}(\phi) = -\frac{1}{I} \sum_i \left(\min_{m\in[M]} F_{\theta_m}(s_i, a_i^\phi, z_i)^T z_i \right)$$

  /* Update all networks

  $\theta_m \leftarrow \theta_m - \xi \nabla_{\theta_m} (\mathscr{L}_{\text{FB}}(\theta_k, \omega)$ for all $m \in [M]$

  $\omega \leftarrow \omega - \xi \nabla_\omega (\sum_{l\in[m]} \mathscr{L}_{\text{FB}}(\theta_l, \omega) + \lambda \cdot \mathscr{L}_{\text{ortho}}(\omega))$

  $\phi \leftarrow \phi - \xi \nabla_\phi \mathscr{L}_{\text{actor}}(\phi)$

  /* Update target networks

  $\theta_m^- \leftarrow \zeta\theta_m^- + (1-\zeta)\theta_m$ for all $m \in [M]$

  $\omega^- \leftarrow \zeta\omega^- + (1-\zeta)\omega$

C Experimental Details

C.1 Network Architecture

- For the backward representation network $B(s, z)$, we first preprocess separately s and (s, z). We use for s preprocessing a feedforward neural network with one single hidden layer of 256 units. We use for (s, z) preprocessing a masked network with one single hidden dimension of 256 units, The masked network employs multiplicative binary masks to remove some connections, such that each output layer unit of an autoregressive block is only predicted from the input units of previous blocks. After preprocessing, we concatenate the two outputs and pass them into a two hidden layer masked network that outputs a d-dimensional embedding.
- For the forward network $F(s, a, z)$, we first preprocess separately (s, a) and (s, z) by two feedforward networks with one single hidden layer (with 1024 units) to 512-dimentional space. Then we concatenate their two outputs and pass it into a three hidden layer feedforward networks (with 1024 units) to output a d-dimensional vector. We use an ensemble of two networks for F.
- For the policy network $\pi(s, z)$, we first preprocess separately s and (s, z) by two feedforward networks with one single hidden layer (with 1024 units) to 512-dimentional space. Then we concatenate their two outputs and pass it into another four hidden layer feedforward network (with

Algorithm 2 FB-AWARE

Inputs Offline dataset $\mathcal{D}$, number of ensemble networks M, randomly initialized network $\{F_{\theta_m}\}_{m\in[M]}$, B_ω and π_ϕ, transition mini-batch size J, latent vector mini-batch size I, number of autoregressive blocks K, mixing probability τ_{mix},Polyak coefficient ζ, orthonormality regularisation coefficient λ, temperature β.

for $t = 1, \ldots$ **do**

  /* Sampling

  Sample I latent vectors

  $$z \sim \begin{cases} \mathcal{N}(0, I_d) & \text{with prob } 1-\tau_{\text{mix}} \\ (z_1, \ldots, z_K) = (B_1(s), \ldots, B_K(s, z_1, \ldots, z_{K-1})) \quad \text{where } s \sim \mathcal{D}, & \text{with prob } \tau_{\text{mix}} \end{cases}$$

  $z \leftarrow \sqrt{d}\frac{z}{\|z\|}$

  Sample a mini-batch of $I \times J$ transitions $\{(s_{i,j}, a_{i,j}, s'_{i,j})\}_{i\in[I], j\in[J]}$ from $\mathcal{D}$

  Sample $a'_{i,j} \sim \pi_\phi(s'_{i,j}, z_i)$ for all $i \in [I], j \in [J]$

  $$\mathscr{L}_{\text{FB}}(\theta_m, \omega) = \frac{1}{2IJ(J-1)} \sum_i \sum_{j\neq k} \Big(F_{\theta_k}(s_{i,j}, a_{i,j}, z_i)^\top B_\omega(s'_{i,k}, z_i) - \gamma \frac{1}{M} \sum_{m\in[M]} F_{\theta_m^-}(s'_{i,j}, a'_{i,j}, z_i)^\top B_{\omega^-}(s'_{i,k}, z_i) \Big)^2$$
  $$- \frac{1}{IJ} \sum_i \sum_j F_{\theta_k}(s_{i,j}, a_{i,j}, z_i)^\top B_\omega(s'_{i,j}, z_i), \quad \forall m \in [M]$$

  /* Compute orthonormality regularization loss

  $$\mathscr{L}_{\text{ortho}}(\omega) = \frac{1}{2IJ(J-1)} \sum_i \sum_{j\neq k} (B_\omega(s'_{i,j}, z_i)^\top B_\omega(s'_{i,k}, z_i)^2 - \frac{1}{IJ} \sum_i \sum_j B_\omega(s'_{i,j}, z_i)^\top B_\omega(s'_{i,j}, z_i)$$

  /* Compute actor loss

  $$A(s_{i,j}, a_{i,j}, z_i) \leftarrow \sum_m F_{\theta_m}(s_{i,j}, a_{i,j}, z_i)^T z_i - \mathbb{E}_{a'_{i,j}\sim\pi_\phi(s_{i,j}, z_i)}[\min_m F_{\theta_m}(s_{i,j}, a'_{i,j}, z_i)^T z_i]$$

  $$w(s_{i,j}, a_{i,j}, z_i) \leftarrow \frac{\exp(A(s_{i,j}, a_{i,j}, z_i)/\beta)}{\sum_{i',j'} \exp(A(s_{i',j'}, a_{i',j'}, z_{i'})/\beta)}$$

  $$w'(s_{i,j}, a_{i,j}, z_i) \propto \frac{w(s_{i,j}, a_{i,j}, z_i)}{\sum_{(i',j')\neq(i,j)} w(s_{i',j'}, a_{i',j'}, z_{j'})}$$

  $$\mathscr{L}_{\text{actor}}(\phi) = -\frac{1}{IJ} \sum_{i,j} \overline{w'}(s_{i,j}, a_{i,j}, z_i) \log \pi_\phi(a_{i,j} \mid s_{i,j}, z_i)$$

  /* Update all networks

  $\theta_m \leftarrow \theta_m - \xi \nabla_{\theta_m}(\mathscr{L}_{\text{FB}}(\theta_k, \omega)$ for all $m \in [M]$

  $\omega \leftarrow \omega - \xi \nabla_\omega(\sum_{l\in[m]} \mathscr{L}_{\text{FB}}(\theta_l, \omega) + \lambda \cdot \mathscr{L}_{\text{ortho}}(\omega))$

  $\phi \leftarrow \phi - \xi \nabla_\phi \mathscr{L}_{\text{actor}}(\phi)$

  /* Update target networks

  $\theta_m^- \leftarrow \zeta\theta_m^- + (1-\zeta)\theta_m$ for all $m \in [M]$

  $\omega^- \leftarrow \zeta\omega^- + (1-\zeta)\omega$

1024 units) to output to output a d_A-dimensional vector, then we apply a `Tanh` activation as the action space is $[-1, 1]^{d_A}$.

For all the architectures, we apply a layer normalization and `Tanh` activation in the first layer in order to standardize the states and actions. We use `Relu` for the rest of layers.

