The Laplacian Keyboard: Beyond the Linear Span

Siddarth Chandrasekar **Marlos C. Machado***
Department of Computing Science, University of Alberta, Canada
Alberta Machine Intelligence Institute (Amii)
*Canada CIFAR AI Chair
siddart2@ualberta.ca, machado@ualberta.ca

Abstract

Across scientific disciplines, Laplacian eigenvectors serve as a fundamental basis for simplifying complex systems, from signal processing to quantum mechanics. In reinforcement learning (RL), they similarly form a basis over the state space, enabling reward functions to be approximated by projection onto a small set of eigenvectors. This projection makes zero-shot control possible, but it also imposes a fundamental limitation: the induced policies are only as expressive as the linear span of the chosen eigenvectors. We introduce the Laplacian Keyboard (LK), a hierarchical framework that goes beyond this linear span. LK constructs a task-agnostic library of behaviors from these eigenvectors, forming a behavior basis guaranteed to contain the optimal policy for any reward within the linear span. A meta-policy learns to stitch these behaviors dynamically, enabling efficient learning of policies outside the original linear constraints. We bound the value-function error incurred by approximating rewards with a finite Laplacian basis. Empirically, LK improves over the zero-shot solution and is more sample efficient than standard RL methods.

1 Introduction

Primary causes are unknown to us; but are subject to simple and constant laws, which may be discovered by observation, the study of them being the object of natural philosophy. — J Fourier

This observation from Fourier (1878) highlights a recurring objective in science: *finding simple laws that explain complex phenomena*. Across disciplines, this often involves decomposing systems into fundamental components that reveal underlying structure. Fourier's analysis of heat diffusion demonstrated that temperature evolution can be expressed as independent modes, each an eigenfunction of the Laplacian operator with its own decay rate. Similarly, the Fourier transform decomposes signals into sinusoidal components—also Laplacian eigenvectors—enabling efficient filtering and compression. These examples share a common thread: the eigenvectors of the Laplacian provide a natural basis that simplifies analysis and computation. In this work, we explore this principle in RL, *showing that the eigenvectors of the graph Laplacian form a principled basis for RL—and that hierarchically composing them enables an agent to go beyond the linear span of the basis.*

Prior work has explored bases that can be linearly composed to efficiently solve new tasks, including bases over value functions (Mahadevan, 2005; Bellemare et al., 2019; Farebrother et al., 2023), policies (Agarwal et al., 2025a), options (Barreto et al., 2019; Alegre et al., 2025) and state visitation distributions (Blier et al., 2021; Touati et al., 2023). These methods derive their bases from two seemingly distinct perspectives: *representations* and *behaviors*.

Representation-based methods learn features to approximate task-relevant quantities. Proto-Value Functions (PVFs; Mahadevan, 2005), an early example of this principle in RL, employ graph Laplacian eigenvectors as a basis for value function approximation. Given a new task, the agent learns the linear weights that combine these eigenvectors to approximate the value function. More recently,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

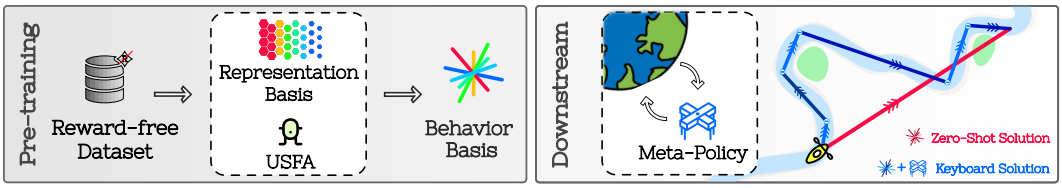


Figure 1: Overview of the Laplacian Keyboard. (*Left*) A reward-free dataset is used to learn a Laplacian representation basis and a USFA, inducing a behavior basis—depicted here as straight-line trajectories in the environment. (*Right*) In a downstream navigation task, a zero-shot policy selects a single behavior and executes it throughout the episode, yielding a potentially suboptimal trajectory (red). The LK meta-policy instead sequences over behaviors, composing a piecewise trajectory (blue) that better solves the task.

Forward-Backward representations (FB; Touati et al., 2023) learn a low-rank representation of the successor measure (Blier et al., 2021) of a policy class, which is then used to approximate reward functions. For any new task, the agent infers the weights that best approximate the corresponding reward function for this basis. If the learned representation spans the reward function, FB produces the optimal policy with no further learning involved. This enables FB to produce *zero-shot* solutions. However, this zero-shot solution is constrained by the linear span of the learned features—the policy can be suboptimal when the task lies outside this span.

The second category learns a basis of behaviors (or options; Sutton et al., 1999) rather than features. Given a new task, these methods use generalized policy improvement (GPI; Barreto et al., 2017) by evaluating each option in the basis and combining them using linear weights, guaranteeing performance at least as good as that of the best individual option, and often better. The Option Keyboard framework (Barreto et al., 2019) extends this idea by learning state-dependent weights, enabling the agent to dynamically *stitch* together options across states. This allows the agent to solve a broader class of tasks than is possible with a single fixed combination of the behavior basis. However, a central limitation is that the choice of the option basis is not well defined and typically relies on manual specification of either the options or their associated reward functions.

In this work, we bridge representation- and behavior-based approaches through the Laplacian Keyboard (LK), a framework that exploits the coupling between graph Laplacian eigenvectors and option discovery. While prior work has established that these eigenvectors induce task-agnostic behaviors (Machado et al., 2017, 2018, 2023), we show how they also form a principled basis for reward approximation. LK combines eigenvectors learned from an offline, reward-free dataset with Universal Successor Feature Approximators (USFA; Borsa et al., 2019), guaranteeing optimal policies for any task whose reward lies within the span of the learned Laplacian eigenvectors.

Many downstream tasks, however, fall outside this regime. To address this limitation, LK introduces efficient adaptation via sequential composition (see Figure 1). Rather than relying on a single linear combination of basis elements, LK decomposes complex tasks into simpler subproblems. Each USFA-induced policy becomes a reusable option in a library of task-agnostic behaviors, and a meta-policy learns to select and stitch these options efficiently. This hierarchical structure enables LK to solve tasks whose reward functions cannot be represented within the basis span. By unifying zero-shot optimality within the basis with efficient composition beyond it, LK overcomes key limitations of both representation- and behavior-based methods.

In summary, this work makes two contributions. (1) Our primary contribution is the *Laplacian Keyboard*, a hierarchical framework that composes Laplacian-induced behaviors via a learned meta-policy. This allows the agent to go beyond the linear span of the representation, yielding improvements over zero-shot baselines while maintaining strong sample efficiency relative to standard RL agents. (2) We complement this with an analysis of Laplacian eigenvectors as a reward basis, providing empirical validation and theoretical value-function approximation error bounds that characterize the expressivity of a finite Laplacian basis.

2 Background

We first formalize the problem addressed by LK and then review related approaches that tackle the same problem by learning a basis of representations or behaviors. We use calligraphic capitals for

sets (e.g., $\mathcal{S}, \mathcal{A}$), capitals for random variables (e.g., S_t, A_t), bold capitals for matrices (e.g., $\mathbf{L}, \mathbf{P}_\pi$), non-bold lowercase for functions (e.g., π, r), and bold lowercase for vectors (e.g., $\mathbf{w}, \mathbf{e}$). Functions over the state or action space are often represented in vectorized form by indexing their values, denoted using bold notation (e.g., $\mathbf{r}$).

2.1 Problem Definition

In RL, the agent-environment interaction is modeled as a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $p(\cdot \mid s, a)$, and discount factor γ. In this work, we define the reward r as a function of the next state. Specifically, for any transition (s, a, s'), the state-based reward is given by $r(s') = \sum_{s,a} p(s' \mid s, a)\, r(s, a, s')$. The goal is to learn a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes discounted expected return, expressed via the action-value function $q_\pi(s, a) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r(S_{t+1}) \mid S_0 = s, A_0 = a\Big]$, where $S_{t+1} \sim p(\cdot \mid S_t, \pi(S_t))$.

Rather than optimizing a single fixed reward, we pre-train an agent to solve *any* downstream task efficiently, without further large-scale environment interaction. Similar to Touati et al. (2023) and Agarwal et al. (2025a), we formulate this objective as a two-phase process, commonly referred to as *Unsupervised RL*. In the pre-training phase, the agent interacts with the environment for M steps to learn the basis without access to any reward function. In the subsequent downstream phase, tasks are specified through reward functions, and the agent must leverage its prior knowledge to solve these tasks within N steps, where $N \ll M$.

2.2 Solution Methods

We provide a brief overview of related unsupervised RL methods, organized by whether they use *representations* or *options* as the underlying basis. A more detailed review is given in Appendix A.

Representation-based methods learn a state representation $\phi : \mathcal{S} \to \mathbb{R}^k$ of dimension k during pre-training, defining a representation basis to approximate downstream rewards as:

$$r(s) \approx \mathbf{w}^\top \boldsymbol{\phi}(s), \quad \forall s \in \mathcal{S}.$$

Simultaneously, a USFA is used to train a policy $\pi(a \mid s, \mathbf{w})$ (Borsa et al., 2019). In the downstream phase, once a task is specified, a weight vector $\mathbf{w}$ is estimated from N samples via simple linear regression to approximate the reward function, which parameterizes the USFA. When the reward function lies exactly within the span of $\boldsymbol{\phi}$, the USFA recovers the optimal policy, making the choice of $\boldsymbol{\phi}$ and the USFA training procedure critical (see the work by Touati et al. (2023); Borsa et al. (2019); Ollivier (2025) for further discussion). This quick adaptation using only a few samples via linear regression is referred to as *zero-shot* learning by Touati et al. (2023).

Methods like Forward-Backward Representations (Touati et al., 2023) and Proto Successor Measure (Agarwal et al., 2025a) model $\boldsymbol{\phi}$ as a low-rank representation of the successor measure of a class of policies. These approaches have been shown to scale to large state and action spaces with minimal human prior (Tirinzoni et al., 2025). The primary limitation of these approaches is expressivity: the linear span of $\boldsymbol{\phi}$ must be rich enough to approximate all potential downstream rewards, often requiring a high-dimensional k to cover a diverse set of complex tasks.

Option-based methods take a complementary approach by learning a reusable option basis that can be linearly combined to solve new tasks. Most methods assume access to reward features $\phi : \mathcal{S} \to \mathbb{R}^d$, which define this basis: each option is trained to be optimal for a reward induced by these features. For downstream rewards of the form $r(s) = \mathbf{w}^\top \boldsymbol{\phi}(s)$, options can be combined via General Policy Evaluation and Improvement (GPE & GPI; Barreto et al., 2020), yielding a policy at least as good as any individual option. We refer to Borsa et al. (2019) for details on GPE & GPI and its connection to the USFA framework.

The Option Keyboard (OK) framework (Barreto et al., 2019) generalizes this approach by learning a meta-policy π_{OK} that composes options using state-dependent weights at execution time, rather than a single fixed combination, enabling solutions to a broader class of tasks (Chandrasekar and Machado, 2025). However, OK has two key limitations: (1) it relies on manually specified reward features $\boldsymbol{\phi}$, and (2) lacks a principled procedure for constructing the option basis. While Alegre et al. (2025) partially addresses the latter, dependence on handcrafted features remains a central limitation when such features are difficult to specify. We discuss this work further in Section 6.

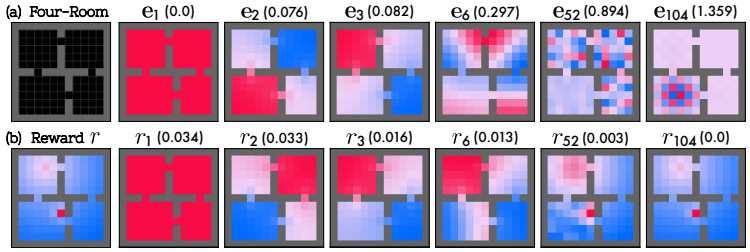


Figure 2: Illustration of the Laplacian basis in the `Four-Rooms` environment (Sutton et al., 1999). Color intensity reflects the magnitude of each value; scales are normalized per subplot. **(a)** Selected graph Laplacian eigenvectors under a uniform random policy; parenthetical values indicate the eigenvector graph total variation (Eqn. 1). **(b)** A sample reward function and its reconstruction using the first k eigenvectors; parenthetical values denote mean squared reconstruction error.

3 The Laplacian Basis

We motivate the use of graph Laplacian eigenvectors as a basis for approximating reward functions. An MDP under a fixed policy π induces a graph with states as vertices and transitions as edges. In the discrete case, the graph Laplacian is defined as $\mathbf{L} \doteq \mathbf{I} - f_{\text{sym}}(\mathbf{P}_\pi)$, where $\mathbf{P}_\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(a|s) p(s'|s, a)$ denotes the transition probability under π, and f_{sym} symmetrizes $\mathbf{P}_\pi$ by averaging it with its time reversal (see Appendix B.1.2 for details). Let $\mathbf{e}_i$ denote the i-th eigenvector of $\mathbf{L}$, ordered by eigenvalue. These eigenvectors encode state-space structure naturally: smaller eigenvalues correspond to smoother eigenvectors, ranging from the constant $\mathbf{e}_1$ to progressively more oscillatory patterns (see Figure 2a).

To formalize this notion of smoothness, we rely on the *graph total variation* (Zhu and Rabbat, 2012); for any function $f : \mathcal{S} \to \mathbb{R}$ defined over the state space:

$$\|f\|_G \doteq \left[\tfrac{1}{2} \sum\nolimits_{(i,j) \in \mathcal{S} \times \mathcal{S}} \mathbf{P}_\pi(i, j)(f(i) - f(j))^2 \right]^{\frac{1}{2}}. \tag{1}$$

The graph total variation captures the local variations weighted by the transition probabilities, smaller values indicating that the signals vary smoothly across connected states (see Figure 2a).

The Laplacian representation is the mapping $\phi : \mathcal{S} \to \mathbb{R}^k$ $(0 < k \leq |\mathcal{S}|)$, defined by $\boldsymbol{\phi}(s) = [\mathbf{e}_1[s], \mathbf{e}_2[s], \ldots, \mathbf{e}_k[s]]$, where $\mathbf{e}_i[s]$ denotes the s-th entry of eigenvector $\mathbf{e}_i$. The eigenvectors of $\mathbf{L}$ form an orthonormal basis for the state-space function class. Stacking representations across all states yields the *Laplacian basis* $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}| \times k}$. With the complete basis ($k = |\mathcal{S}|$), any reward function $r : \mathcal{S} \to \mathbb{R}$ admits an exact decomposition: $\hat{\mathbf{r}}_k \doteq \boldsymbol{\Phi} \left(\boldsymbol{\Phi}^\top \mathbf{r} \right)$ (the least-squares solution). However, computing and storing the full basis is infeasible for large state spaces.

This constraint requires restricting the representation to the first $k \ll |\mathcal{S}|$ eigenvectors, imposing an inductive bias. These eigenvectors, corresponding to the smallest eigenvalues, capture the lowest-frequency components of functions defined over the state space. As a result, any reward function with significant high-frequency structure lies outside the span of this truncated basis and cannot be reconstructed exactly (Figure 2b). This mismatch propagates to the prediction problem: the optimal value function computed from the approximated reward need not coincide with the true optimal value function. Theorem 3.1 bounds this mismatch.

