# Towards An Option Basis To Optimize All Rewards

**Anonymous authors**
Paper under double-blind review

## Abstract

The Option Keyboard framework enables efficient behavior generation by composing a set of basis options. However, it remains unclear how to construct a global and compact basis, from scratch, for solving any given task in an environment. In this work, we investigate using the eigenvectors of the graph Laplacian of the environment to form such a basis. The behaviors obtained from such eigenvectors are known as eigenoptions. We empirically demonstrate that a sufficiently large eigenoption basis, combined with Generalized Policy Improvement, can recover near-optimal policies in the goal-reaching tasks we considered. Building on this, we introduce the Laplacian Keyboard, which matches this performance while requiring a substantially smaller set of options. Finally, we briefly outline a method for constructing a universal optimal option basis capable of solving any task within a given environment.

## 1 Introduction

Options (Sutton et al., 1999) provide a powerful mechanism for credit assignment, exploration, and knowledge transfer. In this work, we focus on the latter—using a set of options to solve new tasks. While learning options from scratch for each task is not scalable, naively reusing previously learned options often leads to suboptimal performance. A potential solution to this problem involves learning a compact set of option bases that can then be combined to create new behaviors for new tasks (Barreto et al., 2017; 2019). This eliminates the need to learn options from scratch for each task, but raises a fundamental question: *What is a global and compact option basis that can be effectively combined to solve new tasks?*

Existing methods for acquiring option bases typically define options through linear combinations of *reward features*, creating a basis by learning options for diverse combinations of features. However, existing techniques (Alegre et al., 2025; Barreto et al., 2019; Carvalho et al., 2023) exhibit limitations in achieving a universal option basis. Specifically, methods relying on manually designed reward features (Alegre et al., 2025; Barreto et al., 2019) restrict the range of tasks these bases can effectively address. Approaches that learn reward features in a transfer reinforcement learning (RL) setting are constrained by the choice of training and testing environments (Carvalho et al., 2023).

In this paper, we investigate an approach to *learning a compact, task-agnostic option basis that does not rely on extensive reward priors nor environmental rewards*. Our idea is motivated by two results. (1) The eigenvectors of the graph Laplacian of an environment induced by a uniform policy can serve as a global basis, enabling agents to learn and represent any task's optimal value function efficiently (Mahadevan, 2005). Complementing this, (2) eigenoptions (Machado et al., 2017), which are options based on these same eigenvectors, can be combined to create diverse behaviors that are useful for exploration (Machado et al., 2023).

We assert that the eigenoptions form a global and compact basis for tackling a broad spectrum of tasks. Building on this premise, we first demonstrate that generalized policy evaluation (GPE) (Barreto et al., 2020) and generalized policy improvement (GPI) (Barreto et al., 2017), when applied over a sufficiently rich set of eigenoptions, can solve goal-reaching tasks in a gridworld. We

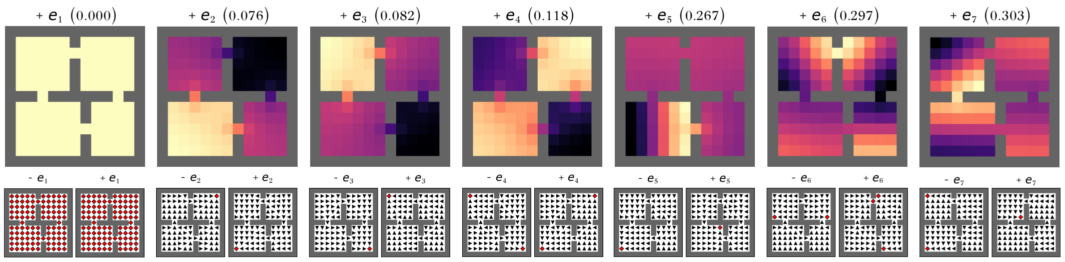

Figure 1: **Top row:** The first seven eigenvectors of the graph Laplacian induced by a random walk in the `Four-Room` domain. The brighter the state's color, the greater the entry of the eigenvector. The value in parentheses denotes the total graph variation (Mallat, 1999), with lower values indicating smoother eigenvectors. **Bottom row:** The first seven eigenoptions corresponding to both the negative and positive eigenvectors. The arrows indicate the optimal action, and the red diamond indicates the termination points.

then evaluate the performance of the Option Keyboard with an eigenoption basis, a combination we refer to as the *Laplacian Keyboard* (LK). This enables the composition of complex behaviors using a significantly smaller eigenoption basis to solve the same goal-reaching tasks mentioned above.

## 2 Preliminaries

**Reinforcement Learning:** A standard agent-environment interaction in RL is modeled as a Markov Decision Process (MDP), defined by the tuple $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P(\cdot \mid s, a)$ represents the transition distribution over next states given the current state, $s$, and action, $a$, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1)$ is the discount factor. The goal of the agent is to learn a policy, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, to maximize the total sum of discounted rewards, $\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$, where the actions are sampled based on $\pi$. We denote the transition matrix as $\mathbf{P}_\pi$, where $(\mathbf{P}_\pi)_{s,s'} = \sum_{a \in \mathcal{A}} \pi(s, a) P(s'|s, a)$ is the probability of transitioning from state $s$ to state $s'$ while following policy $\pi$.