C.2 Hyperparameters

Table 2: Hyperparameters used for the backward / features architectures.

Hyperparameter	backward / features	Autoregressive backward / features
Input variables	s	(s, z)
Hidden layers	2	2
Hidden units	256	256
Activations	ReLU	ReLU
First-layer activation	layernorm + tanh	layernorm + tanh
Input normalization	None	residuel autoregressive l2-normalization for z
Output normalization	l2-normalization	autoregressive group l2-normalization

Table 3: Hyperparameters used for the forward/ successor features and actor architectures

Hyperparameter	forward / successor features	actors
Input variables	(s, a, z)	(s, z)
Embeddings	one over (s, a) and one over (s, z)	one over (s) and one over (s, z)
Embedding hidden layers	2	2
Embedding hidden units	1024	1024
Embedding output dim	512	512
Hidden layers	3	4
Hidden units	1024	1024
Activations	ReLU	ReLU
First-layer activation	layernorm + tanh	layernorm + tanh
Output activation	linear	tanh
Number of parallel networks	2	1

Table 4: Hyperparameters used for FB pretraining.

Hyperparameter	Value
batch-size	1024
z dimension d	env-dependant
F network	Tab. 3
actor network	Tab. 3
B network	Tab. 2
Learning rate for F	10^{-4}
Learning rate for actor	10^{-4}
Learning rate for B	10^{-4}
Coefficient for orthonormality loss	100
z distribution	
-goals from the online buffer	50%
-uniform on unit sphere	50%
Polyak coefficient for target network update	0.005
z-batch-size (for AR)	1
for AW	
- temperature β	sweep over $\{0.1, 1, 10, 100, 300\}$
-Number of sampled action to estimate Advantage	4
-Number of sampled action for ES	50

C.3 Out-Of-Dataset Tasks for DMC Locomotion

To evaluate our new unsupervised RL algorithm across a set of diverse unseen problems, we extend the DeepMind Control suite with 15 new unseen tasks as defined by the following objectives:

- **cheetah bounce** – Simulated cheetah agent is rewarded for advancing while elevating its trunk and maximizing vertical velocity.
- **cheetah march** – Simulated cheetah agent is rewarded for advancing at a constant pace.
- **cheetah stand** – Simulated cheetah agent is rewarded for standing upright on its back leg.
- **cheetah headstand** – Simulated cheetah agent is rewarded for standing on its head while raising its back leg.
- **quadruped bounce** – Simulated quadruped agent is rewarded for advancing while elevating its trunk and maximizing vertical velocity.
- **quadruped skip** – Simulated quadruped agent is rewarded for moving in a diagonal pattern across the environment.
- **quadruped march** – Simulated quadruped agent is rewarded for advancing at a constant pace.
- **quadruped trot** – Simulated quadruped agent is rewarded for advancing while minimizing feet contact with the ground.
- **walker flip** – Simulated walker for performing a cartwheel, flipping its body 360 degrees.
- **walker march** – Simulated walker agent is rewarded for advancing at a constant pace.
- **walker skyreach** – Simulated walker agent is rewarded for pushing either of its legs to maximize vertical reach.

- **walker pullup** – Simulated walker agent is rewarded for pushing its upper trunk vertically while keeping its feet firmly grounded.
- **humanoid dive** – Simulated humanoid agent is rewarded for diving head-first to maximize vertical velocity.
- **humanoid march** – Simulated humanoid agent is rewarded for advancing at a constant pace.
- **humanoid skip** – Simulated humanoid agent is rewarded for moving in a diagonal pattern across the environment.

We hope this new set of problems might facilitate the evaluation of simulated robotics agents for the broader RL field, even beyond the unsupervised setting.

D Full Tables of Results

D.1 Jaco Arm Results

Domain	Task	FB	FB-AW	FB-AWARE (4)	FB-AWARE (8)	LAP-AW
jaco	reach_bottom_left	$49.0_{\pm 25.5}$	$43.9_{\pm 8.6}$	$56.3_{\pm 8.6}$	$63.6_{\pm 6.4}$	$25.9_{\pm 5.9}$
jaco	reach_bottom_right	$30.8_{\pm 7.5}$	$71.5_{\pm 18.2}$	$57.6_{\pm 16.5}$	$58.6_{\pm 21.1}$	$34.0_{\pm 13.9}$
jaco	reach_random1	$18.0_{\pm 8.0}$	$42.9_{\pm 15.7}$	$64.4_{\pm 17.0}$	$63.9_{\pm 10.7}$	$20.4_{\pm 12.6}$
jaco	reach_random2	$23.4_{\pm 6.4}$	$55.5_{\pm 5.6}$	$72.8_{\pm 10.7}$	$63.7_{\pm 8.1}$	$14.3_{\pm 5.1}$
jaco	reach_random3	$43.2_{\pm 27.7}$	$39.6_{\pm 5.9}$	$53.1_{\pm 6.4}$	$59.0_{\pm 12.1}$	$14.6_{\pm 5.6}$
jaco	reach_random4	$32.6_{\pm 23.3}$	$57.4_{\pm 11.5}$	$68.4_{\pm 11.0}$	$69.9_{\pm 10.3}$	$24.1_{\pm 2.8}$
jaco	reach_top_left	$32.6_{\pm 12.3}$	$41.0_{\pm 5.4}$	$41.9_{\pm 8.3}$	$62.7_{\pm 14.9}$	$10.3_{\pm 2.2}$
jaco	reach_top_right	$21.5_{\pm 11.6}$	$25.9_{\pm 9.2}$	$43.6_{\pm 9.7}$	$48.3_{\pm 12.1}$	$21.4_{\pm 5.2}$
jaco	Average	$31.4_{\pm 15.3}$	$47.2_{\pm 10.0}$	$57.3_{\pm 11.0}$	$61.2_{\pm 12.0}$	$20.6_{\pm 6.7}$

Table 5: JACO results on RND dataset, with dimension $d = 64$

Domain	Task	FB	FB-AW	FB-AWARE (4)	FB-AWARE (8)	LAP-AW
jaco	reach_bottom_left	$33.8_{\pm 17.3}$	$76.0_{\pm 12.0}$	$88.1_{\pm 18.5}$	$64.6_{\pm 13.4}$	$41.3_{\pm 10.2}$
jaco	reach_bottom_right	$51.3_{\pm 10.2}$	$86.3_{\pm 9.4}$	$87.7_{\pm 15.0}$	$96.8_{\pm 6.9}$	$47.8_{\pm 18.1}$
jaco	reach_random1	$32.6_{\pm 18.1}$	$75.3_{\pm 10.5}$	$85.4_{\pm 9.2}$	$87.0_{\pm 13.0}$	$30.9_{\pm 5.3}$
jaco	reach_random2	$22.9_{\pm 10.0}$	$86.3_{\pm 9.1}$	$104.1_{\pm 7.5}$	$95.5_{\pm 12.7}$	$28.8_{\pm 6.3}$
jaco	reach_random3	$31.2_{\pm 9.0}$	$68.3_{\pm 11.3}$	$89.5_{\pm 17.7}$	$61.9_{\pm 8.2}$	$24.7_{\pm 7.2}$
jaco	reach_random4	$21.6_{\pm 6.3}$	$82.2_{\pm 9.6}$	$101.2_{\pm 17.6}$	$82.9_{\pm 10.5}$	$34.7_{\pm 10.9}$
jaco	reach_top_left	$44.4_{\pm 16.6}$	$59.5_{\pm 18.3}$	$56.5_{\pm 9.6}$	$46.0_{\pm 17.7}$	$32.1_{\pm 10.2}$
jaco	reach_top_right	$28.3_{\pm 13.0}$	$44.2_{\pm 12.5}$	$47.5_{\pm 5.6}$	$39.7_{\pm 8.4}$	$23.1_{\pm 7.1}$
jaco	Average	$33.3_{\pm 12.6}$	$72.2_{\pm 11.6}$	$82.5_{\pm 12.6}$	$71.8_{\pm 11.4}$	$32.9_{\pm 9.4}$