Theorem 3.1 (Value Approximation Error Bound)**.** *Let $r : \mathcal{S} \to \mathbb{R}$ be a reward function with bounded variation*[1] *and $\hat{\mathbf{r}}_k$ its reconstruction using the first k eigenvectors. Let $\mathbf{v}^*$ and $\mathbf{v}_k^*$ be their corresponding optimal value functions. Then,*

$$\|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \frac{\|r\|_G}{(1 - \gamma)\sqrt{\lambda_{k+1}}}, \tag{2}$$

where λ_{k+1} is the smallest eigenvalue of the graph Laplacian not included in the truncation.

[1] A formal definition of *bounded variation* is provided in Eqn. 10 in Appendix B.1. Intuitively, it requires the reward function to vary smoothly over the state space.

The proof is provided in Appendix B.2. It proceeds by first establishing an upper bound on the reconstruction error of the reward function when using the first k eigenvectors of the graph Laplacian, then using the Bellman contraction property to show that this error amplifies by no more than $(1-\gamma)^{-1}$, yielding the stated bound on the value function.

The bound identifies two factors that jointly govern approximation error: (1) the smoothness of the reward, captured by $\|r\|_G$, and (2) the size of the basis k, which controls $\lambda_k \in [0, 2]$. Smaller $\|r\|_G$ and larger λ_{k+1} both tighten the bound, showing how signal smoothness and basis dimensionality together determine the quality of the induced value function. This result serves as both a justification and a limitation of using a truncated Laplacian basis for zero-shot control: smooth rewards are well-approximated by a small basis, while rewards with substantial high-frequency content demand larger ones—and no fixed truncated basis suffices for all reward functions. This motivates the hierarchical composition introduced next, which treats the Laplacian basis not as a complete solution, but as a starting point for tackling tasks beyond the span of the representation.

4 The Laplacian Keyboard

Having established the Laplacian representation as an effective reward basis, we now describe how the Laplacian Keyboard is constructed and used for downstream control (see Appendix H for the pseudocode).

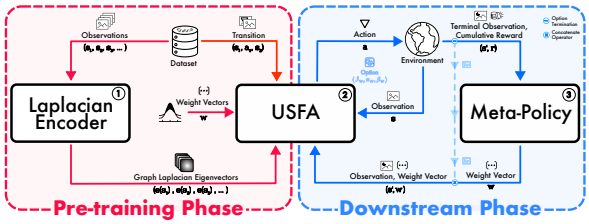


Figure 3: The Laplacian Keyboard framework.

4.1 Pre-training Phase

The goal of the pre-training phase is to learn a representation from reward-free data such that, when presented with a new task, the agent can solve it with only a handful of rewarded transitions. We achieve this by learning the Laplacian basis and training a USFA over it, as described below.

We assume access to a reward-free dataset, $\mathcal{D}$, that induces a transition kernel, $\mathbf{P}_\pi$, and its associated graph Laplacian, $\mathbf{L}$. The first k eigenvectors of $\mathbf{L}$ are approximated via the Augmented Lagrangian Laplacian Objective (Gomez et al., 2024), a contrastive objective comprising an alignment term that minimizes the distance between representations of transitionally adjacent states, and a repulsion term that enforces representational orthonormality—jointly recovering the eigenvectors of $\mathbf{L}$ (see Appendix C.2).

The learned representation $\boldsymbol{\phi}(s) \in \mathbb{R}^k$, where each dimension corresponds to a Laplacian eigenvector,[2] is used to parameterize a family of reward functions of the form $r_{\mathbf{w}}(s) = \mathbf{w}^\top \boldsymbol{\phi}(s)$. To model policies and value functions for this reward family, we train a Universal Successor Feature Approximator (USFA; Borsa et al., 2019), which generalizes successor representations (Dayan, 1993) by conditioning on the reward parameter $\mathbf{w}$. The USFA consists of a policy π and a corresponding successor feature estimator $\boldsymbol{\psi}$. The successor features (Barreto et al., 2017) encode the expected discounted accumulation of the state features under the conditioned policy:

$$\boldsymbol{\psi}(s, a, \mathbf{w}) = \mathbb{E}_{\pi_{\mathbf{w}}}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(S_{t+1}) \,\middle|\, S_0 = s,\, A_0 = a\right]. \tag{3}$$

Training of the USFA is performed off-policy on transitions (s, a, s') from $\mathcal{D}$, with weight vectors $\mathbf{w}$ sampled to jointly condition the policy and successor feature estimator. For a given transition (s, a, s'), the successor features are trained via Bellman consistency,

$$\boldsymbol{\psi}(s, a, \mathbf{w}) = \boldsymbol{\phi}(s') + \gamma\, \mathbb{E}_{a' \sim \pi_{\mathbf{w}}(\cdot|s')} \boldsymbol{\psi}(s', a', \mathbf{w}), \tag{4}$$

yielding optimal action-values $q^*_{\mathbf{w}}(s, a) = \mathbf{w}^\top \boldsymbol{\psi}(s, a, \mathbf{w})$ for any reward in the span of the Laplacian eigenvectors (Borsa et al., 2019). In continuous action spaces, π is optimized via policy gradients; in discrete spaces, actions are selected greedily as $\arg\max_a \mathbf{w}^\top \boldsymbol{\psi}(s, a, \mathbf{w})$ (Borsa et al., 2019).

Following Touati et al. (2023), we sample weight vectors $\mathbf{w}$ from a broad distribution during USFA training to promote wide coverage of the reward-function space. For any reward function in the

[2] In practice the encoder outputs $k + 1$ eigenfunctions; the first, the trivial constant eigenvector, is discarded.

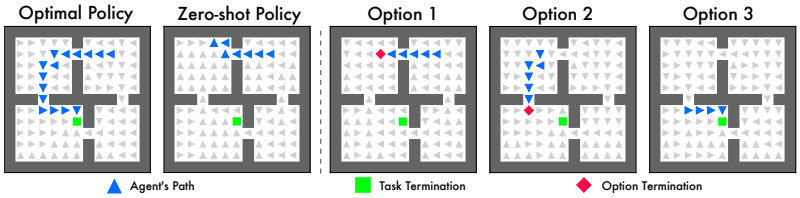


Figure 4: The LK stitches behaviors from the learned basis to approximate the optimal policy when the reward function lies outside the span of the learned representation. We use the same reward function r from Figure 2. The *first* panel contains the optimal policy of the original reward function to reach the goal state. The *second* panel displays the optimal policy of the reconstructed reward function with $k = 5$ eigenvectors (r_5). This policy fails to reach the goal state. The *last three* panels show how LK stitches three individual options, each terminated after a fixed horizon $t_{\text{term}} = 6$. Together, these options form a near-optimal policy that replicates the optimal behavior.

eigenvector span, the USFA outputs the optimal policy; for reward functions outside this span, the USFA instead optimizes the projected reward $\hat{\mathbf{r}}_k$, and Theorem 3.1 bounds how far the resulting optimal value can deviate from that of the true task. This bound highlights a fundamental limitation: for any fixed truncated basis, some reward functions remain outside the span.

4.2 Downstream Phase

The Laplacian Keyboard is designed precisely to overcome the limitation of a fixed linear span. Rather than committing to a single fixed behavior at execution time, LK trains a meta-policy that adapts the behavior basis to the demands of the downstream tasks. Prior option-based methods that pursue a similar goal typically rely on manually specified reward features or predefined training tasks—limiting their generality to settings where such supervision is available. LK avoids these requirements entirely: the behavior basis is learned directly from reward-free offline data, scales with its coverage and the chosen value of k, and is not restricted to a finite set of options.

Concretely, the USFA provides a continuous library of option policies parameterized by $\mathbf{w} \in \mathbb{R}^k$, and a meta-policy $\pi_{\text{LK}} : \mathcal{S} \to \mathbb{R}^k$ is trained to sequence these options to solve arbitrary downstream tasks. The hierarchical control mechanism is illustrated in Figure 3: π_{LK} outputs a weight vector $\mathbf{w}$ that instantiates an option via the USFA; this option executes for a fixed horizon t_{term}; control then returns to π_{LK}, which selects a new weight vector based on the updated state. This process repeats until the task is solved.

When the downstream reward function lies within the span of the learned representation, the optimal weight vector $\mathbf{w}$ satisfying $r(s) = \mathbf{w}^\top \boldsymbol{\phi}(s)$ lies within the meta-policy's action space, so the meta-policy can recover the zero-shot solution by selecting it at every decision point. For rewards outside this span, the meta-policy can instead compose multiple options—each optimal for its respective weight vector—into a policy that no single weight vector can express, as illustrated in Figure 4. This compositional structure is precisely what enables LK to address reward functions that lie beyond the linear span of the learned representation.

We also expect LK to improve sample efficiency over flat agents—those acting directly in the primitive action space—for two reasons. (1) By learning over options rather than primitive actions, LK propagates value information across multiple timesteps, which can accelerate credit assignment (Sutton et al., 1999). (2) Exploration in this structured space corresponds to exploring over coherent, temporally-extended behaviors (Machado et al., 2017, 2023; Jinnai et al., 2019), which can reduce variance and enable faster downstream learning. Crucially, the action space of π_{LK} is the continuous space of weight vectors $\mathbf{w}$—a smooth control space in which small changes in the policy output lead to similar behaviors, rather than qualitatively different action sequences.

5 Empirical Validation on DMC

We empirically show that the Laplacian representations enable zero-shot RL and that LK's hierarchical structure enables sample efficient improvements beyond the limits of linear combinations.

Table 1: Zero-shot performance of FB versus Laplacian Basis for $k = 50$ (mean $\pm$ standard error, $n = 30$) on the APS dataset.

Task	FB	Laplacian
CHEETAH *(avg)*	622	591
`Run`	329 $\pm$17	265 $\pm$13
`Run-B`	281 $\pm$8	297 $\pm$13
`Walk`	899 $\pm$34	848 $\pm$30
`Walk-B`	981 $\pm$3	953 $\pm$24
QUADRUPED *(avg)*	612	597
`Jump`	625 $\pm$12	674 $\pm$22
`Run`	428 $\pm$10	425 $\pm$10
`Stand`	799 $\pm$15	817 $\pm$23
`Walk`	595 $\pm$12	470 $\pm$12
WALKER *(avg)*	484	588
`Flip`	211 $\pm$20	604 $\pm$21
`Run`	266 $\pm$16	327 $\pm$21
`Stand`	660 $\pm$30	589 $\pm$45
`Walk`	798 $\pm$15	833 $\pm$30

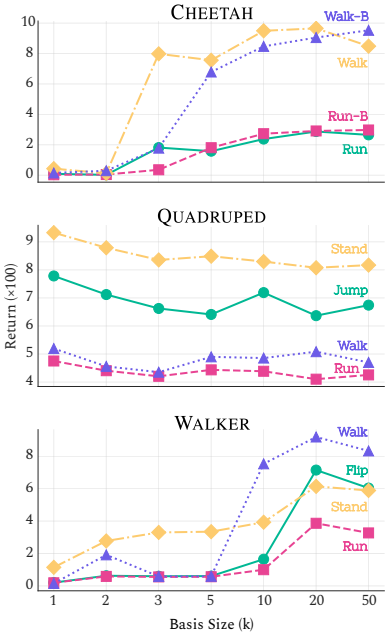


Figure 5: Zero-shot performance of the Laplacian Basis for varying basis sizes.

Environment: We evaluate the LK on three DMC domains (Tassa et al., 2018): CHEETAH (`Run, Run-B, Walk, Walk-B`), QUADRUPED (`Jump, Run, Stand, Walk`), and WALKER (`Flip, Run, Stand, Walk`), where `-B` denotes backward variants. Each domain provides 4 tasks and offline datasets collected under three exploration policies—`APS`, `Proto`, and `RND` (Yarats et al., 2022)—yielding 36 task-dataset combinations across diverse data-collection strategies.

Implementation: For each domain–dataset pair and basis size k, we train a Laplacian encoder and a corresponding USFA, yielding 9 pre-trained agents per basis size used for zero-shot evaluation. A meta-policy is then trained for each downstream task, giving 36 meta-policies per basis size (9 domain–dataset pairs $\times$ 4 tasks). Across all basis sizes $k \in \{1, 2, 3, 5, 10, 20, 50\}$, this amounts to 252 settings in total. Both the USFA and meta-policies are trained with TD3 (Fujimoto et al., 2018). Following Touati et al. (2023), zero-shot performance is evaluated using 10K labeled samples; each meta-policy is trained for 200K environment interactions, evaluated every 10K steps over 10 episodes. Options are terminated after $t_{\text{term}} = 5$ steps (see Appendix F for an ablation). All results are averaged over 30 independent runs.

5.1 Linear Span of the Laplacian Basis

We first establish the *zero-shot performance* of the Laplacian basis as a foundation for the hierarchical experiments that follow. To calibrate its strength, we compare against Forward-Backward (FB) representations—a strong baseline (Touati et al., 2023) with theoretical guarantees (Blier et al., 2021)—not to claim state-of-the-art zero-shot performance, but to confirm that the Laplacian basis is a credible starting point. Table 1 reports results at $k = 50$ using the `APS` dataset (additional results in Appendix D.2); we focus on this basis size following Touati et al. (2023), where FB performs best and performance is shown to plateau beyond it. A sign test finds no significant difference between the two methods, confirming the Laplacian basis matches an established alternative in zero-shot performance.

Yet this strong performance does not simply scale with basis size—and this limitation is not unique to the Laplacian basis; FB exhibits the same saturation behavior (Touati et al., 2023). Figure 5 shows that while CHEETAH and WALKER benefit modestly from larger k, gains saturate somewhere between $k = 20$ and $k = 50$. More strikingly, QUADRUPED achieves its best performance at $k = 1$: the leading eigenvector, which captures torso-height regulation central to all tasks, already contains most task-relevant structure, and adding components only makes the representation harder to learn and the downstream USFA harder to optimize (see Appendix E). Importantly, this saturation does not reflect

Table 2: Relative improvement of LK over the zero-shot performance (*left*) and ratio of area under the evaluation curve (AUC) over the first 200k environment steps (*right*), across basis sizes k. Results for the APS dataset averaged over 30 runs.