In this work, we focus on a *reward-free* MDP setting (Agarwal et al., 2024), where the reward function $R$ is not specified during the initial phase of learning. The objective is to explore and understand the environment's dynamics without being guided by a specific reward, enabling the agent to efficiently solve downstream tasks once a reward function is later introduced. Related prior works (e.g., Alegre et al. 2025; Barreto et al. 2019) assume access to a set of reward features, $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^n$, which defines every possible downstream task. Each task's reward function corresponds to a linear combination of these features, where the weights can be interpreted as expressing preferences over them.

**Laplacian Representation:** Any MDP can be represented as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of vertices and $\mathcal{E}$ represents the edges. By this formulation, the states become the vertices and the edges between the states are characterized by $\mathbf{P}_\pi$. The Laplacian of such a graph is typically defined as $\mathbf{L} = \mathbf{I} - \frac{1}{2}(\mathbf{P}_\pi + \mathbf{P}_\pi^\top)$, ensuring that $\mathbf{L}$ remains symmetric. The eigenvectors of $\mathbf{L}$ induced by a random policy have been shown to capture the temporal properties of the environment and serve as a *global* basis to represent any value function of any policy for any reward function (Mahadevan, 2005; Mahadevan & Maggioni, 2007). Given an eigenvector $\mathbf{e}$, we denote its $s$-th entry, associated to state $s$, as $\mathbf{e}[s] = [\mathbf{e}_1[s], \mathbf{e}_2[s], \ldots, \mathbf{e}_{|\mathcal{S}|}[s]]$ where $\mathbf{e}_i[s]$ denotes the $i$-th entry of the eigenvector $\mathbf{e}$. In Figure 1, we plot the first seven eigenvectors of the graph Laplacian induced by a random policy in the `Four-Room` domain (Sutton et al., 1999). Observe that as the eigenvector index increases, the smoothness of the eigenvector decreases, transitioning from the constant valued $\mathbf{e}_1$ to eigenvectors with increasingly oscillatory behavior. In fact, the last few eigenvectors, corresponding to the highest frequencies, have a chessboard-like pattern.

**Options:** Options provide a mechanism for temporal abstraction, allowing an agent to execute a sequence of actions as a single, higher-level decision. Formally, an option $\omega$ is defined by a tuple $w = \langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$. Here, $\mathcal{I}_\omega$ is the initiation set, specifying the states from which the option can be invoked. Upon initiation, the agent follows the option's policy, $\pi_\omega$, until a state within its termination set, $\beta_\omega$, is reached.[1] Thus, an option can be viewed as a temporally extended action that initiates in specific states, follows its own policy, and terminates according to a defined condition.

**Eigenoptions:** Eigenoptions (Machado et al., 2017; 2018; 2023) are options defined using reward features based on the eigenvectors of the graph Laplacian. Prior work (Klissarov & Machado, 2023; Machado et al., 2023) learn eigenoptions by optimizing reward functions corresponding to individual eigenvectors, which could be thought of as one-hot linear combinations. For an MDP with $|\mathcal{S}|$ states, we can define $2|\mathcal{S}|$ eigenoptions by optimizing over both $+\mathbf{e}_i$ and $-\mathbf{e}_i$. Specifically, in the tabular case, for the transition $(s, a, s')$, the reward used to train the eigenoption corresponding to $+\mathbf{e}_i$ is $r_{+\mathbf{e}_i} = \mathbf{e}_i[s'] - \mathbf{e}_i[s]$, while for $-\mathbf{e}_i$ it is $r_{-\mathbf{e}_i} = \mathbf{e}_i[s] - \mathbf{e}_i[s']$. These options inherit the temporal structure encoded in the Laplacian: eigenvectors with lower eigenvalues tend to span the entire state space, resulting in longer options, whereas those associated with higher eigenvalues produce shorter options on average (see Figure 1) (Machado et al., 2023). Notably, the eigenoption derived from the first non-constant eigenvector has been widely adopted for exploration (Jinnai et al., 2019; 2020; Klissarov & Machado, 2023; Machado et al., 2023).

**General Policy Evaluation:** Given a policy, $\pi$, and a set of reward functions, $\mathcal{R} = \{R_i\}_{i=1}^n$, GPE enables efficient computation of the action-value function for any new reward of the form $R' = \sum_{i=1}^n w_i R_i$, where $\mathbf{w} \in \mathbb{R}^n$:

$$Q^\pi_{R'}(s, a) = \sum_{i=1}^n w_i Q^\pi_{R_i}(s, a). \tag{1}$$

Here, $Q^\pi_{R_i}$ denotes the action-value function of policy $\pi$ under reward function $R_i$. Once the set $\{Q^\pi_{R_i}\}_{i=1}^n$ is precomputed, GPE allows rapid evaluation of $\pi$ for any reward function within the linear span of $\mathcal{R}$ (Barreto et al., 2020).