Table 6: JACO results on RND dataset, with dimension $d = 128$

Domain	Task	FB	FB-AW	FB-AWARE (4)	FB-AWARE (8)	LAP-AW
jaco	reach_bottom_left	$33.1_{\pm 23.3}$	$60.2_{\pm 26.8}$	$101.0_{\pm 7.5}$	$91.5_{\pm 16.8}$	$42.7_{\pm 15.0}$
jaco	reach_bottom_right	$10.5_{\pm 2.4}$	$69.2_{\pm 32.5}$	$116.9_{\pm 9.7}$	$90.9_{\pm 12.9}$	$63.9_{\pm 13.8}$
jaco	reach_random1	$31.9_{\pm 21.8}$	$49.5_{\pm 25.5}$	$86.0_{\pm 19.2}$	$84.7_{\pm 12.4}$	$45.4_{\pm 4.8}$
jaco	reach_random2	$32.1_{\pm 16.4}$	$50.9_{\pm 26.2}$	$99.7_{\pm 17.6}$	$94.9_{\pm 14.5}$	$47.2_{\pm 8.4}$
jaco	reach_random3	$43.6_{\pm 18.4}$	$39.6_{\pm 22.4}$	$72.4_{\pm 13.6}$	$81.5_{\pm 9.9}$	$44.9_{\pm 9.4}$
jaco	reach_random4	$32.8_{\pm 15.4}$	$58.3_{\pm 34.5}$	$98.8_{\pm 15.7}$	$93.1_{\pm 14.2}$	$51.3_{\pm 8.2}$
jaco	reach_top_left	$29.4_{\pm 11.2}$	$38.4_{\pm 18.6}$	$43.6_{\pm 10.3}$	$65.1_{\pm 21.1}$	$43.0_{\pm 6.1}$
jaco	reach_top_right	$27.7_{\pm 6.2}$	$29.8_{\pm 19.2}$	$44.8_{\pm 20.1}$	$45.4_{\pm 13.8}$	$37.0_{\pm 9.4}$
jaco	Average	$30.1_{\pm 14.4}$	$49.5_{\pm 25.7}$	$82.9_{\pm 14.2}$	$80.9_{\pm 14.4}$	$46.9_{\pm 9.4}$