Task	Improv. over zero-shot (%)							Downstream AUC ratio						
	1	2	3	5	10	20	50	1	2	3	5	10	20	50
CHEETAH *(avg)*	14701	11496	219	98	15	−3	0	0.6	0.5	1.2	0.9	1.3	1.4	1.3
Run	9143	5474	34	−21	17	15	20	0.5	0.6	1.6	0.8	1.7	2.0	2.0
Run-B	25129	21084	497	16	31	−22	−6	0.4	0.3	0.5	0.7	0.7	0.6	0.7
Walk	1956	4169	20	−25	−7	−2	7	0.8	0.8	2.1	1.1	1.9	2.0	1.8
Walk-B	22575	15258	325	424	20	−4	−20	0.7	0.4	0.7	0.9	1.0	0.8	0.9
QUADRUPED *(avg)*	0	−11	−8	4	3	23	26	3.3	2.6	2.5	3.0	2.9	3.4	3.5
Jump	0	−18	−15	4	−15	21	16	3.7	2.6	2.6	3.0	2.7	3.4	3.5
Run	0	−11	−10	1	2	20	14	3.3	2.7	2.5	3.0	2.9	3.3	3.2
Stand	0	−21	−15	−4	−1	17	15	3.3	2.4	2.4	2.8	2.7	3.2	3.2
Walk	1	6	9	18	25	34	59	3.1	2.8	2.5	3.2	3.2	3.7	3.9
WALKER *(avg)*	1399	341	393	444	447	59	72	0.9	1.1	1.0	1.1	2.1	2.3	2.0
Flip	1299	451	492	556	894	54	52	0.9	1.0	1.0	1.1	2.5	2.6	2.0
Run	682	254	217	255	504	73	92	0.8	1.1	0.9	1.0	2.4	2.7	2.4
Stand	378	172	88	102	157	101	86	0.8	0.9	0.9	0.9	1.4	1.5	1.3
Walk	3237	488	774	863	234	6	60	1.0	1.3	1.1	1.2	2.3	2.3	2.2

an optimality ceiling—a task-specific agent trained from scratch on individual tasks still achieves substantially higher performance, indicating that headroom remains. *Taken together, the Laplacian basis is a strong zero-shot foundation, yet simply scaling the basis size yields no consistent gains.* The next section shows how LK overcomes this ceiling through hierarchical composition.

5.2 Beyond the Linear Span of the Laplacian Basis

In this section, we ask two questions about LK's hierarchical structure: (1) does it enable policies that go beyond the linear span of the Laplacian basis? and (2) does it improve sample efficiency relative to flat agents?

Having established that $k = 20$ and $k = 50$ are near-optimal for zero-shot performance, we focus our analysis on $k \in \{1, 2, 3, 5, 10\}$—the regime where the zero-shot solution alone is insufficient—and report $k = 20$ and $k = 50$ for reference. This directly probes question (1): if LK improves in this regime, the gain cannot be attributed solely to a strong zero-shot solution; it must arise from downstream use of the behavior library. To probe question (2), we train a task-specific flat TD3 agent as a strong task-specific reference. We note that a constrained basis in DMC proxies the general setting: a basis insufficient in a simple domain is analogous to a much larger basis that remains insufficient in a more complex one.

Table 2 (left) answers question (1), reporting the percentage improvement of LK over its zero-shot performance after 200K environment steps. In CHEETAH and WALKER, LK improves on most tasks across all basis sizes, with the largest gains at small k. The result is most striking at $k = 1$, where the USFA represents only two options (the positive and negative directions of the single eigenvector) and no linear combinations are possible. Yet, switching between just these two options yields improvements exceeding 100% on several tasks, confirming that hierarchy does real work beyond what the basis alone affords. As k increases, improvement magnitudes decrease but remain largely positive. QUADRUPED is the exception: tasks here are already well-approximated by the leading eigenvector, so hierarchical learning degenerates to finding a single weight vector in $\mathbb{R}^k$—a difficult optimization with little to gain. We note that LK can, in principle, recover the zero-shot solution if rewarded offline transitions are available, guaranteeing performance no worse than the zero-shot baseline. The results here, however, are achieved without that assumption—LK learns without any privileged reward information.

Table 2 (right) answers question (2), reporting the ratio of LK's AUC to TD3's AUC over the first 200K steps. A ratio above 1 means LK accumulates more return than TD3 over the same budget. In CHEETAH and WALKER, the ratio exceeds 1 for $k \geq 3$ across most tasks, indicating that even a modestly expressive basis is sufficient for LK to be more sample efficient than the flat agent. The effect is strongest in QUADRUPED, where ratios reach at least $2.4\times$ across all basis sizes—here,

the Laplacian basis already captures most task-relevant structure, giving the meta-policy a strong initialization to build on while TD3 starts from scratch.

Taken together, both questions are answered affirmatively. LK goes beyond what the linear basis can express. Even with a severely constrained basis, hierarchical composition improves over the zero-shot solution where scaling alone fails. And across basis sizes, LK accumulates returns faster than a flat agent learning from scratch (learning curves in Appendix D.3), all without the privileged reward information the zero-shot solution requires.

6 Comparison Against Privileged Baseline

As a final empirical evaluation, we conduct a controlled case study comparing LK to the Option Keyboard Basis (OKB; Alegre et al., 2025), a method that constructs its option basis from environment-provided reward features spanning downstream tasks, with theoretical guarantees of recovering a basis sufficient for optimal control. This comparison directly asks: can LK achieve competitive performance without access to such privileged, handcrafted reward information?

We evaluate LK on the ITEM-COLLECTOR domain—the setting Alegre et al. (2025) used to validate their proposed method. It is a 10×10 toroidal gridworld where the agent must collect items in a fixed order. OKB requires explicit, handcrafted reward features to construct its option basis; LK, by contrast, requires no such privileged information and is pre-trained solely on a reward-free dataset collected via a uniform random walk. We consider two LK variants that differ in how they compose options; in both, the meta-policy outputs a weight vector $\mathbf{w} \in \mathbb{R}^k$. *LK-USFA* executes the corresponding USFA policy $\pi_{\mathbf{w}}$ directly, while *LK-GPE&GPI* uses $\mathbf{w}$ to combine a fixed library of $2k$ basis policies—the USFA policies conditioned on $\mathbf{w}_{\pm \mathbf{e}_i}$—by evaluating each under $\mathbf{w}$ (GPE) and then acting greedily with respect to the best of these evaluations (GPI) (Barreto et al., 2017). This ablation isolates the contribution of continuous composition over discrete composition (additional details in Appendix G).

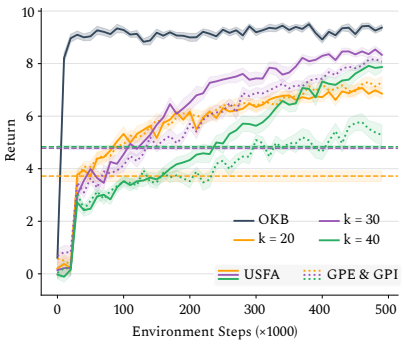


Figure 6: Evaluation returns for LK (varying k) and OKB, averaged over 30 runs (shaded: standard error). Dashed horizontal lines indicate the zero-shot performance.

Figure 6 shows that despite having no access to reward features, LK-USFA achieves performance close to OKB, with the gap narrowing as training progresses. Notably, all LK variants improve over their respective zero-shot performance, confirming that hierarchy adds value beyond the linear basis. LK-GPE&GPI lags behind LK-USFA at larger basis sizes, with the gap widening as k increases—a consequence of compounding approximation error in the two-step pipeline. Together, the results show that strong downstream performance does not require reward-feature engineering: Laplacian eigenvectors form a scalable behavior basis, and continuous composition is the more effective meta-policy parameterization.

7 Conclusion

This work makes two contributions. Our primary contribution is the *Laplacian Keyboard*, a hierarchical framework that composes Laplacian-induced behaviors via a learned meta-policy, allowing the agent to go beyond the linear span of the representation. This compositional structure is important in large environments, where no fixed finite basis should be expected to cover the full diversity of downstream tasks (Javed and Sutton, 2024). Rather than treating the basis as a complete solution, LK treats it as a structured starting point—yielding improvements over zero-shot baselines while maintaining strong sample efficiency relative to standard RL agents. We complement this with our second contribution: an analysis of Laplacian eigenvectors as a reward basis, providing empirical validation and theoretical value-function approximation error bounds that characterize the expressivity of a finite Laplacian basis.

Several directions follow naturally from this work. The *Keyboard* framework is not specific to Laplacian representations: any behavior basis, such as HILP (Park et al., 2024) or FB (Touati et al., 2023), could serve as a drop-in replacement, giving rise to a family of *Keyboard* agents. The Laplacian representation, and many others, are global, and representing localized rewards requires many components. Multi-scale representations such as diffusion wavelets (Mahadevan and Maggioni, 2005) could express such rewards more compactly. A fully online formulation is another natural extension, integrating representation-driven option discovery (Klissarov and Machado, 2023; Machado et al., 2023; Machado, 2025) with LK to remove the dependence on an offline dataset. Finally, applying LK to goal-reaching tasks requires effective reward shaping, which could leverage these learned representations (Wu et al., 2019).

Two limitations are worth noting. Options are currently terminated after a fixed horizon rather than through principled termination conditions, which we expect to limit option quality. Additionally, while LK can in principle recover the zero-shot solution—and therefore should perform no worse—empirical results are mixed in a few cases, which we attribute to exploration. The zero-shot weights lie within the meta-policy's search space but are not reliably found within the downstream budget. Warm-starting the meta-policy from the zero-shot weights, or shaping its exploration toward them, could potentially restore the guarantee while preserving LK's capacity to improve upon it—an important direction for future work.

Acknowledgements

We thank Diego Gomez, Jesse Farebrother and Pranaya Jajoo for helpful discussions, and Lucas N. Alegre for sharing the OKB code. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chair Program, and Alberta Innovates. It was also enabled in part by computational resources provided by the Digital Research Alliance of Canada.

References

Joseph Fourier. *The Analytical Theory of Heat*. Cambridge University Press, 1878.

Sridhar Mahadevan. Proto-Value Functions: Developmental Reinforcement Learning. In *International Conference on Machine Learning*, 2005.

Marc G. Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A Geometric Perspective on Optimal Representations for Reinforcement Learning. In *Neural Information Processing Systems*, 2019.

Jesse Farebrother, Joshua Greaves, Rishabh Agarwal, Charline Le Lan, Ross Goroshin, Pablo Samuel Castro, and Marc G. Bellemare. Proto-Value Networks: Scaling Representation Learning with Auxiliary Tasks. In *International Conference on Learning Representations*, 2023.

Siddhant Agarwal, Harshit Sikchi, Peter Stone, and Amy Zhang. Proto Successor Measure: Representing the Behavior Space of an RL Agent. In *International Conference on Machine Learning*, 2025a.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, and Doina Precup. The Option Keyboard: Combining Skills in Reinforcement Learning. In *Neural Information Processing Systems*, 2019.

Lucas Nunes Alegre, Ana L. C. Bazzan, André Barreto, and Bruno C. da Silva. Constructing an Optimal Behavior Basis for the Option Keyboard. In *Neural Information Processing Systems*, 2025.

Léonard Blier, Corentin Tallec, and Yann Ollivier. Learning Successor States and Goal-Dependent Values: A Mathematical Viewpoint. *arXiv:2101.07123*, 2021.

Ahmed Touati, Jérémy Rapin, and Yann Ollivier. Does Zero-Shot Reinforcement Learning Exist? In *International Conference on Learning Representations*, 2023.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, Hado van Hasselt, and David Silver. Successor Features for Transfer in Reinforcement Learning. In *Neural Information Processing Systems*, 2017.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *International Conference on Machine Learning*, 2017.

Marlos C. Machado, Clemens Rosenbaum, Xiaoxiao Guo, Miao Liu, Gerald Tesauro, and Murray Campbell. Eigenoption Discovery through the Deep Successor Representation. In *International Conference on Learning Representations*, 2018.

Marlos C. Machado, André Barreto, Doina Precup, and Michael Bowling. Temporal Abstraction in Reinforcement Learning with the Successor Representation. *Journal of Machine Learning Research*, 24(80):1–69, 2023.

Diana Borsa, André 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*, 2019.

Yann Ollivier. Which Features are Best for Successor Features? *arXiv:2502.10790*, 2025.

Andrea Tirinzoni, Ahmed Touati, Jesse Farebrother, Mateusz Guzek, Anssi Kanervisto, Yingchen Xu, Alessandro Lazaric, and Matteo Pirotta. Zero-shot Whole-body Humanoid Control via Behavioral Foundation Models. In *International Conference on Learning Representations*, 2025.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast Reinforcement Learning with Generalized Policy Updates. *Proceedings of the National Academy of Sciences*, 117 (48):30079–30087, 2020.

Siddarth Chandrasekar and Marlos C. Machado. Towards An Option Basis To Optimize All Rewards. In *Workshop on Reinforcement Learning Beyond Rewards@ Reinforcement Learning Conference*, 2025.

Xiaofan Zhu and Michael Rabbat. Approximating Signals Supported on Graphs. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

Diego Gomez, Michael Bowling, and Marlos C. Machado. Proper Laplacian Representation Learning. In *International Conference on Learning Representations*, 2024.

Peter Dayan. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624, 1993.

Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering Options for Exploration by Minimizing Cover Time. In *International Conference on Machine Learning*, 2019.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. *arXiv:1801.00690*, 2018.

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:2201.13425*, 2022.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, 2018.

Khurram Javed and Richard S. Sutton. The Big World Hypothesis and Its Ramifications for Artificial Intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024.

Seohong Park, Tobias Kreiman, and Sergey Levine. Foundation Policies with Hilbert Representations. In *International Conference on Machine Learning*, 2024.

Sridhar Mahadevan and Mauro Maggioni. Value Function Approximation with Diffusion Wavelets and Laplacian Eigenfunctions. In *Neural Information Processing Systems*, 2005.

Martin Klissarov and Marlos C. Machado. Deep Laplacian-Based Options for Temporally-Extended Exploration. In *International Conference on Machine Learning*, 2023.

Marlos C. Machado. Representation-Driven Option Discovery in Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2025.

Yifan Wu, George Tucker, and Ofir Nachum. The Laplacian in RL: Learning Representations with Efficient Approximations. In *International Conference on Learning Representations*, 2019.

Pranaya Jajoo, Harshit Sikchi, Siddhant Agarwal, Amy Zhang, Scott Niekum, and Martha White. Regularized Latent Dynamics Prediction is a Strong Baseline for Behavioral Foundation Models. In *International Conference on Learning Representations*, 2026.

Marco Bagatella, Matteo Pirotta, Ahmed Touati, Alessandro Lazaric, and Andrea Tirinzoni. TD-JEPA: Latent-predictive Representations for Zero-Shot Reinforcement Learning. In *International Conference on Learning Representations*, 2026.

Siddhant Agarwal, Caleb Chuck, Harshit Sikchi, Jiaheng Hu, Max Rudolph, Scott Niekum, Peter Stone, and Amy Zhang. A Unified Framework for Unsupervised Reinforcement Learning Algorithms. In *Workshop on Reinforcement Learning Beyond Rewards@ Reinforcement Learning Conference*, 2025b.

Harshit Sikchi, Andrea Tirinzoni, Ahmed Touati, Yingchen Xu, Anssi Kanervisto, Scott Niekum, Amy Zhang, Alessandro Lazaric, and Matteo Pirotta. Fast adaptation with behavioral foundation models. *Reinforcement Learning Journal*, 6:1048–1074, 2025.

Safa Alver and Doina Precup. Constructing a Good Behavior Basis for Transfer Using Generalized Policy Updates. In *International Conference on Learning Representations*, 2022.

Tom Zahavy, André Barreto, Daniel J. Mankowitz, Shaobo Hou, Brendan O'Donoghue, Iurii Kemaev, and Satinder Singh. Discovering a Set of Policies for the Worst Case Reward. In *International Conference on Learning Representations*, 2021.