Table 7: JACO results on RND dataset, with dimension $d = 256$

D.2 DMC Locomotion Results

Domain	Task	LAP	LAP-AW	FB-AW	FB	FB-ARE (4)	FB-ARE (8)
cheetah	walk	$641.0_{\pm 137.7}$	$528.9_{\pm 22.9}$	$520.8_{\pm 56.6}$	$780.3_{\pm 182.7}$	$737.5_{\pm 204.7}$	$686.1_{\pm 44.8}$
cheetah	run	$156.5_{\pm 36.8}$	$116.0_{\pm 6.8}$	$141.2_{\pm 19.8}$	$306.8_{\pm 87.3}$	$261.1_{\pm 95.3}$	$241.8_{\pm 57.3}$
cheetah	walk_backward	$930.9_{\pm 77.9}$	$452.3_{\pm 212.6}$	$839.5_{\pm 49.3}$	$732.5_{\pm 167.3}$	$769.9_{\pm 209.9}$	$762.3_{\pm 205.6}$
cheetah	run_backward	$230.7_{\pm 42.6}$	$109.7_{\pm 37.2}$	$196.2_{\pm 22.4}$	$136.5_{\pm 35.6}$	$174.1_{\pm 48.6}$	$186.4_{\pm 66.4}$
cheetah	in_dataset_avg	$489.8_{\pm 73.8}$	$301.7_{\pm 69.9}$	$424.4_{\pm 37.0}$	$489.0_{\pm 118.2}$	$485.7_{\pm 139.6}$	$469.2_{\pm 93.5}$
quadruped	walk	$509.1_{\pm 38.9}$	$418.3_{\pm 42.7}$	$438.2_{\pm 192.6}$	$608.4_{\pm 72.0}$	$630.1_{\pm 96.9}$	$604.0_{\pm 116.8}$
quadruped	run	$457.6_{\pm 27.7}$	$355.6_{\pm 86.5}$	$391.2_{\pm 91.9}$	$392.7_{\pm 31.4}$	$417.4_{\pm 30.6}$	$376.1_{\pm 29.2}$
quadruped	stand	$681.6_{\pm 221.6}$	$731.3_{\pm 166.6}$	$762.2_{\pm 152.8}$	$687.9_{\pm 29.6}$	$761.7_{\pm 75.9}$	$705.4_{\pm 58.1}$
quadruped	jump	$464.5_{\pm 167.3}$	$493.4_{\pm 147.1}$	$563.7_{\pm 139.1}$	$567.0_{\pm 10.6}$	$609.2_{\pm 42.3}$	$580.3_{\pm 37.1}$
quadruped	in_dataset_avg	$528.2_{\pm 113.9}$	$499.6_{\pm 110.7}$	$538.9_{\pm 144.1}$	$564.0_{\pm 35.9}$	$604.6_{\pm 61.4}$	$566.4_{\pm 60.3}$
walker	stand	$963.6_{\pm 15.3}$	$803.3_{\pm 61.9}$	$452.6_{\pm 85.8}$	$728.5_{\pm 83.0}$	$632.7_{\pm 151.7}$	$516.1_{\pm 191.0}$
walker	walk	$908.8_{\pm 28.1}$	$605.3_{\pm 36.4}$	$572.0_{\pm 25.3}$	$669.9_{\pm 46.6}$	$607.9_{\pm 140.2}$	$552.2_{\pm 268.4}$
walker	run	$318.7_{\pm 15.0}$	$196.6_{\pm 13.1}$	$181.2_{\pm 16.5}$	$356.2_{\pm 20.9}$	$290.4_{\pm 22.7}$	$240.0_{\pm 122.7}$
walker	spin	$982.9_{\pm 3.5}$	$627.9_{\pm 135.1}$	$963.7_{\pm 5.3}$	$974.9_{\pm 10.0}$	$983.4_{\pm 1.3}$	$788.2_{\pm 391.2}$
walker	in_dataset_avg	$793.5_{\pm 15.5}$	$558.3_{\pm 61.6}$	$542.4_{\pm 33.2}$	$682.4_{\pm 40.1}$	$628.6_{\pm 79.0}$	$524.1_{\pm 243.3}$
cheetah	bounce	$600.4_{\pm 23.0}$	$428.2_{\pm 210.5}$	$539.9_{\pm 25.8}$	$415.7_{\pm 119.2}$	$472.8_{\pm 44.2}$	$462.2_{\pm 23.8}$
cheetah	march	$290.8_{\pm 63.2}$	$233.6_{\pm 12.9}$	$279.2_{\pm 39.0}$	$561.4_{\pm 183.7}$	$531.5_{\pm 187.6}$	$460.9_{\pm 119.2}$
cheetah	stand	$790.1_{\pm 107.9}$	$249.5_{\pm 131.6}$	$738.7_{\pm 66.2}$	$780.9_{\pm 105.7}$	$629.3_{\pm 49.3}$	$762.9_{\pm 179.9}$
cheetah	headstand	$577.9_{\pm 145.1}$	$288.9_{\pm 236.8}$	$728.4_{\pm 83.3}$	$794.7_{\pm 9.4}$	$791.1_{\pm 52.0}$	$765.1_{\pm 70.3}$
cheetah	out_of_dataset_avg	$564.8_{\pm 84.8}$	$300.0_{\pm 147.9}$	$571.6_{\pm 53.6}$	$638.2_{\pm 104.5}$	$606.2_{\pm 83.3}$	$612.8_{\pm 98.3}$
quadruped	bounce	$179.5_{\pm 76.1}$	$123.2_{\pm 64.4}$	$189.3_{\pm 192.0}$	$276.1_{\pm 57.1}$	$196.0_{\pm 105.6}$	$251.1_{\pm 47.9}$
quadruped	skip	$365.3_{\pm 108.4}$	$458.0_{\pm 114.5}$	$559.5_{\pm 220.3}$	$603.3_{\pm 30.7}$	$635.2_{\pm 34.4}$	$615.3_{\pm 35.2}$
quadruped	march	$478.7_{\pm 14.1}$	$370.4_{\pm 80.7}$	$396.4_{\pm 123.7}$	$458.3_{\pm 20.4}$	$466.0_{\pm 38.7}$	$419.7_{\pm 32.0}$
quadruped	trot	$310.6_{\pm 7.0}$	$246.7_{\pm 54.6}$	$278.3_{\pm 122.4}$	$357.6_{\pm 12.1}$	$380.9_{\pm 47.6}$	$335.3_{\pm 35.4}$
quadruped	out_of_dataset_avg	$333.5_{\pm 51.4}$	$299.6_{\pm 78.5}$	$355.9_{\pm 164.6}$	$423.8_{\pm 30.1}$	$419.5_{\pm 56.6}$	$405.4_{\pm 37.6}$
walker	flip	$605.4_{\pm 42.4}$	$435.0_{\pm 27.7}$	$293.6_{\pm 83.8}$	$445.5_{\pm 77.4}$	$462.0_{\pm 68.7}$	$322.6_{\pm 147.7}$
walker	march	$695.8_{\pm 47.8}$	$364.3_{\pm 29.0}$	$359.9_{\pm 15.2}$	$518.4_{\pm 95.9}$	$400.5_{\pm 178.0}$	$390.3_{\pm 190.4}$
walker	skyreach	$653.8_{\pm 64.1}$	$423.1_{\pm 61.6}$	$406.0_{\pm 13.8}$	$417.0_{\pm 34.9}$	$331.7_{\pm 44.4}$	$261.5_{\pm 151.7}$
walker	pullup	$264.8_{\pm 80.5}$	$58.4_{\pm 39.2}$	$264.5_{\pm 60.5}$	$305.9_{\pm 109.4}$	$463.1_{\pm 107.5}$	$214.6_{\pm 155.1}$
walker	out_of_dataset_avg	$554.9_{\pm 58.7}$	$320.2_{\pm 39.4}$	$331.0_{\pm 43.3}$	$421.7_{\pm 79.4}$	$414.3_{\pm 99.7}$	$297.2_{\pm 161.2}$

Table 8: DMC Locomotion results on RND dataset, with dimension 64, averaged over 100 episodes. Humanoid is not included, as RND produces insufficient exploration for Humanoid: even classical single-task (non-FB) training fails.

Domain	Task	FB	FB-AW	FB-AWARE (4)	FB-AWARE (8)	LAP-AW
cheetah	walk	$985.3_{\pm 3.1}$	$983.6_{\pm 6.6}$	$967.5_{\pm 28.4}$	$982.0_{\pm 4.2}$	$978.5_{\pm 14.0}$
cheetah	run	$213.2_{\pm 123.5}$	$560.2_{\pm 40.7}$	$525.7_{\pm 53.6}$	$547.6_{\pm 20.5}$	$448.9_{\pm 222.9}$
cheetah	walk_backward	$971.0_{\pm 24.3}$	$979.7_{\pm 5.1}$	$984.1_{\pm 0.7}$	$985.1_{\pm 1.2}$	$982.8_{\pm 3.5}$
cheetah	run_backward	$302.5_{\pm 58.9}$	$473.9_{\pm 6.0}$	$454.0_{\pm 17.9}$	$465.8_{\pm 9.9}$	$413.8_{\pm 25.1}$
cheetah	in_dataset_avg	$618.0_{\pm 52.4}$	$749.4_{\pm 14.6}$	$732.8_{\pm 25.1}$	$745.1_{\pm 8.9}$	$706.0_{\pm 66.4}$
quadruped	walk	$389.5_{\pm 238.2}$	$935.5_{\pm 6.6}$	$926.8_{\pm 4.2}$	$919.3_{\pm 8.4}$	$819.1_{\pm 132.3}$
quadruped	run	$298.1_{\pm 105.4}$	$580.8_{\pm 62.5}$	$606.2_{\pm 33.0}$	$566.0_{\pm 47.8}$	$610.6_{\pm 89.3}$
quadruped	stand	$615.4_{\pm 191.0}$	$941.1_{\pm 6.6}$	$947.9_{\pm 4.7}$	$940.3_{\pm 9.4}$	$911.5_{\pm 27.9}$
quadruped	jump	$429.6_{\pm 134.5}$	$779.1_{\pm 48.1}$	$841.8_{\pm 8.2}$	$751.0_{\pm 75.9}$	$782.8_{\pm 48.4}$
quadruped	in_dataset_avg	$433.2_{\pm 167.3}$	$809.1_{\pm 30.9}$	$830.7_{\pm 12.5}$	$794.2_{\pm 35.4}$	$781.0_{\pm 74.5}$
walker	stand	$744.0_{\pm 119.3}$	$962.2_{\pm 14.7}$	$963.9_{\pm 3.7}$	$963.4_{\pm 4.3}$	$961.2_{\pm 8.2}$
walker	walk	$780.0_{\pm 310.8}$	$943.8_{\pm 20.8}$	$941.4_{\pm 7.4}$	$922.4_{\pm 16.4}$	$934.3_{\pm 12.7}$
walker	run	$422.5_{\pm 167.4}$	$594.9_{\pm 12.1}$	$606.5_{\pm 6.2}$	$600.8_{\pm 37.8}$	$518.9_{\pm 39.7}$
walker	spin	$481.6_{\pm 226.3}$	$894.7_{\pm 84.4}$	$820.9_{\pm 114.6}$	$894.8_{\pm 63.6}$	$802.0_{\pm 180.7}$
walker	in_dataset_avg	$607.0_{\pm 205.9}$	$848.9_{\pm 33.0}$	$833.2_{\pm 33.0}$	$845.3_{\pm 30.5}$	$804.1_{\pm 60.4}$
humanoid	walk	$9.5_{\pm 11.8}$	$793.5_{\pm 16.1}$	$789.4_{\pm 18.0}$	$791.5_{\pm 7.3}$	$715.4_{\pm 35.1}$
humanoid	stand	$7.7_{\pm 3.8}$	$720.0_{\pm 23.5}$	$728.3_{\pm 30.1}$	$711.6_{\pm 24.4}$	$587.2_{\pm 34.0}$
humanoid	run	$2.4_{\pm 1.6}$	$276.5_{\pm 11.0}$	$266.7_{\pm 4.8}$	$273.6_{\pm 5.2}$	$246.4_{\pm 9.7}$
humanoid	in_dataset_avg	$6.5_{\pm 5.7}$	$596.7_{\pm 16.9}$	$594.8_{\pm 17.6}$	$592.3_{\pm 12.3}$	$516.3_{\pm 26.3}$
cheetah	bounce	$351.8_{\pm 119.9}$	$479.0_{\pm 22.3}$	$506.9_{\pm 18.9}$	$494.6_{\pm 9.4}$	$338.4_{\pm 38.3}$
cheetah	march	$521.4_{\pm 161.8}$	$897.8_{\pm 38.1}$	$903.6_{\pm 30.6}$	$921.9_{\pm 8.9}$	$819.7_{\pm 60.8}$
cheetah	stand	$731.5_{\pm 248.4}$	$419.1_{\pm 48.7}$	$472.7_{\pm 49.4}$	$548.7_{\pm 37.7}$	$184.1_{\pm 44.9}$
cheetah	headstand	$560.4_{\pm 185.9}$	$849.7_{\pm 49.9}$	$848.0_{\pm 33.8}$	$806.0_{\pm 21.4}$	$7.2_{\pm 11.4}$
cheetah	out_of_dataset_avg	$541.3_{\pm 179.0}$	$661.4_{\pm 39.8}$	$682.8_{\pm 33.2}$	$692.8_{\pm 19.4}$	$337.3_{\pm 38.9}$
quadruped	bounce	$114.7_{\pm 98.1}$	$181.2_{\pm 61.4}$	$284.2_{\pm 31.7}$	$223.7_{\pm 20.6}$	$202.8_{\pm 52.1}$
quadruped	skip	$425.0_{\pm 139.2}$	$654.6_{\pm 88.0}$	$835.2_{\pm 86.9}$	$705.4_{\pm 67.8}$	$769.2_{\pm 94.6}$
quadruped	march	$304.1_{\pm 133.2}$	$747.1_{\pm 94.2}$	$791.8_{\pm 30.6}$	$800.0_{\pm 7.9}$	$742.5_{\pm 123.0}$
quadruped	trot	$228.3_{\pm 127.1}$	$573.0_{\pm 46.9}$	$614.8_{\pm 15.9}$	$594.6_{\pm 6.1}$	$509.8_{\pm 73.1}$
quadruped	out_of_dataset_avg	$268.0_{\pm 124.4}$	$539.0_{\pm 72.6}$	$631.5_{\pm 41.3}$	$580.9_{\pm 25.6}$	$556.1_{\pm 85.7}$
walker	flip	$404.7_{\pm 234.4}$	$909.9_{\pm 20.0}$	$913.6_{\pm 14.5}$	$914.5_{\pm 18.3}$	$780.2_{\pm 66.2}$
walker	march	$663.3_{\pm 245.6}$	$826.7_{\pm 41.3}$	$841.0_{\pm 45.1}$	$811.8_{\pm 17.7}$	$725.8_{\pm 25.7}$
walker	skyreach	$396.5_{\pm 35.2}$	$365.2_{\pm 42.7}$	$366.7_{\pm 25.2}$	$404.3_{\pm 51.5}$	$284.6_{\pm 52.7}$
walker	pullup	$124.1_{\pm 101.7}$	$334.4_{\pm 112.7}$	$523.7_{\pm 31.4}$	$454.8_{\pm 46.9}$	$113.9_{\pm 48.8}$
walker	out_of_dataset_avg	$397.2_{\pm 154.2}$	$609.0_{\pm 54.2}$	$661.3_{\pm 29.1}$	$646.4_{\pm 33.6}$	$476.1_{\pm 48.3}$
humanoid	dive	$165.8_{\pm 7.9}$	$404.1_{\pm 11.1}$	$409.8_{\pm 17.9}$	$396.5_{\pm 16.1}$	$242.6_{\pm 18.1}$
humanoid	march	$6.3_{\pm 8.4}$	$669.6_{\pm 23.5}$	$659.5_{\pm 23.6}$	$661.2_{\pm 18.3}$	$559.6_{\pm 53.1}$
humanoid	skip	$2.1_{\pm 0.9}$	$233.7_{\pm 39.2}$	$234.9_{\pm 11.4}$	$221.4_{\pm 15.1}$	$126.0_{\pm 18.2}$
humanoid	out_of_dataset_avg	$58.1_{\pm 5.7}$	$435.8_{\pm 24.6}$	$434.7_{\pm 17.6}$	$426.4_{\pm 16.5}$	$309.4_{\pm 29.8}$

Table 9: DMC locomotion results on MOOD dataset, with dimension = 64 for walker,cheetah, quadruped, and dimension = 128 for humanoid, averaged over 100 episodes

D.3 Additional reward prompts

We demonstrate the adaptability of our FB-AWARE model on the DMC humanoid by showcasing its behavior in response to various reward functions. In Figure 5, we illustrate the agent's actions when prompted by the following reward functions:

- LEFT_HAND: the task consists in raising the left hand while standing. Specifically, the reward function is defined as having a velocity close to zero (exponential term), having an upright torso, and maintaining the height of the left wrist above a certain threshold while keeping the height of the right wrist below a different threshold.

$$\mathrm{R}_{\text{LEFT_HAND}} = \exp(-(v_x^2 + v_y^2)) * \text{upright} * \mathbb{I}\{\text{left_wrist_z} > 2\} * \mathbb{I}\{\text{right_wrist_z} < 0.9\}$$