Wilka Carvalho, Andre Saraiva, Angelos Filos, Andrew Lampinen, Loic Matthey, Richard L. Lewis, Honglak Lee, Satinder Singh, Danilo Jimenez Rezende, and Daniel Zoran. Combining Behaviors with the Successor Features Keyboard. In *Neural Information Processing Systems*, 2023.

David I. Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-Frequency Analysis on Graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.

Thomas Rupf, Marco Bagatella, Marin Vlastelica, and Andreas Krause. Optimistic Task Inference for Behavior Foundation Models. In *International Conference on Learning Representations*, 2026.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-Quality Single-File Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

A Extended Related Work

The Laplacian Keyboard bridges representation-based and behavior-based solution methods to address the *zero-shot* RL problem. In this section, we review closely related work along both directions, extending the discussion beyond that provided in the main paper.

Representation-based Methods: A common approach to zero-shot RL is to learn task-agnostic state representations that can be reused for downstream control. Examples include Hilbert Foundation Policies (HILP; Park et al., 2024), Proto Successor Measures (PSM; Agarwal et al., 2025a), Regularized Latent Dynamics Prediction (RLDP; Jajoo et al., 2026), and TD-JEPA (Bagatella et al., 2026). While these methods differ in their objectives, they all aim to construct representations that support policy transfer without additional environment interaction.

HILP learns representations sufficient for expressing goal-reaching value functions and uses them to construct intrinsic reward functions for training a latent-conditioned policy. PSM learns a basis over state visitation distributions, enabling downstream behaviors to be synthesized as linear combinations of these bases and supporting zero-shot policy construction via reward-dependent recombination. RLDP and TD-JEPA learn predictive latent representations by modeling future states while regularizing against feature collapse; these representations are then paired with USFA to enable zero-shot transfer. A unified treatment of several representation-based methods is provided by Agarwal et al. (2025b).

Closely related in spirit to LK are the LoLa and ReLa algorithms proposed by Sikchi et al. (2025), which also exploit pre-trained knowledge to improve downstream task performance. However, unlike LK, these methods are not hierarchical in nature and are primarily motivated by reducing approximation errors arising during pre-training and inference. As a result, they are better characterized as fine-tuning or correction mechanisms rather than frameworks for skill composition.

Behavior-based Methods: Set of Independent Policies (SIP; Alver and Precup, 2022), Set Max Policies (SMP; Zahavy et al., 2021), and the Successor Feature Keyboard (SFK; Carvalho et al., 2023) are few examples that focus on learning a reusable set of behaviors or policies to enable zero-shot control.

SIP assumes access to a set of independent reward features and learns a policy for each feature. At test time, GPE & GPI are used to recover the optimal policy for any reward expressed as a linear combination of these features. SMP retains the linear reward assumption but optimizes for robustness by learning policies that maximize worst-case performance over admissible rewards; zero-shot transfer is again achieved using GPE & GPI. SFK relaxes the assumption of known reward features and learns a policy basis by extracting shared structure across a set of training tasks.

A unifying backbone of these methods is the successor representation (SR; Dayan, 1993) and its generalization, successor features (SF; Barreto et al., 2017). These frameworks decompose the value function into a dynamics-dependent component, capturing long-term state visitation, and a task-specific component defined by the reward. SR represents each state by its expected discounted future state occupancies under a policy, while SF extends this idea to feature spaces by modeling expected discounted feature activations. Under the assumption of linear reward functions, this decomposition allows value functions to be computed as inner products between SF and reward weights, naturally enabling zero-shot transfer via GPE & GPI and USFA.

B Laplacian Basis

In this section, we provide the proof for Theorem 3.1. We first provide the necessary background, followed by the proof. We reintroduce a few details for completeness.

B.1 Background

The proof and theorem are partly inspired by the spectral graph theory literature. Hence we first introduce concepts in the language of graph theory before using them for RL.

B.1.1 Spectral Theory

Graph Laplacian: Consider an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Each edge $(i, j) \in \mathcal{E}$ carries a non-negative weight $w_{ij} \geq 0$ quantifying the connection strength between vertices i and j. These weights define the weighted adjacency matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $\mathbf{W}_{ij} = w_{ij}$ for $(i, j) \in \mathcal{E}$ and $\mathbf{W}_{ij} = 0$ otherwise. The vertex degrees $d_i = \sum_j w_{ij}$ form the degree matrix $\mathbf{D} = \mathrm{diag}(d_1, d_2, \ldots, d_{|\mathcal{V}|})$. For undirected graphs, the adjacency matrix exhibits symmetry: $w_{ij} = w_{ji} \, \forall (i, j) \in \mathcal{E}$.

The *graph Laplacian matrix* is defined as $\mathbf{L} \doteq \mathbf{D} - \mathbf{W}$, a real symmetric positive semi-definite matrix admitting the eigendecomposition $\mathbf{L} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^\top$. Here, $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{|\mathcal{V}|}]$ contains orthonormal eigenvectors and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{|\mathcal{V}|})$ contains the corresponding non-negative eigenvalues ordered such that $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{|\mathcal{V}|}$. These eigenvectors encode structural properties of the graph in an ordered fashion: eigenvectors associated with smaller eigenvalues (low frequencies) exhibit smooth variations across the graph, while those associated with larger eigenvalues (high frequencies) capture rapid oscillations between neighboring vertices.

Graph Fourier Transform: Any graph signal $f : \mathcal{V} \to \mathbb{R}$ defined over the vertices admits an exact decomposition as a linear combination of the Laplacian eigenvectors:

$$\mathbf{f} = \sum_{i=1}^{|\mathcal{V}|} \tilde{f}_i \mathbf{e}_i, \tag{5}$$

where the *Graph Fourier coefficients* $\tilde{f}_i = \langle \mathbf{f}, \mathbf{e}_i \rangle$ represent the projection of $\mathbf{f}$ onto the i-th eigenvector. The *Graph Fourier Transform* (GFT) $\tilde{\mathbf{f}} = \mathbf{E}^\top \mathbf{f}$ maps signals from the vertex domain to the spectral domain, where $\tilde{\mathbf{f}} = [\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_{|\mathcal{V}|}]^\top$ denotes the signal's spectral representation. This transformation serves as the graph analog of the classical Fourier transform, decomposing arbitrary graph signals into fundamental frequency components. Signals dominated by large coefficients $\tilde{f}_i$ for low i vary smoothly across the graph topology, while those with significant high-index coefficients exhibit complex, irregular patterns that vary sharply between connected vertices.

A fundamental property of the GFT is energy preservation across domains, formalized by Parseval's theorem Shuman et al. (2016):

Lemma B.1 (Parseval's Theorem)**.** *The energy of a signal in the vertex domain equals its energy in the spectral domain:*

$$\sum_{i=1}^{|\mathcal{V}|} f(i)^2 = \sum_{i=1}^{|\mathcal{V}|} \tilde{f}_i^2. \tag{6}$$

This property allows us to measure energy of the signal as a function of the Graph Fourier coefficients.

Truncated Spectral Approximation: For large graphs where $|\mathcal{V}|$ is large, computing and storing the full basis becomes computationally prohibitive. This constraint requires restricting the representation to a *truncated spectral basis* consisting of the first $k \ll |\mathcal{V}|$ eigenvectors corresponding to the smallest eigenvalues. The resulting approximation

$$\hat{\mathbf{f}}_k = \sum_{i=1}^{k} \tilde{f}_i \mathbf{e}_i \tag{7}$$

captures the signal's low-frequency components—variations that are smooth with respect to the graph structure. This dimensionality reduction induces an inductive bias toward smoother signals that vary gradually across connected vertices while filtering out high-frequency fluctuations. The truncation sacrifices exact reconstruction in favor of computational efficiency.

The approximation error induced by truncation can be quantified through Parseval's theorem. Since the omitted eigenvectors correspond to the high-frequency components, the squared reconstruction error is given by

$$\text{error}(f, k) \doteq \sum_{i=k+1}^{|\mathcal{V}|} \tilde{f}_i^2, \tag{8}$$

representing the energy concentrated in the discarded frequencies. This quantity measures the information loss incurred by restricting the reconstruction space to the subspace spanned by the first k eigenvectors. As k increases, the approximation error decreases, tightening the reconstruction quality.

Graph Signal Smoothness: To formalize the notion of smoothness for signals defined over graphs, we rely on the *graph total variation*, which quantifies local variations weighted by edge connectivity (Zhu and Rabbat, 2012).

Definition B.2 (Graph Total Variation)**.** For any graph signal $f : \mathcal{V} \to \mathbb{R}$, the graph total variation is defined as

$$\|f\|_G \doteq \left[\frac{1}{2} \sum_{(i,j)\in\mathcal{E}} w_{ij} (f(i) - f(j))^2 \right]^{\frac{1}{2}}. \tag{9}$$

The graph total variation captures the differences between signal values at connected vertices, weighted by their edge strengths. Smaller values of $\|f\|_G$ indicate that the signal varies smoothly across connected vertices. The factor $\frac{1}{2}$ accounts for double-counting in undirected graphs; Zhu and Rabbat (2012) omit this term by enumerating each edge exactly once.

A signal f exhibits *bounded variation* if there exists a constant $0 < C \ll \lambda_{|\mathcal{V}|}$ satisfying

$$\|f\|_G^2 \leq C \cdot \|f\|^2, \tag{10}$$

where $\|f\|$ denotes the standard ℓ_2 norm of f viewed as a vector over the vertices. This condition ensures that the signal exhibits controlled variability with respect to the underlying graph topology. The reconstruction error induced by truncation can be bounded using this smoothness measure,

Lemma B.3 (Reconstruction Bound Zhu and Rabbat (2012))**.** *For any signal f with bounded variation, the squared reconstruction error using the first k eigenvectors satisfies*

$$\text{error}(f, k) \leq \|f\|_G^2 \cdot \lambda_{k+1}^{-1}. \tag{11}$$

The bound reveals two key factors influencing approximation quality. (1) Signals that vary smoothly over the graph structure, reflected by smaller $\|f\|_G$, incur smaller errors. (2) Increasing the basis size improves reconstruction: as k grows, the corresponding eigenvalue λ_k is non-decreasing, tightening the bound. Together, they characterize how signal smoothness and basis dimensionality jointly determine reconstruction accuracy.

B.1.2 Laplacian for RL

The spectral framework introduced above extends naturally to RL.

Graph Laplacian for MDPs: An MDP under a fixed policy π induces a weighted graph $G = (\mathcal{S}, \mathcal{E})$ over the state space, where states form the vertex set and edges encode feasible transitions. Connectivity between states is governed by the transition matrix $\mathbf{P}_\pi$, defined entrywise as $\mathbf{P}_\pi(s, s') \doteq \sum_{a\in\mathcal{A}} \pi(a \mid s)\, p(s' \mid s, a)$. Since $\mathbf{P}_\pi$ is not symmetric in general, it cannot directly serve as the adjacency matrix of an undirected graph. Constructing the Laplacian therefore requires a symmetrization operator f_{sym} consistent with the dynamics, obtained by averaging the chain with its time reversal (Gomez et al., 2024). Letting ρ denote the stationary distribution of $\mathbf{P}_\pi$, the *time-reversed transition matrix* is given by $\mathbf{P}_\pi^* \doteq \text{diag}(\rho)^{-1} \mathbf{P}_\pi^\top \text{diag}(\rho)$, and the graph Laplacian is defined as

$$\boldsymbol{L} \doteq \boldsymbol{I} - f_{\text{sym}}(\mathbf{P}_\pi), \qquad f_{\text{sym}}(\mathbf{P}_\pi) \doteq \tfrac{1}{2}\left(\mathbf{P}_\pi + \mathbf{P}_\pi^*\right). \tag{12}$$

The entry $\mathbf{P}_\pi^*(s, s')$ represents the probability that the chain arrived at s from s' under stationarity, so f_{sym} assigns each edge a weight reflecting transitions in both directions. It serves two roles: it makes the chain reversible with respect to ρ, and it preserves row-stochasticity, so the degree matrix reduces to the identity and no explicit degree normalization is needed. While $\boldsymbol{L}$ is not symmetric as a matrix, it is self-adjoint under the ρ-weighted inner product. Consequently, it retains the spectral properties established above: its eigenvalues are real and non-negative, $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{|\mathcal{S}|}$, and its eigenvectors $\{\mathbf{e}_i\}_{i=1}^{|\mathcal{S}|}$ form an orthonormal basis, with respect to ρ, for functions over the state space. When ρ is uniform, $\mathbf{P}_\pi^* = \mathbf{P}_\pi^\top$, and f_{sym} reduces to $\frac{1}{2}(\mathbf{P}_\pi + \mathbf{P}_\pi^\top)$.

Spectral Approximation of Reward Functions: The primary graph signals of interest in RL are the reward function $r : \mathcal{S} \to \mathbb{R}$ and the value function $v : \mathcal{S} \to \mathbb{R}$. When these functions exhibit bounded variation with respect to the state space—meaning that states with high transition probability tend to have similar rewards or values—they can be efficiently approximated using a truncated Laplacian basis. In this work, we focus on the spectral representation of reward functions. Any reward function $r : \mathcal{S} \to \mathbb{R}$ admits the approximation

$$\hat{\mathbf{r}}_k = \sum_{i=1}^{k} \langle \mathbf{r}, \mathbf{e}_i \rangle \, \mathbf{e}_i = \sum_{i=1}^{k} \tilde{r}_i \, \mathbf{e}_i, \tag{13}$$

where the first k eigenvectors capture the low-frequency components of the reward signal. This truncated basis restricts the representable reward space to smooth functions that vary gradually across connected states, filtering out high-frequency reward structures that exhibit sharp variations in the state space.

B.2 Formal Proof

With the necessary background established, we now bound the change in the optimal value function caused by using an approximated reward function.

Theorem B.4 (Value Approximation Error Bound (restated))**.** *Let $r : \mathcal{S} \to \mathbb{R}$ be a reward function with bounded variation and $\hat{\mathbf{r}}_k$ its reconstruction using the first k eigenvectors. Let $\mathbf{v}^*$ and $\mathbf{v}_k^*$ be their corresponding optimal value functions. Then,*

$$\|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \frac{\|r\|_G}{(1-\gamma)\sqrt{\lambda_{k+1}}}, \tag{14}$$

where λ_{k+1} is the smallest eigenvalue of the graph Laplacian not included in the truncation.

Proof. We establish the bound in two stages: first, we bound the approximation error $\|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty$, then propagate this through the value function.

Step 1: Bounding the Signal Approximation Error.