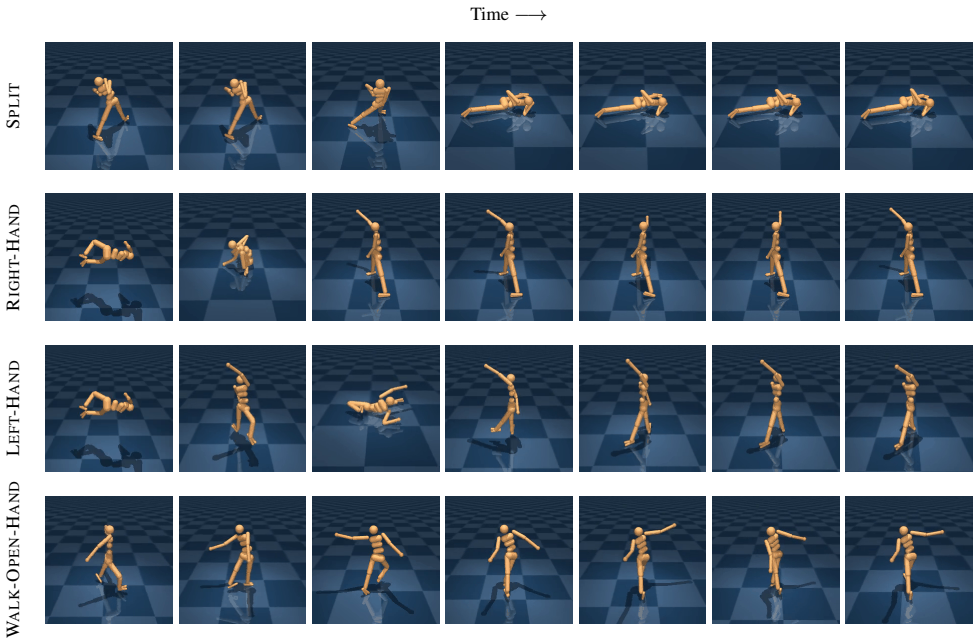

Figure 5: Example of behaviors inferred by from reward equations.

- RIGHT_HAND: the task consists in raising the right hand while standing. Specifically, the reward function is defined as having a velocity close to zero (exponential term), having an upright torso, and maintaining the height of the right wrist above a certain threshold while keeping the height of the left wrist below a different threshold.

$$\mathrm{R}_{\text{RIGHT_HAND}} = \exp(-(v_x^2 + v_y^2)) * \text{upright} * \mathbb{I}\{\text{left_wrist_z} < 0.9\} * \mathbb{I}\{\text{right_wrist_z} > 0.9\}$$

- WALK_OPEN_HAND: the task consists in walking while keeping the two hands open. The reward function is defined as having a velocity above some threshold and the absolute distance between the y coordinate of the left and right wrist above some threshold.

$$\mathrm{R}_{\text{WALK_OPEN_HAND}} = \mathbb{I}\{v_x^2 + v_y^2 > 5\} * \mathbb{I}\{|\text{left_wrist_y} - \text{right_wrist_y}| > 1.2\}$$

- SPLIT: the task consists in doing a split on the ground. The reward can be described as having a velocity close to zero, the height of the pelvis below some threshold and the absolute distance between the y coordinate of the left and right ankle above some threshold.

$$\mathrm{R}_{\text{SPLIT}} = \exp(-(v_x^2 + v_y^2)) * \mathbb{I}\{\text{z_pelvis} < 0.2\} * \mathbb{I}\{|\text{left_ankle_y} - \text{right_ankle_y}| > 0.5\}$$

E Ablations

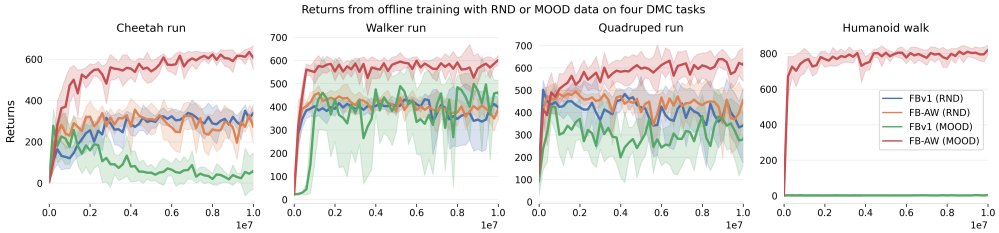

Figure 6: Performance on four representative tasks of the DMC when training FB-AW and vanilla FB (FBv1) either from mixed objective MOOD or pure RND data. The advantage of AW is clear on the mixed-objective MOOD datasets. The RND dataset does not allow FB to reach top performance.

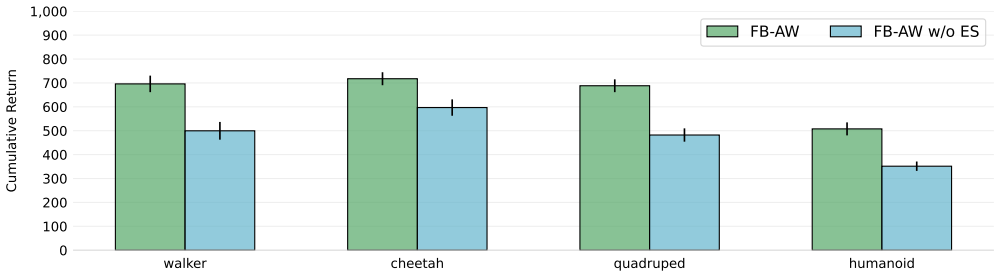

Figure 7: Cumulative reward averaged over all DMC Locomotion tasks, achieved by FB-AW with or without Evaluation-Sampling, trained on MOOD dataset.

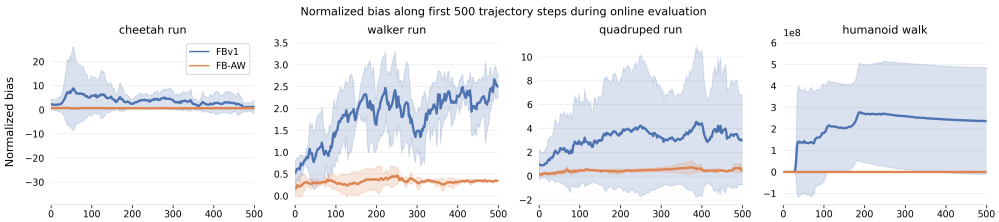

Figure 8: AW improves accuracy of reward prediction by the B model. We plot the bias of estimated rewards when progressing through a trajectory, namely, the difference between the actual trajectory return and the return $\sum_t B(s_t)^T z$ predicted by FB, after offline training on MOOD (averaged across 5 agents, 10 trajectories each). Vanilla FB provides overoptimistic values.