Here, $\mathbf{r} - \hat{\mathbf{r}}_k$ is the residual reward function. By Parseval's theorem (Lemma B.1), the ℓ_2 error is

$$\|\mathbf{r} - \hat{\mathbf{r}}_k\|_2^2 = \sum_s (r(s) - \hat{r}_k(s))^2 = \sum_{i=k+1}^{|\mathcal{S}|} \tilde{r}_i^2, \tag{15}$$

where $\tilde{r}_i$ is the i-th Graph Fourier coefficient. Applying Lemma B.3, we obtain

$$\|\mathbf{r} - \hat{\mathbf{r}}_k\|_2^2 \leq \frac{\|r\|_G^2}{\lambda_{k+1}}. \tag{16}$$

Since the ℓ_∞ norm is bounded by the ℓ_2 norm, i.e., $\|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty^2 \leq \|\mathbf{r} - \hat{\mathbf{r}}_k\|_2^2$, we have

$$\|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty \leq \frac{\|r\|_G}{\sqrt{\lambda_{k+1}}}. \tag{17}$$

Step 2: Propagating Error Through the Value Function.

Let v^* and v_k^* be the optimal value functions of r and $\hat{r}_k$. By the Bellman optimality equation:

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r(s') + \gamma v^*(s') \right] \tag{18}$$

To bound the difference $\|v^* - v_k^*\|_\infty$, we start with the point-wise difference:

$$\begin{aligned} v^*(s) - v_k^*(s) &= \max_a \left[\sum_{s'} p(s'|s,a)(r(s') + \gamma v^*(s')) \right] - \max_a \left[\sum_{s'} p(s'|s,a)(\hat{r}_k(s') + \gamma v_k^*(s')) \right] \\ &\leq \max_a \left[\sum_{s'} p(s'|s,a) \left((r(s') - \hat{r}_k(s')) + \gamma (v^*(s') - v_k^*(s')) \right) \right] \end{aligned} \tag{19}$$

where we used the property $\max_a f(a) - \max_a g(a) \leq \max_a (f(a) - g(a))$.

Taking the supremum over $s \in \mathcal{S}$ on both sides:

$$\begin{aligned} \|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty &\leq \left\| \max_a \sum_{s'} p(s'|s,a) \left[(r(s') - \hat{r}_k(s')) + \gamma (v^*(s') - v_k^*(s')) \right] \right\|_\infty \\ &\leq \max_{s,a} \sum_{s'} p(s'|s,a) \, |r(s') - \hat{r}_k(s')| + \gamma \max_{s,a} \sum_{s'} p(s'|s,a) \, |v^*(s') - v_k^*(s')| \end{aligned} \tag{20}$$

Since $\sum_{s'} p(s'|s,a) = 1$, the weighted average is bounded by the maximum difference:

$$\sum_{s'} p(s'|s,a) |r(s') - \hat{r}_k(s')| \leq \|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty \quad \text{and} \quad \sum_{s'} p(s'|s,a) |v^*(s') - v_k^*(s')| \leq \|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \tag{21}$$

Substituting these back into our inequality, we obtain:

$$\|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty + \gamma \|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \tag{22}$$

Rearranging yields

$$(1-\gamma) \|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty. \tag{23}$$

$$\|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \frac{\|\mathbf{r} - \hat{\mathbf{r}}_k\|_\infty}{(1-\gamma)}. \tag{24}$$

Note that when the reward function lies in the span of the first k eigenvectors, exact reconstruction holds ($r = \hat{r}_k$), yielding zero error. Alternatively, this upper bound can be expressed in terms of the graph total variation. Substituting the bound from Equation 17:

$$\|\mathbf{v}^* - \mathbf{v}_k^*\|_\infty \leq \frac{\|r\|_G}{(1-\gamma)\sqrt{\lambda_{k+1}}}. \tag{25}$$

Finally, we note that the bound in Eqn. 25 is looser, but it offers a more direct interpretation than the bound in Eqn. 24. This udes the proof.

□

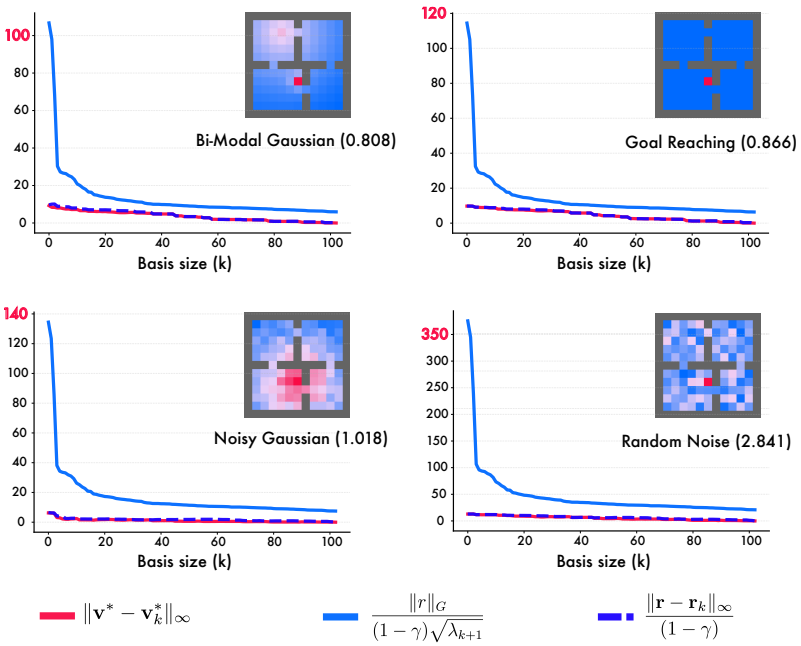


Figure 7: Reward reconstruction and induced value function error in `Four-Rooms`. For four reward functions of increasing complexity (shown insets), we plot the value error and two upper bounds (Equation 24 and 25) as a function of basis size k. The value in parentheses corresponds to the reward function's graph total variation. This environment has a total of 104 states.

B.3 Empirical Evaluation of Theorem 3.1

Figure 7 illustrates an empirical evaluation in the `Four-Rooms` domain using four simple reward functions. We measure the approximation error of the resulting value function as a function of the basis size used to reconstruct the reward. The reward associated with any transition (s, a, s') depends solely on the successor state s', and episodes terminate upon reaching the designated bright red states.

The basis is formed by eigenvectors induced by a uniform random policy, and value functions are computed via value iteration. Across all settings, the proposed bound constitutes a strict upper bound on the observed value approximation error. As the basis size increases, the gap between the bound and the empirical error consistently decreases. Additionally, reward functions that vary smoothly over the state space exhibit substantially lower approximation error than noisier reward functions, highlighting the bias induced by using a truncated basis.

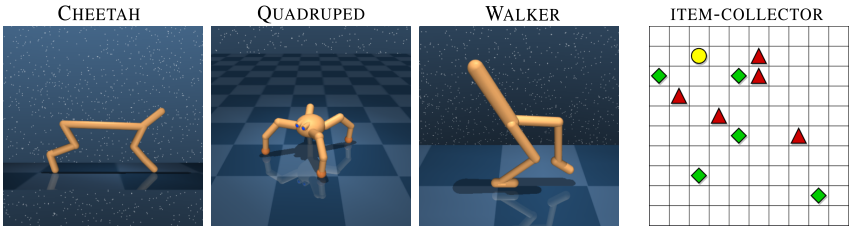


Figure 8: The 4 environments we use to empirically evaluate the Laplacian Keyboard.

C Experimental Details

We provide more experimental details in this section for clarity and reproducibility.

C.1 Environments

DeepMind Control (DMC) Suite (Tassa et al., 2018) comprises a variety of continuous control domains; in this work, we focus on three specific domains.

- CHEETAH is a bipedal agent actuated through a 6-dimensional continuous action space corresponding to joint torques. The observation space is 17-dimensional and consists of joint positions and joint velocities. In the `Walk` and `Run` tasks, the reward is a linear tolerance function of the forward torso velocity, encouraging the agent to exceed a task-specific speed threshold. The `Walk-Backward` and `Run-Backward` tasks are defined analogously by reversing the direction of the velocity term, thereby incentivizing backward locomotion.
- QUADRUPED is a four-legged agent actuated through a 12-dimensional continuous action space corresponding to joint torques. The observation space is 78-dimensional and consists of egocentric joint states, torso motion, inertial measurements, and force–torque readings at the feet. In the `Stand` task, the reward is a linear tolerance function encouraging an upright torso. The `Jump` task additionally rewards the agent for raising its center of mass above a target height while maintaining an upright posture. In the `Walk` and `Run` tasks, the reward is the product of the uprightness term and a forward-velocity term, incentivizing the agent to move at a task-specific target speed while remaining upright.
- WALKER is a planar bipedal agent actuated through a 6-dimensional continuous action space corresponding to joint torques. The observation space is 24-dimensional and consists of body orientations, torso height, and joint velocities. In the `Run` tasks, the reward is a linear tolerance function of the horizontal torso velocity, encouraging the agent to achieve a target forward or backward speed. In the `Flip` tasks, the reward is instead defined over the torso's angular momentum magnitude, incentivizing rapid rotational motion independent of translational velocity.

Across all DMC domains, tasks share the same underlying dynamics and differ only in their reward functions. Episodes have a fixed horizon of 1000 timesteps without early termination. We measure the undiscounted return, corresponding to setting $\gamma = 1$, as the performance metric.

ITEM-COLLECTOR is a discrete grid-world environment originally introduced by Barreto et al. (2019) and later used in Alegre et al. (2025) (Figure 8). The environment consists of a 10×10 toroidal grid and a discrete action space of four cardinal movements. At the beginning of each episode, five items of each of two distinct types are placed uniformly at random across the grid. The agent's objective is to collect all items of one type before collecting those of the other. Episodes terminate after 50 timesteps and admit a maximum undiscounted return of 10. The privileged reward features are two-dimensional and indicate which item type, if any, is collected at each transition. Performance is evaluated using the undiscounted return, corresponding to a discount factor of $\gamma = 1$.

C.2 Training Details

Laplacian Encoder: The Laplacian Encoder is trained using ALLO, the contrastive objective shown in Equation 26:

$$\max_{\boldsymbol{\beta}} \min_{\mathbf{u}\in\mathbb{R}^{k|\mathcal{S}|}} \underbrace{\sum_{i=1}^{k}\langle \mathbf{u}_i, \mathbf{L}\mathbf{u}_i\rangle}_{\text{smoothness}} + \underbrace{\sum_{j=1}^{k}\sum_{k=1}^{j}\Big[\beta_{jk}\big(\langle \mathbf{u}_j, [\![\mathbf{u}_k]\!]\rangle - \delta_{jk}\big) + b\big(\langle \mathbf{u}_j, [\![\mathbf{u}_k]\!]\rangle - \delta_{jk}\big)^2\Big]}_{\text{orthogonality constraints}} \tag{26}$$

Each training batch consists of a positive pair of states and a negative state. The positive pair is used to estimate the smoothness term, while the negative sample contributes to the orthogonality constraints. To construct a positive pair, we sample an episode and a reference state from that episode, and then sample a paired state according to a geometric distribution parameterized by γ_{allo}. The negative sample is drawn as another random state from the same episode. The Laplacian Encoder is trained via gradient descent and maps raw observations to a k-dimensional Laplacian representation.

Universal Successor Feature Approximators (USFA): USFA is trained as a value function, instantiated as the critic in a TD3 agent for continuous action spaces and as the Q-network in a Double DQN agent for discrete action spaces. We do not employ additional offline RL-specific techniques. To expose the USFA to a broad class of reward functions during pre-training, we sample weight vectors $\mathbf{w} \in \mathbb{R}^k$ following Touati et al. (2023).

1. Weight vectors sampled uniformly from the sphere of radius $\sqrt{k}$, corresponding to reward functions given by random linear combinations of the Laplacian eigenvectors.
2. Weight vectors obtained by sampling a random state from the offline dataset and using its Laplacian representation as $\mathbf{w}$, corresponding to a goal-reaching reward with the sampled state as the goal.

The actor takes the raw observation and the weight vector as input and outputs an action. The critic takes the raw observation, action, and weight vector as input and outputs the corresponding successor feature. Gradients from the USFA do not propagate into the Laplacian Encoder.

Zero-shot: After training the Laplacian encoder and the USFA, and given a downstream task, we randomly sample N transitions $\{(s_i, a_i, s_i')\}_{i=1}^{N}$ from the offline dataset and assign each transition its corresponding reward. The zero-shot weight vector is then estimated as

$$\mathbf{w} \;=\; \frac{1}{N}\sum_{i=1}^{N} r(s_i')\,\boldsymbol{\phi}(s_i'),$$

which coincides with the least-squares solution under the assumption that the learned feature matrix is full rank. This weight vector parameterizes the USFA and induces the corresponding zero-shot policy.

Computing the zero-shot weight vector requires access to rewarded transitions from the offline dataset—specifically, reward labels $r(s')$ for states uniformly sampled from $\mathcal{D}$. This is a non-trivial assumption: the agent cannot obtain such labels through environment interaction alone, as it would require querying rewards for arbitrary offline states. This privileged access is precisely what LK's hierarchical phase removes—the meta-policy learns to compose behaviors through online interaction, without any assumption on rewarded offline transitions. Recent work (Rupf et al., 2026) further explores how an agent can approach the *ideal* zero-shot performance through interaction alone, without relying on this assumption.

Meta-Policy: The meta-policy is trained online using TD3. At each decision point, it takes the raw observation as input and outputs a weight vector $\mathbf{w} \in \mathbb{R}^k$, which parameterizes the USFA to instantiate an option. The instantiated option then executes for a fixed horizon t_{term}, during which environment rewards are accumulated. The meta-policy is trained to maximize the discounted return $\sum_{t=0}^{t_{\text{term}}} \gamma^t r_t$. Exploration in the weight space is achieved by adding Gaussian noise to the output weight vectors. During meta-policy training, both the Laplacian Encoder and the USFA are kept fixed, and gradients do not propagate through either module.

Table 3: Hyperparameters for DMC.

Hyperparameter	Symbol	Value
Laplacian Encoder (ALLO)		
Total gradient steps	N_{allo}	10^6
Sampling discount factor	γ_{allo}	0.5
Step size	α_{allo}	10^{-4}
USFA Module (TD3-based)		
Total gradient steps	$N_{\text{usfa-c}}$	10^6
Gradient clipping coefficient	$\lambda_{\text{usfa-c}}$	0.001
Target network update coefficient	$\tau_{\text{usfa-c}}$	0.001
Policy update delay	$d_{\text{usfa-a}}$	1
Policy noise standard deviation	$\sigma_{\text{usfa-a}}$	0.0
Actor step size	$\alpha_{\text{usfa-a}}$	10^{-4}
Critic step size	$\alpha_{\text{usfa-c}}$	10^{-3}
Discount factor	γ_{usfa}	0.98
Meta-Agent (TD3-based)		
Samples for zero-shot task inference	N	10^4
Total gradient steps	N_{lk}	2×10^5
Batch size	B_{lk}	16
Target network update coefficient	τ_{lk}	0.01
Policy update delay	$d_{\text{lk-a}}$	5
Exploration noise standard deviation	$\epsilon_{\text{lk-a}}$	0.01
Actor step size	$\alpha_{\text{lk-a}}$	10^{-4}
Critic step size	$\alpha_{\text{lk-c}}$	10^{-3}
Option horizon	t_{term}	5
Discount factor	γ_{lk}	0.98

Table 4: Hyperparameters for ITEM-COLLECTOR

Hyperparameter	Symbol	Value
Laplacian Encoder (ALLO)		
Total gradient steps	N_{allo}	5×10^5
Sampling discount factor	γ_{allo}	0.1
Step size	α_{allo}	10^{-4}
USFA Module (DDQN-based)		
Total gradient steps	N_{usfa}	10^6
Gradient clipping coefficient	λ_{usfa}	0.01
Target network update coefficient	τ_{usfa}	0.001
Target network update frequency	d_{usfa}	1
Step size	α_{usfa}	10^{-4}
Discount factor	γ_{usfa}	0.95
Meta-Agent (TD3-based)		
Samples for zero-shot task inference	N	5×10^4
Total gradient steps	N_{lk}	5×10^5
Batch size	B_{lk}	32
Target network update coefficient	τ_{lk}	0.001
Policy update delay	$d_{\text{lk-a}}$	10
Exploration noise standard deviation	$\epsilon_{\text{lk-a}}$	0.1
Actor step size	$\alpha_{\text{lk-a}}$	10^{-4}
Critic step size	$\alpha_{\text{lk-c}}$	10^{-4}
Option horizon	t_{term}	5
Discount factor	γ_{lk}	0.95

C.3 Hyperparameter Selection:

In this section, we describe the hyperparameter selection procedure for LK on the DMC suite. For ITEM-COLLECTOR, hyperparameters were derived by loosely adapting the settings used for DMC.

Pre-training: The majority of our hyperparameters are adopted from prior work, specifically the work by Gomez et al. (2024) for the Laplacian encoder training and by Touati et al. (2023) for the USFA module training. During the pretraining phase, we observed improved training stability and final performance when using asymmetric step sizes for the USFA module: a smaller step size for the actor and a larger step size for the critic (SF estimator). We conducted a grid search over the following hyperparameters: gradient clipping coefficient $\lambda_{\text{usfa-c}} \in \{0.01, 0.001\}$, target network update coefficient $\tau_{\text{usfa-c}} \in \{0.01, 0.001\}$, policy update delay $d_{\text{usfa-a}} \in \{1, 3, 5\}$, policy noise standard deviation $\sigma_{\text{usfa-a}} \in \{0.0, 0.1, 0.2\}$. Here the subscripts usfa-a and usfa-c refer to the actor and the SF estimator modules, respectively. The final pretraining hyperparameters were selected based on cumulative zero-shot performance across all 12 domain-task pairs on the APS dataset, with a basis size of $k = 50$, averaged over 10 random seeds.

Downstream: For the downstream adaptation phase, we fixed the pretraining hyperparameters determined above and conducted a separate grid search to optimize the meta-policy. Using a reduced basis size of $k = 3$, we searched over: exploration noise $\sigma_{\text{lk}} \in \{0.01, 0.001\}$, batch size $B_{\text{lk}} \in \{16, 32, 64\}$, policy update delay $d_{\text{lk}} \in \{1, 3, 5\}$, target network update coefficient $\tau_{\text{lk}} \in \{0.01, 0.001\}$, and option duration $t_{\text{term}} \in \{1, 5, 10\}$. The optimal configuration was selected based on performance on CHEETAH `Run` and `Run-B` across 10 seeds, then applied uniformly across all domains, tasks, and basis sizes.

Baselines: We adopt the baseline implementations and hyperparameter settings for flat TD3 agents from Huang et al. (2022) and Yarats et al. (2022), and for FB from Tirinzoni et al. (2025) and Touati et al. (2023) in DMC, as well as from Alegre et al. (2025) for OKB on the ITEM-COLLECTOR task. To ensure a fair comparison with LK, FB is trained for 1M gradient steps, with all other hyperparameters kept consistent with Tirinzoni et al. (2025).

C.4 Computation Resources

All experiments are conducted on NVIDIA L40S with 48GB VRAM.

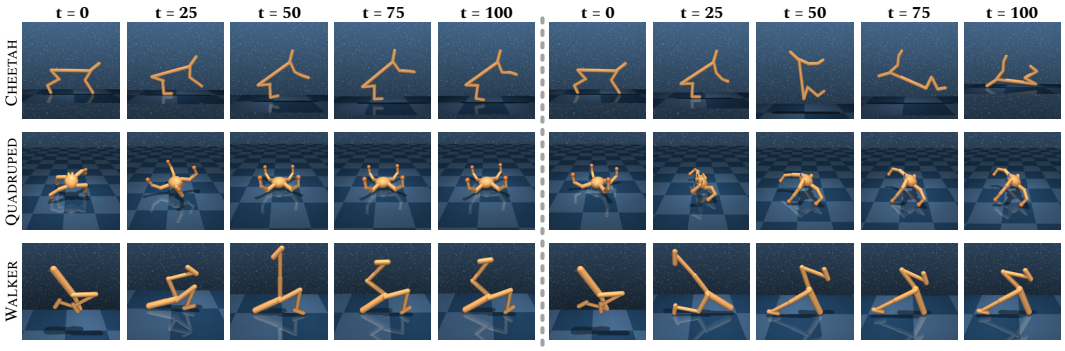


Figure 9: Instances from the Laplacian behavior basis obtained from the APS dataset. The left panel shows the behavior corresponding to $\mathbf{w}_{-\mathbf{e}_1} = [-1, 0, 0, \dots]$, while the right panel shows the behavior corresponding to $\mathbf{w}_{+\mathbf{e}_1} = [+1, 0, 0, \dots]$.

D Additional Results

D.1 Laplacian Behavior Basis

Figure 9 illustrates the behaviors induced by the first Laplacian eigenvector across the three domains, corresponding to the negative ($\mathbf{w}_{-\mathbf{e}_1}$) and positive ($\mathbf{w}_{+\mathbf{e}_1}$) directions. Although the specific behaviors differ across environments, the two directions consistently induce opposing outcomes: in CHEETAH, forward hopping versus an inverted posture; in QUADRUPED, an inverted stance versus an upright posture; in WALKER, a backward headstand versus a forward one. This oppositional structure is not unique to the first eigenvector—we observe it consistently across other directions and multiple runs.

Figure 10 extends this analysis to $\mathbf{w}_{\pm\mathbf{e}_2}$ and $\mathbf{w}_{\pm\mathbf{e}_3}$, revealing a broader repertoire of emergent behaviors. In CHEETAH, these include backflipping, sitting on hindquarters, hopping while seated, and transitioning from a backflip into a headstand. QUADRUPED learns to lie on its side or back with distinct limb configurations, while WALKER acquires headstands, multiple forms of splits, and lying flat. While the specific behaviors may vary across runs and datasets due to stochasticity in the unsupervised optimization, the approach consistently produces a diverse and temporally coherent behavioral basis.

Crucially, these behaviors emerge from reward-free data alone—no hand-crafted features or explicit skill enumeration required. Prior option-based methods (Barreto et al., 2019; Alegre et al., 2025) relied on carefully engineered reward features to construct a behavior basis, a process that demands domain knowledge and does not scale. Laplacian eigenvectors recover this structure automatically from environment dynamics.

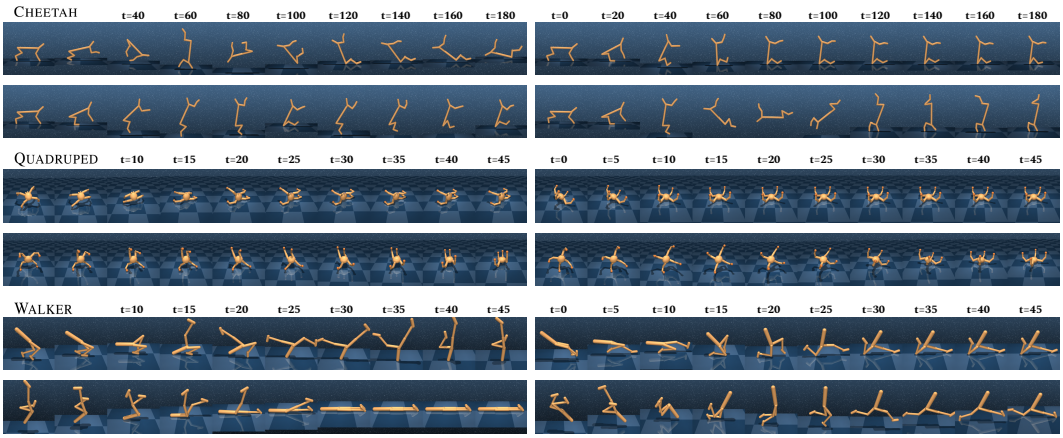


Figure 10: Behaviors learned during pre-training on the APS dataset. Within each domain, behaviors correspond (clockwise) to $\mathbf{w}_{-\mathbf{e}_2}$, $\mathbf{w}_{+\mathbf{e}_2}$, $\mathbf{w}_{+\mathbf{e}_3}$, and $\mathbf{w}_{-\mathbf{e}_3}$.

Table 5: Zero-shot performance comparison of FB and Laplacian basis across datasets for $k = 50$. Reported values are mean $\pm$ standard error, estimated from 30 independent runs.

	APS		Proto		RND	
Task	FB	Laplacian	FB	Laplacian	FB	Laplacian
CHEETAH *(avg)*	622 $\pm$31	591 $\pm$31	588 $\pm$33	551 $\pm$33	447 $\pm$27	208 $\pm$19
Run	329 $\pm$17	265 $\pm$13	248 $\pm$8	216 $\pm$14	169 $\pm$9	108 $\pm$13
Run-B	281 $\pm$8	297 $\pm$13	217 $\pm$8	217 $\pm$7	191 $\pm$9	51 $\pm$5
Walk	899 $\pm$34	848 $\pm$30	937 $\pm$15	833 $\pm$38	621 $\pm$27	446 $\pm$43
Walk-B	981 $\pm$3	953 $\pm$24	948 $\pm$14	937 $\pm$14	807 $\pm$27	228 $\pm$24
QUADRUPED *(avg)*	612 $\pm$13	597 $\pm$17	228 $\pm$11	268 $\pm$12	569 $\pm$14	662 $\pm$17
Jump	625 $\pm$12	674 $\pm$22	221 $\pm$15	288 $\pm$20	602 $\pm$8	701 $\pm$13
Run	428 $\pm$10	425 $\pm$10	162 $\pm$13	204 $\pm$13	361 $\pm$7	470 $\pm$8
Stand	799 $\pm$15	817 $\pm$23	332 $\pm$28	385 $\pm$28	720 $\pm$14	911 $\pm$14
Walk	595 $\pm$12	470 $\pm$12	195 $\pm$13	195 $\pm$15	593 $\pm$22	565 $\pm$26
WALKER *(avg)*	484 $\pm$25	588 $\pm$22	665 $\pm$23	713 $\pm$24	604 $\pm$16	444 $\pm$29
Flip	211 $\pm$20	604 $\pm$21	476 $\pm$30	682 $\pm$11	548 $\pm$19	236 $\pm$38
Run	266 $\pm$16	327 $\pm$21	396 $\pm$13	343 $\pm$7	405 $\pm$7	211 $\pm$24
Stand	660 $\pm$30	589 $\pm$45	896 $\pm$11	884 $\pm$37	704 $\pm$28	433 $\pm$33
Walk	798 $\pm$15	833 $\pm$30	891 $\pm$8	941 $\pm$4	759 $\pm$14	896 $\pm$16

D.2 Generality of the Laplacian Basis

Table 5 reports zero-shot performance of FB and the Laplacian basis ($k = 50$) for all three datasets. Neither method consistently dominates the other, though dataset choice does affect relative performance—for instance, the Laplacian basis outperforms FB on WALKER with APS, while FB does better with RND. Overall, the two methods perform similarly across conditions, consistent with the sign test reported in Section 5.1.

Figure 11 visualizes the zero-shot performance of the Laplacian basis as a function of the basis size for each dataset separately, with the corresponding numerical values reported in Table 6. The qualitative trend from the main paper holds broadly: performance improves with k and saturates, with QUADRUPED remaining flat across all basis sizes and datasets—consistently confirming that $k = 1$ captures the dominant task structure in this domain regardless of data collection strategy. Across CHEETAH and WALKER, dataset quality plays a meaningful role: APS yields the strongest performance and saturates earliest (around $k = 20$). Notably, the relative ordering of tasks within each domain is largely preserved across datasets.

These results reveal a dataset bias: offline data quality directly shapes the expressivity of the learned basis, with APS consistently outperforming Proto and RND at smaller k. This is not specific to the Laplacian representation—FB exhibits the same trend (Table 5)—but a general consequence of learning from fixed offline data.

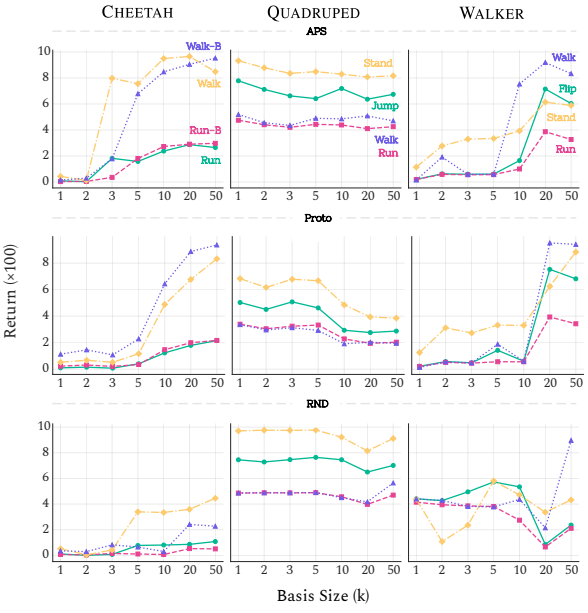


Figure 11: Zero-shot performance of the Laplacian basis as a function of basis size and dataset. Each row corresponds to a dataset, each curve to a task, and points indicate averages over 30 random seeds.

Table 6: Zero-shot performance of the Laplacian basis for different sizes and datasets. Reported values are mean ± standard error, estimated from 30 independent runs.