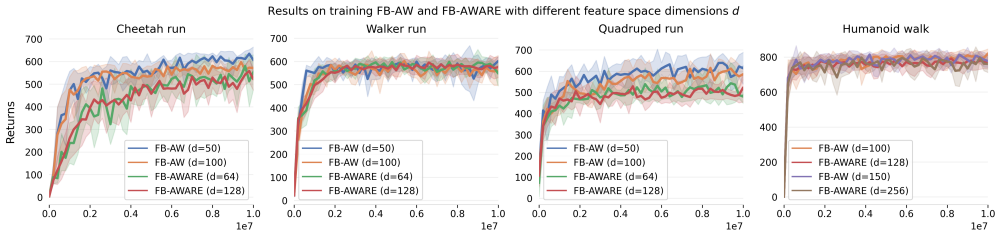

Figure 9: Effect of the z dimension for FB-AW and FB-AWARE on the MOOD mixed objective datasets

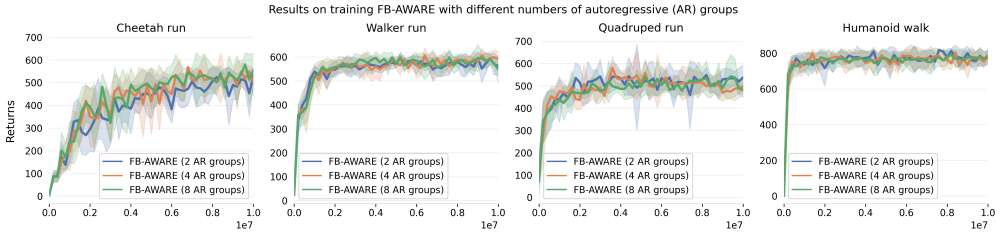

Figure 10: Effect of the number of autoregressive groups for FB-AWARE on the MOOD mixed objective datasets

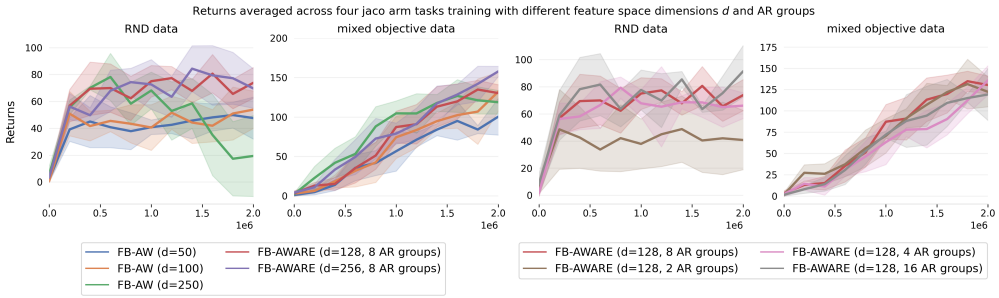

Figure 11: Effect of modifying the z dimension and the number of autoregressive groups for FB-AW and FB-AWARE, for performance in the Jaco arm environment

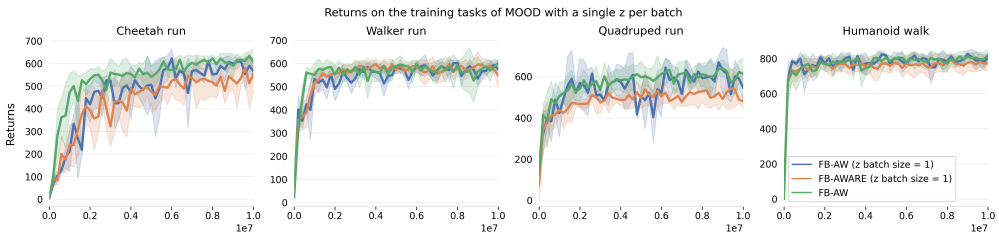

Figure 12: Effect of training FB-AW with a single z-per-batch like FB-AWARE (Section B.2)

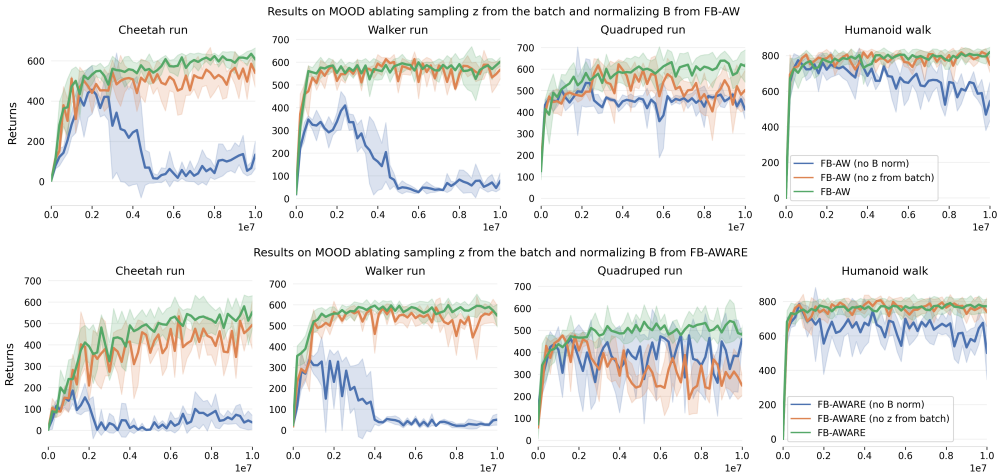

Figure 13: Ablations for Appendix B.2: Effects from training without the B normalization and without sampling 50% of z from other states in the minibatch for FB-AW (Top) and FB-AWARE (Bottom)