		Basis Size k						
Dataset	**Task**	1	2	3	5	10	20	50
APS	CHEETAH *(avg)*	17 ±2	12 ±3	298 ±28	443 ±29	577 ±33	612 ±32	591 ±31
	Run	9 ±2	2 ±0	182 ±10	158 ±6	238 ±9	287 ±10	265 ±13
	Run-B	3 ±1	5 ±2	36 ±2	181 ±12	273 ±21	291 ±19	297 ±13
	Walk	43 ±7	12 ±1	798 ±32	756 ±24	949 ±8	965 ±5	848 ±30
	Walk-B	15 ±4	29 ±12	178 ±13	679 ±48	847 ±49	905 ±39	953 ±24
	QUADRUPED *(avg)*	676 ±17	622 ±17	588 ±17	606 ±16	618 ±16	591 ±16	597 ±17
	Jump	778 ±5	712 ±8	662 ±9	641 ±15	719 ±12	637 ±21	674 ±22
	Run	475 ±1	440 ±4	421 ±5	443 ±7	439 ±6	410 ±9	425 ±10
	Stand	933 ±6	879 ±9	835 ±9	849 ±15	830 ±21	807 ±20	817 ±23
	Walk	519 ±7	456 ±12	435 ±18	490 ±16	486 ±9	508 ±5	470 ±12
	WALKER *(avg)*	42 ±4	148 ±13	126 ±11	128 ±11	353 ±30	659 ±24	588 ±22
	Flip	20 ±0	63 ±1	60 ±0	61 ±0	165 ±35	715 ±26	604 ±21
	Run	19 ±0	59 ±1	56 ±0	56 ±0	100 ±17	387 ±18	327 ±21
	Stand	114 ±2	277 ±18	330 ±0	335 ±1	393 ±27	614 ±56	589 ±45
	Walk	15 ±1	192 ±37	59 ±0	60 ±0	753 ±61	919 ±20	833 ±30
Proto	CHEETAH *(avg)*	50 ±6	65 ±10	47 ±7	105 ±17	351 ±31	486 ±32	551 ±33
	Run	10 ±1	15 ±5	8 ±1	41 ±9	123 ±16	180 ±14	216 ±14
	Run-B	23 ±3	31 ±4	21 ±4	35 ±9	147 ±15	200 ±10	217 ±7
	Walk	52 ±6	67 ±25	52 ±11	117 ±30	488 ±65	677 ±61	833 ±38
	Walk-B	113 ±16	146 ±22	108 ±20	228 ±53	644 ±63	888 ±21	937 ±14
	QUADRUPED *(avg)*	466 ±16	417 ±17	456 ±18	439 ±19	300 ±15	267 ±14	268 ±12
	Jump	503 ±18	451 ±21	508 ±24	462 ±30	294 ±22	276 ±23	288 ±20
	Run	339 ±14	305 ±18	324 ±17	333 ±18	228 ±16	195 ±15	204 ±13
	Stand	684 ±27	617 ±34	679 ±33	668 ±36	485 ±33	394 ±36	385 ±28
	Walk	336 ±16	296 ±20	313 ±18	293 ±23	192 ±14	204 ±18	195 ±15
	WALKER *(avg)*	46 ±4	119 ±10	104 ±9	180 ±19	127 ±11	681 ±24	713 ±24
	Flip	22 ±1	57 ±2	48 ±2	142 ±38	60 ±0	753 ±25	682 ±11
	Run	21 ±1	52 ±1	46 ±2	56 ±0	56 ±0	394 ±9	343 ±7
	Stand	126 ±6	312 ±9	273 ±12	333 ±1	330 ±1	625 ±54	884 ±37
	Walk	14 ±1	56 ±2	47 ±2	189 ±55	60 ±0	953 ±2	941 ±4
RND	CHEETAH *(avg)*	26 ±3	10 ±1	37 ±5	123 ±15	113 ±13	185 ±13	208 ±19
	Run	11 ±2	1 ±0	8 ±2	78 ±8	81 ±6	86 ±3	108 ±13
	Run-B	7 ±1	4 ±1	15 ±3	11 ±2	7 ±1	53 ±4	51 ±5
	Walk	52 ±9	4 ±1	43 ±10	340 ±32	335 ±21	359 ±14	446 ±43
	Walk-B	36 ±5	29 ±4	82 ±15	65 ±13	30 ±6	242 ±22	228 ±24
	QUADRUPED *(avg)*	672 ±19	670 ±19	674 ±19	680 ±19	644 ±19	570 ±19	662 ±17
	Jump	745 ±11	728 ±7	746 ±9	764 ±6	746 ±12	650 ±22	701 ±13
	Run	486 ±2	488 ±1	488 ±1	489 ±0	457 ±10	398 ±14	470 ±8
	Stand	970 ±3	976 ±1	975 ±1	977 ±1	922 ±18	815 ±28	911 ±14
	Walk	487 ±3	488 ±2	488 ±3	490 ±2	450 ±10	417 ±18	565 ±26
	WALKER *(avg)*	433 ±2	340 ±12	375 ±12	478 ±9	429 ±14	176 ±17	444 ±29
	Flip	441 ±2	427 ±2	495 ±10	573 ±2	534 ±17	85 ±6	236 ±38
	Run	415 ±3	395 ±3	386 ±4	381 ±3	275 ±14	66 ±7	211 ±24
	Stand	438 ±4	108 ±1	236 ±31	581 ±2	472 ±35	337 ±3	433 ±33
	Walk	440 ±2	429 ±2	382 ±7	378 ±8	437 ±11	215 ±53	896 ±16

Table 7: Relative improvement of LK over the zero-shot performance (*left*) and ratio of area under the evaluation curve (AUC) over the first 200k environment steps (*right*), across basis sizes k. Results for the `Proto` and `RND` datasets, each averaged over 30 runs.

Task	Improv. over zero-shot (%)							Downstream AUC ratio						
	1	2	3	5	10	20	50	1	2	3	5	10	20	50
Proto														
CHEETAH *(avg)*	1987	8217	7672	21718	931	148	39	0.5	0.6	0.8	1.0	1.3	1.5	1.5
Run	2137	15471	14704	63998	531	141	99	0.7	0.7	1.1	1.0	1.7	2.0	2.0
Run-B	2137	2517	3799	7625	1804	38	26	0.3	0.4	0.4	0.7	0.7	0.9	0.9
Walk	1626	13815	6496	9069	571	403	29	0.8	1.0	1.2	1.2	1.7	2.0	2.0
Walk-B	2048	1066	5690	6180	818	10	2	0.4	0.4	0.5	1.0	1.0	1.2	1.2
QUADRUPED *(avg)*	−10	−22	−5	−28	1	42	−2	2.0	1.2	1.7	0.9	0.9	1.1	0.7
Jump	−9	−14	−19	−40	1	37	−5	2.1	1.4	1.7	0.7	0.8	1.2	0.8
Run	−15	−19	−7	−44	−10	33	−1	1.9	1.3	1.6	0.8	1.0	1.2	0.7
Stand	2	−26	−7	−26	−11	37	14	2.3	1.2	1.8	1.1	1.1	1.2	1.0
Walk	−19	−28	12	−3	24	60	−14	1.6	0.9	1.6	0.9	0.8	1.0	0.5
WALKER *(avg)*	1264	388	549	623	799	41	9	0.8	0.9	1.0	1.8	1.8	2.2	1.8
Flip	1168	467	690	790	997	49	10	0.8	0.9	1.0	1.9	1.9	2.6	2.0
Run	583	223	309	534	660	23	19	0.7	0.9	0.9	1.9	2.1	2.6	2.1
Stand	324	85	141	157	172	94	13	0.7	0.8	0.9	1.2	1.3	1.4	1.2
Walk	2979	778	1056	1009	1365	−1	−4	0.9	1.1	1.2	2.0	2.0	2.3	2.0
RND														
CHEETAH *(avg)*	4500	18658	11940	909	1137	111	335	0.5	0.4	0.5	0.7	0.6	0.7	0.6
Run	3985	37205	14968	182	77	78	907	0.5	0.6	0.6	1.0	0.8	0.9	1.0
Run-B	4743	4936	1800	2049	2331	148	137	0.3	0.2	0.2	0.2	0.2	0.2	0.2
Walk	4443	29938	29536	103	69	41	177	0.6	0.8	0.8	1.1	1.1	1.1	1.1
Walk-B	4829	2556	1457	1301	2073	179	117	0.5	0.3	0.3	0.4	0.4	0.4	0.3
QUADRUPED *(avg)*	−2	−3	−14	−22	−32	−35	−41	3.2	3.1	2.8	2.4	2.0	1.6	1.7
Jump	−2	−5	−8	−18	−39	−41	−36	3.4	3.2	3.1	2.7	2.0	1.6	1.9
Run	1	−2	−14	−21	−31	−30	−43	3.4	3.3	2.8	2.5	2.0	1.8	1.7
Stand	1	−3	−13	−26	−28	−34	−41	3.4	3.2	2.9	2.4	2.2	1.7	1.8
Walk	−7	−2	−20	−25	−32	−36	−43	2.7	2.7	2.2	2.1	1.7	1.4	1.6
WALKER *(avg)*	−1	88	65	13	46	560	224	1.4	1.4	1.3	1.4	1.5	1.8	1.9
Flip	0	1	6	−2	14	660	536	1.4	1.4	1.6	1.7	1.7	1.7	2.0
Run	0	0	−23	−12	15	551	225	2.4	2.3	1.7	1.8	1.6	2.1	2.2
Stand	−5	351	258	20	105	164	135	0.6	0.7	0.8	1.0	1.3	1.3	1.3
Walk	0	1	19	48	51	864	0	1.1	1.1	1.1	1.2	1.5	2.1	2.1

D.3 Hierarchical Composition

Table 7 reports the percentage improvement of LK over its zero-shot performance (left) and sample efficiency gains over the flat TD3 (right) on the `Proto` and `RND` datasets. The qualitative usions from the main paper hold across both datasets, though the magnitude of gains varies.

In CHEETAH and WALKER, hierarchical composition remains most beneficial at small basis sizes, with improvements exceeding 1000% on several tasks at $k = 1$ and diminishing returns as k increases. The `Proto` dataset tends to yield larger raw improvements than `RND`. An exception is WALKER under `RND`, where improvements at small k are near zero, analogous to the pattern observed in QUADRUPED. In QUADRUPED, LK fails to consistently improve over the zero-shot solution across both datasets, with the effect most pronounced under `RND`, where improvements are negative across settings.

The learning curves across all three datasets (`APS`: Figure 12, `Proto`: Figure 13, `RND`: Figure 14) reveal a consistent pattern: LK converges to a stable policy rapidly, often within a fraction of the interaction budget of the flat agent. The key takeaway is that *without access to rewarded offline transitions, LK improves over the zero-shot solution and stabilizes quickly*. When the zero-shot initialization is already strong—as in QUADRUPED across all datasets—this fast convergence translates directly into strong AUC ratios (Table 7, right). In CHEETAH under `RND`, LK improves over the zero-shot solution but does not consistently match the flat agent's cumulative returns, suggesting that dataset coverage limits how much structure the Laplacian basis can extract. This points to a broader usion: the offline dataset shapes the expressivity of the learned basis, and with it, the ceiling on what hierarchical composition can achieve.

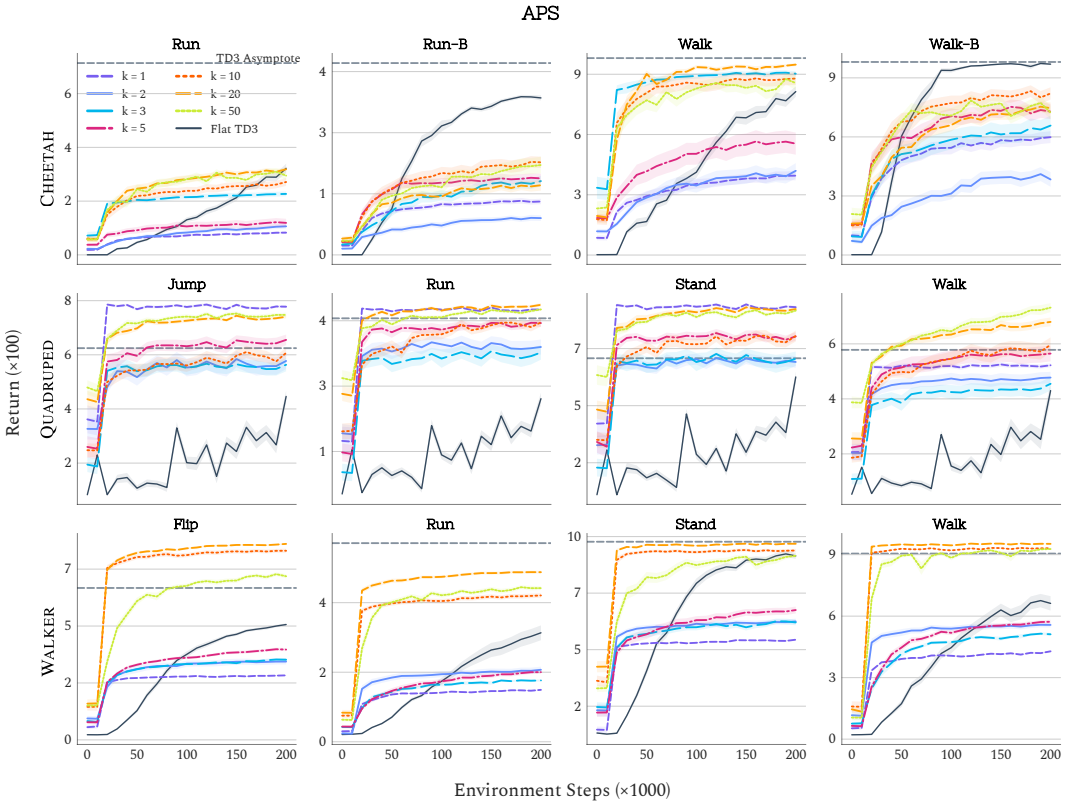


Figure 12: Evaluation curves for LK pre-trained on the `APS` dataset with varying basis sizes, and for a flat TD3 agent. Shaded region indicates the standard error estimated over 30 independent runs.

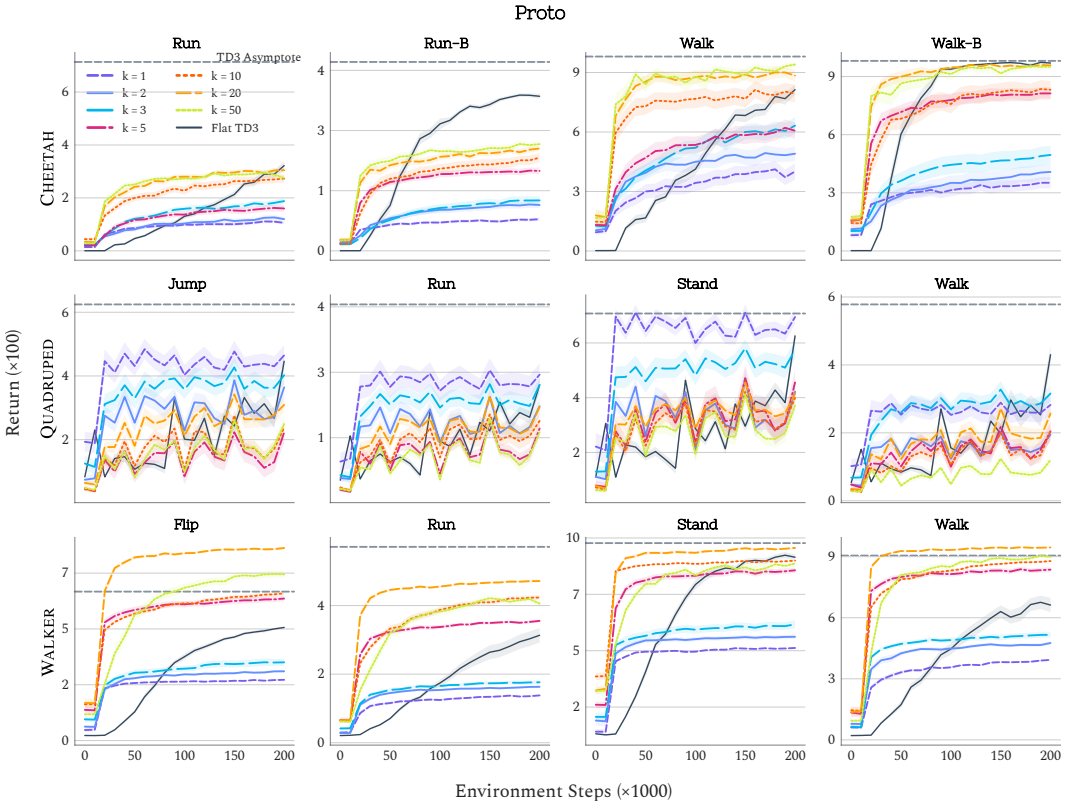


Figure 13: Evaluation curves for LK pre-trained on the `Proto` dataset with varying basis sizes, and for a flat TD3 agent. Shaded region indicates the standard error estimated over 30 independent runs.

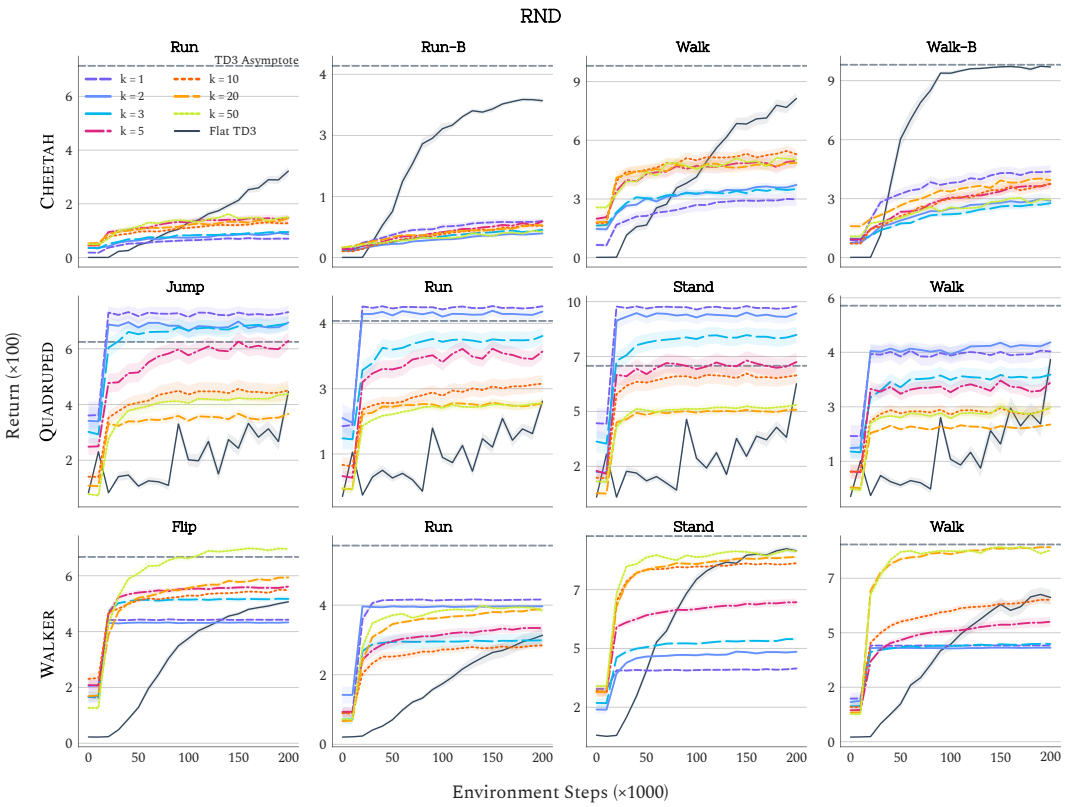


Figure 14: Evaluation curves for LK pre-trained on the RND dataset with varying basis sizes, and for a flat TD3 agent. Shaded region indicates the standard error estimated over 30 independent runs.

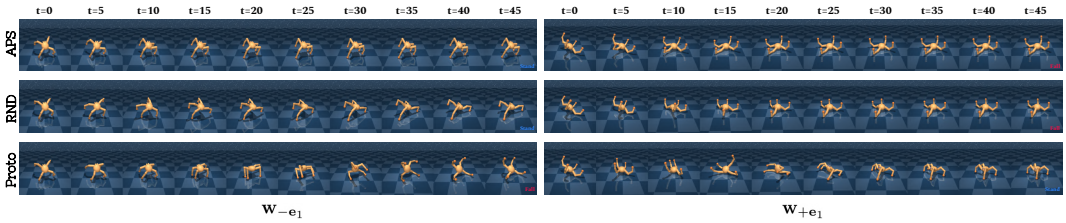


Figure 15: Behaviors induced by $\mathbf{w}_{-\mathbf{e}_1}$ and $\mathbf{w}_{+\mathbf{e}_1}$ in the QUADRUPED domain in all three datasets.

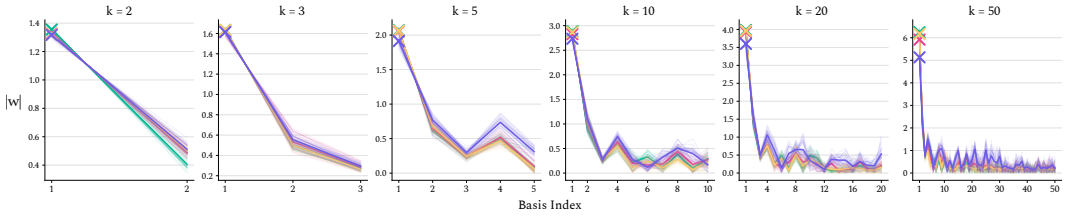


Figure 16: Absolute zero-shot weight magnitudes for all tasks, grouped by basis size. Each subplot shows all task-specific weights with low opacity, together with the mean absolute weight plotted as a thick line. Note that the entries in the x-axis are different indices for the weight magnitudes, there is no notion of time progression or size within a single panel; we connect the datapoints with a line to ease visualization. The first basis coefficient is indicated by a cross.

E The Curious Case of QUADRUPED

In this section, we analyze the behaviors induced by the weight vectors $\mathbf{w}_{-\mathbf{e}_1}$ and $\mathbf{w}_{+\mathbf{e}_1}$ in QUADRUPED to characterize their role in task performance. Across all three datasets and independent runs, these vectors consistently induce two dominant behaviors: maintaining an upright posture or collapsing onto the back (see Figure 15). From arbitrary initial configurations, conditioning on $\mathbf{w}_{-\mathbf{e}_1}$ or $\mathbf{w}_{+\mathbf{e}_1}$ modulates the torso's elevation relative to the ground, effectively controlling upright versus collapsed postures.

Coincidentally, all four QUADRUPED tasks include an uprightness term that measures alignment between the torso's vertical axis and the global vertical axis. The `Stand` task is defined solely by this term and encourages recovery from arbitrary initial orientations. The `Jump` task modulates the uprightness term with a height-dependent factor that increases with center-of-mass elevation and saturates at a target height. The `Walk` and `Run` tasks further scale the uprightness term by a forward-velocity factor that saturates at task-specific target speeds, with `Walk` favoring lower-speed locomotion and `Run` favoring higher-speed locomotion.

This shared uprightness structure aligns closely with the behaviors induced by $\mathbf{w}_{-\mathbf{e}_1}$ and $\mathbf{w}_{+\mathbf{e}_1}$, providing a direct explanation for the strong performance observed when using only the leading eigenvector as the basis. Figure 16 illustrates this effect by showing the estimated zero-shot weight vectors across tasks and basis sizes on the `APS` dataset. Across all conditions, the coefficient corresponding to the first basis component consistently dominates, indicating that it captures the primary reward-aligned direction. Additional eigenvectors contribute comparatively little and primarily introduce representational overhead, which correlates with the observed degradation in zero-shot performance as the basis size increases.

Consistent with this interpretation, Figure 11 shows that average performance decreases as task rewards deviate from a pure uprightness objective, following the ordering `Stand` $>$ `Jump` $>$ `Walk` $\approx$ `Run`. Tasks that introduce additional height- or velocity-dependent modulation require structure beyond the dominant basis direction, reducing the effectiveness of a minimal representation.

F Effect of Option Termination Horizon

Options in LK are terminated after a fixed horizon t_{term} rather than through a state-dependent condition. All results reported in Section 5.2 use $t_{\text{term}} = 5$. Here we study how sensitive LK's performance is to this choice. Table 8 reports the area under the curve (AUC) for LK on CHEETAH with the APS dataset ($k = 50$, 10 seeds) across $t_{\text{term}} \in \{5, 10, 15\}$.

Table 8: AUC for LK on CHEETAH (APS, $k = 50$) across termination horizons t_{term}.

t_{term}	Run	Run-B	Walk	Walk-B
5	5256	3249	15 690	11 309
10	5412	3103	14 223	13 742
15	5967	2888	17 018	12 345

No single value of t_{term} dominates across all tasks, yet all three values of t_{term} achieve strong performance, indicating that LK is not highly sensitive to this choice. That said, fixed-horizon termination is a simplification: learned or state-dependent termination conditions could better align option boundaries with task structure and likely improve performance further, which we leave for future work.

G Meta-Policy Parameterization: USFA vs GPE&GPI

We detail the two meta-policy variants introduced in Section 6. *LK-USFA* uses a continuous parameterization: the meta-policy outputs a weight vector $\mathbf{w} \in \mathbb{R}^k$, and the USFA directly executes the policy $\pi_{\mathbf{w}}$ optimal for the induced reward $r_{\mathbf{w}}(s) = \mathbf{w}^\top \boldsymbol{\phi}(s)$. *LK-GPE&GPI* uses a discrete parameterization: given a library of $2k$ basis policies induced by $\{\mathbf{e}_1, \ldots, \mathbf{e}_k, -\mathbf{e}_1, \ldots, -\mathbf{e}_k\}$, the meta-policy outputs a weight vector $\mathbf{w}$. The primitive action is selected by Generalized Policy Evaluation (GPE) followed by Generalized Policy Improvement (GPI) over this library:

$$\pi_{\text{GPI}}(s) = \arg\max_a \max_{i=1,\ldots,2k} \boldsymbol{\psi}^{\pi_i}(s,a)^\top \mathbf{w}, \tag{27}$$

where $\boldsymbol{\psi}^{\pi_i}$ are the SFs of the i-th basis policy. The positive and negative basis vectors together ensure that for any task direction $\mathbf{w}$, the basis contains a policy aligned with each signed component.

The continuous variant can in principle express any direction in $\mathbb{R}^k$ and is theoretically guaranteed to recover the optimal policy for the induced reward. The discrete variant must approximate this through two successive steps—policy evaluation under each basis policy, followed by improvement via maximization—accumulating error at each stage. Most prior work (Barreto et al., 2019; Carvalho et al., 2023; Alegre et al., 2025) adopts the discrete formulation; this ablation tests whether the continuous alternative offers a meaningful advantage. Note that discrete composition is not straightforward in continuous action spaces, as it requires an $\arg\max$ over actions—which is why we restrict this comparison to the discrete-action ITEM-COLLECTOR domain.

Figure 6 supports this analysis. At $k = 20$, both formulations perform similarly; however, the gap widens as k increases to 30 and 40. We attribute this to the compounding effect of representation error: larger basis dimensions make the learning problem harder, yielding less accurate SF estimates. These inaccuracies are then amplified across the two steps of GPE&GPI, whereas the continuous formulation absorbs them into a single direct optimization.

H Laplacian Keyboard: Architecture and Algorithm

We present Python-style pseudocode for training the Laplacian encoder, the USFA, and the meta-policy. Low-level implementation details are omitted for clarity.

```
def train_laplacian_encoder(self):

    # sample contrastive state pairs
    obs, pos_obs, neg_obs = self.
        sample_laplacian_batch()

    # encode states
    phi      = self.encoder(obs)
    phi_pos  = self.encoder(pos_obs)
    phi_neg  = self.encoder(neg_obs)

    # graph-smoothness loss
    graph_loss = mean((phi - phi_pos) ** 2)

    # orthogonality-enforcing loss
    error = compute_ortho_error(phi, phi_neg)

    dual_loss = sum(self.dual_vars.detach() * error)
    barrier_loss = sum(self.barrier_coefs.detach() *
         error.pow(2))

    # optimize encoder
    loss = graph_loss + dual_loss + barrier_loss
    self.encoder_optimizer.step()
```

```
def train_usfa(self):

    # sample transition batch
    obs, action, next_obs, done = self.
        sample_usfa_batch()
    w = self.sample_task_weight()

    # compute Laplacian features
    phi_next = self.encoder(next_obs)

    # sample next action
    next_action = self.actor(next_obs, w).sample()

    # compute target successor features
    psi_target = phi_next + (1 - done) * self.gamma
        * self.target_psi(next_obs, next_action, w)

    # predict successor features
    psi_pred = self.psi(obs, action, w)

    # TD loss
    psi_loss = mse(psi_pred - psi_target.detach())
    self.psi_optimizer.step()

    # optionally update actor and target networks
    if update_actor:
        actor_loss = -(psi_pred * w).sum(-1).mean()
        self.actor_optimizer.step()
        self.update_target_networks()
```

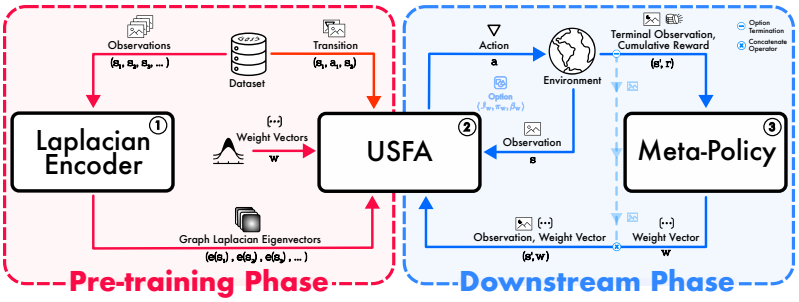


Figure 17: The Laplacian Keyboard Framework. During pre-training, the agent first learns graph Laplacian eigenvectors through the Laplacian Encoder ①, then uses these as state representations to learn a continuous library of options via a USFA ②. In the downstream phase, a meta-policy ③ learns to stitch these base policies, enabling rapid and sample-efficient adaptation to new tasks. Numbers indicate the learning sequence of each module.

```python
def train_downstream(self):
    obs = self.env.reset()
    done = False
    option_return = 0
    option_length = 0
    option_obs = obs

    # sample initial option
    w = self.meta_actor(obs)

    while not done:

        # act using USFA-conditioned policy
        action = self.actor(obs, w).sample()
        next_obs, reward, done = self.env.step(
            action)

        # accumulate option return
        option_return += (self.gamma **
            option_length) * reward
        option_length += 1

        # terminate option
        if option_length >= self.max_option_length
            or done:
            self.replay_buffer.store(option_obs, w,
                next_obs, option_return, self.gamma**
                option_length, done)
            w = self.meta_actor(next_obs)
            option_return = 0
            option_length = 0
            option_obs = next_obs

        obs = next_obs

        # update meta-policy
        if update_meta:
            batch = self.replay_buffer.sample()
            self.update_critic(batch)
            self.update_actor(batch)
            self.update_target_networks()
```