Domain	Task	Vanilla FB	FB(fully par. + no min.)	FB-AW(no fully par. + min.)	FB-AW (WIS)	FB-AW
cheetah	walk	$275.5_{\pm 343.5}$	$500.2_{\pm 285.2}$	$987.5_{\pm 0.6}$	$975.0_{\pm 23.5}$	$975.6_{\pm 24.6}$
cheetah	run	$77.1_{\pm 87.3}$	$106.0_{\pm 50.4}$	$548.3_{\pm 38.6}$	$548.4_{\pm 30.0}$	$610.7_{\pm 49.2}$
cheetah	walk_backward	$199.3_{\pm 209.1}$	$349.7_{\pm 394.5}$	$984.8_{\pm 0.4}$	$984.5_{\pm 1.2}$	$984.7_{\pm 0.4}$
cheetah	run_backward	$49.1_{\pm 58.6}$	$78.6_{\pm 118.3}$	$461.0_{\pm 6.6}$	$477.4_{\pm 7.8}$	$481.4_{\pm 4.2}$
cheetah	in_dataset_avg	$0.175_{\pm 0.203}$	$0.295_{\pm 0.249}$	$0.939_{\pm 0.019}$	$0.944_{\pm 0.022}$	$0.971_{\pm 0.028}$
quadruped	walk	$318.3_{\pm 65.7}$	$763.9_{\pm 145.2}$	$889.4_{\pm 80.5}$	$938.8_{\pm 34.7}$	$958.0_{\pm 9.1}$
quadruped	run	$279.4_{\pm 45.8}$	$439.8_{\pm 90.2}$	$446.1_{\pm 50.6}$	$599.4_{\pm 70.4}$	$673.4_{\pm 33.1}$
quadruped	stand	$619.6_{\pm 121.1}$	$831.5_{\pm 147.8}$	$901.3_{\pm 84.7}$	$944.9_{\pm 44.0}$	$975.9_{\pm 4.5}$
quadruped	jump	$435.0_{\pm 79.8}$	$644.1_{\pm 127.7}$	$708.2_{\pm 20.2}$	$767.4_{\pm 74.9}$	$798.4_{\pm 55.0}$
quadruped	in_dataset_avg	$0.493_{\pm 0.093}$	$0.800_{\pm 0.153}$	$0.877_{\pm 0.070}$	$0.976_{\pm 0.070}$	$1.026_{\pm 0.032}$
walker	stand	$906.5_{\pm 80.4}$	$976.2_{\pm 9.5}$	$964.8_{\pm 3.5}$	$978.9_{\pm 3.1}$	$956.1_{\pm 34.7}$
walker	walk	$892.8_{\pm 102.7}$	$939.9_{\pm 44.7}$	$946.7_{\pm 10.3}$	$960.1_{\pm 7.3}$	$955.4_{\pm 16.6}$
walker	run	$462.2_{\pm 37.5}$	$487.0_{\pm 44.1}$	$480.5_{\pm 52.6}$	$583.3_{\pm 20.9}$	$579.8_{\pm 45.8}$
walker	spin	$422.1_{\pm 121.4}$	$464.9_{\pm 196.4}$	$923.2_{\pm 30.1}$	$759.2_{\pm 173.6}$	$789.6_{\pm 117.7}$
walker	in_dataset_avg	$0.749_{\pm 0.093}$	$0.799_{\pm 0.081}$	$0.913_{\pm 0.032}$	$0.918_{\pm 0.055}$	$0.917_{\pm 0.061}$
humanoid	walk	$3.6_{\pm 5.0}$	$2.3_{\pm 1.1}$	$677.6_{\pm 52.8}$	$779.9_{\pm 37.1}$	$785.0_{\pm 20.4}$
humanoid	stand	$5.1_{\pm 1.9}$	$4.8_{\pm 0.8}$	$481.3_{\pm 59.5}$	$750.3_{\pm 43.3}$	$801.7_{\pm 45.8}$
humanoid	run	$1.1_{\pm 0.8}$	$1.1_{\pm 0.6}$	$256.8_{\pm 22.9}$	$274.4_{\pm 20.4}$	$294.9_{\pm 17.5}$
humanoid	in_dataset_avg	$0.005_{\pm 0.004}$	$0.004_{\pm 0.002}$	$0.793_{\pm 0.074}$	$0.965_{\pm 0.058}$	$1.014_{\pm 0.049}$
cheetah	bounce	$109.6_{\pm 83.9}$	$315.4_{\pm 102.8}$	$436.3_{\pm 47.1}$	$469.3_{\pm 39.7}$	$502.5_{\pm 19.2}$
cheetah	march	$135.8_{\pm 182.5}$	$257.8_{\pm 110.2}$	$766.9_{\pm 92.7}$	$892.6_{\pm 35.0}$	$917.4_{\pm 13.4}$
cheetah	stand	$206.6_{\pm 383.7}$	$684.7_{\pm 267.4}$	$426.9_{\pm 191.6}$	$288.9_{\pm 162.6}$	$387.6_{\pm 98.4}$
cheetah	headstand	$233.8_{\pm 369.0}$	$923.7_{\pm 57.1}$	$488.0_{\pm 345.1}$	$407.0_{\pm 369.8}$	$854.5_{\pm 42.0}$
cheetah	out_dataset_avg	$0.214_{\pm 0.308}$	$0.684_{\pm 0.166}$	$0.660_{\pm 0.207}$	$0.642_{\pm 0.188}$	$0.832_{\pm 0.053}$
quadruped	bounce	$133.6_{\pm 82.8}$	$248.9_{\pm 41.0}$	$204.3_{\pm 61.0}$	$246.6_{\pm 31.3}$	$231.8_{\pm 18.4}$
quadruped	skip	$478.9_{\pm 113.2}$	$601.0_{\pm 72.3}$	$554.0_{\pm 1.3}$	$722.7_{\pm 112.4}$	$836.1_{\pm 104.4}$
quadruped	march	$280.8_{\pm 73.1}$	$517.2_{\pm 108.0}$	$478.8_{\pm 41.8}$	$802.1_{\pm 55.9}$	$860.2_{\pm 26.7}$
quadruped	trot	$178.3_{\pm 38.0}$	$381.5_{\pm 82.6}$	$412.1_{\pm 48.6}$	$605.3_{\pm 16.9}$	$615.9_{\pm 13.6}$
quadruped	out_dataset_avg	$0.524_{\pm 0.175}$	$0.897_{\pm 0.159}$	$0.834_{\pm 0.105}$	$1.187_{\pm 0.105}$	$1.245_{\pm 0.075}$
walker	flip	$630.3_{\pm 111.7}$	$744.4_{\pm 105.4}$	$771.8_{\pm 104.5}$	$891.2_{\pm 24.9}$	$896.5_{\pm 41.2}$
walker	march	$709.4_{\pm 182.1}$	$744.6_{\pm 183.3}$	$593.9_{\pm 12.6}$	$788.3_{\pm 63.7}$	$797.2_{\pm 36.8}$
walker	skyreach	$321.7_{\pm 130.0}$	$423.0_{\pm 70.3}$	$392.7_{\pm 15.1}$	$364.3_{\pm 48.0}$	$389.8_{\pm 6.5}$
walker	pullup	$114.7_{\pm 98.5}$	$101.3_{\pm 104.1}$	$33.9_{\pm 8.2}$	$309.7_{\pm 145.5}$	$376.5_{\pm 219.5}$
walker	out_dataset_avg	$0.645_{\pm 0.196}$	$0.747_{\pm 0.163}$	$0.674_{\pm 0.052}$	$0.849_{\pm 0.101}$	$0.889_{\pm 0.101}$
humanoid	dive	$159.2_{\pm 12.2}$	$166.6_{\pm 29.5}$	$357.1_{\pm 36.6}$	$387.9_{\pm 39.8}$	$404.0_{\pm 5.7}$
humanoid	march	$2.9_{\pm 3.5}$	$2.9_{\pm 1.8}$	$479.6_{\pm 48.5}$	$623.1_{\pm 64.6}$	$645.9_{\pm 49.7}$
humanoid	skip	$2.0_{\pm 1.2}$	$1.7_{\pm 0.4}$	$81.3_{\pm 26.7}$	$254.4_{\pm 23.8}$	$250.1_{\pm 24.0}$
humanoid	out_dataset_avg	$0.112_{\pm 0.011}$	$0.117_{\pm 0.021}$	$0.536_{\pm 0.072}$	$0.788_{\pm 0.079}$	$0.805_{\pm 0.049}$

Table 10: Ablations regarding the various components from Section 3.2: advantage weighting, using the average versus the min of the two target networks for representing uncertainty, using fully parallel architectures for the two target networks, and using improved weighted importance sampling (IWIS) versus ordinary WIS. As described in the text, FB-AW (right column) has advantage weighting, uses the average instead of the min, has fully parallel architectures, and uses IWIS. Vanilla FB (left column) has the opposite settings. We compare other combinations in between. We report performance on the mixed objective datasets from MOOD, on both in-dataset and out-of-dataset tasks. The representation dimension is $d = 50$ for Cheetah, Quadruped, Walker, and $d = 100$ for Humanoid.