**General Policy Improvement:** Given a set of policies, GPI allows one to construct a new policy that is at least as good as any individual policy in the set (Barreto et al., 2017). Let $\Pi = \{\pi_i\}_{i=1}^n$ be a set of $n$ policies, and let $R'$ be a new reward function. The GPI policy, $\pi^{GPI}_{R'}$, is defined as

$$\pi^{\text{GPI}}_{R'}(s) \in \arg\max_{a \in \mathcal{A}} \max_{\pi_i \in \Pi} Q^{\pi_i}_{R'}(s, a), \tag{2}$$

where $Q^{\pi_i}_{R'}$ is $i$-th policy evaluated on the new reward function. The newly obtained policy is at least as good as any other policy $\pi_i \in \Pi$. Hence, GPI offers a principled method to combine different policies to construct an improved policy.

GPE and GPI, together with a basis of reward functions and their corresponding policies, allow the synthesis of improved policies for any reward function within the span of the reward basis. Barreto et al. (2017) show that the performance gap between the GPI-derived policy, $\pi^{\text{GPI}}_{R'}$, and the true optimal policy can be bounded by how similar the target reward function, $R'$, is to the existing set of tasks $\mathcal{R}$. With a sufficiently diverse and representative set of policies, GPE and GPI can closely approximate the optimal policy for a wide range of reward functions.

**Option Keyboard:** The Option Keyboard (OK) (Barreto et al., 2019) generalizes GPE and GPI by enabling state-dependent combinations of policies or options. In the standard GPI formulation, a fixed weight vector $\mathbf{w} \in \mathbb{R}^n$ is used to combine $n$ policies, resulting in a *static* behavior that applies the same policy combination uniformly across all states.

[1]In this paper, because we only consider deterministic termination conditions, we abuse the notation of $\beta$ to call it a termination set instead of a termination condition.

In contrast, the OK introduces a meta-policy, $\pi_{\text{OK}} : \mathcal{S} \to \mathbb{R}^n$,[2] that outputs a state-dependent weight that *dynamically* combines the available options. This allows the agent to adapt its behavior based on the current state, enabling the synthesis of a broader set of behaviors from a relatively small set of base options. Operationally, OK can be viewed as a two-step process. The meta-policy, $\pi_{\text{OK}}$, first generates a weight vector $\mathbf{w}$ based on the current state. This weight is then used to combine the base options via GPE and GPI, resulting in a new composite option that is executed until termination. The meta-policy generates a new weight, and this process repeats until the task is solved.

In the context of the OK framework, options are typically constructed using a set of reward features $\boldsymbol{\phi} \in \mathbb{R}^d$, which define a space of reward functions of the form $r_w(s, a, s') = \boldsymbol{w}^\top \boldsymbol{\phi}(s, a, s')$. Each option is optimized for a particular weight vector $\boldsymbol{w}$ in this space. A basis of such options enables efficient transfer to new reward functions of a similar form (Alegre et al., 2022).

## 3 GPE and GPI with Eigenoptions

Machado et al. (2023) demonstrated that eigenoptions can be effectively combined using GPE and GPI to generate diverse exploratory behavior. In contrast, our focus lies in solving tasks. We begin by evaluating how well eigenoptions, when combined with GPE and GPI, can solve goal-reaching tasks. Our primary objective is to assess how closely the GPI-derived policy, $\pi_K^{\text{GPI}}$, approximates the optimal policy, $\pi_K^*$, for a reward reconstructed from the first $K$ eigenvectors of the graph Laplacian. The reward approximation is given by:

$$R_K \approx \sum_{i=1}^{K} w_i^{GPI} \mathbf{e}_i, \quad \text{where} \quad w_i^{GPI} = \langle \mathbf{e}_i, R \rangle \, . \tag{3}$$

The weights, $\mathbf{w}^{\text{GPI}} = [w_1^{\text{GPI}}, \ldots w_K^{\text{GPI}}]$, are reused to linearly combine the corresponding eigenoptions. Applying GPE followed by GPI over these options yields the composed policy $\pi_K^{\text{GPI}}$.

Our decision to use eigenoptions stems from the fact that the eigenvectors of the graph Laplacian $\mathbf{L}$ form a complete basis for functions over the state space. In particular, they can represent any reward function of the form $R(s, a, s') = R(s')$ (Mahadevan, 2005). These eigenvectors act as reward features that define eigenoptions (Machado et al., 2017). To build the basis of eigenoptions, we generate two eigenoptions from each eigenvector, resulting in $2 \times K$ options for a basis of size $K$.

We evaluate our method on goal-reaching tasks in the `Four-Room` domain. In goal-reaching tasks, the reward is zero for all transitions except when reaching the goal state, in which the agent receives a $+1$ reward. The `Four-Room` environment has 104 states, out of which 1 is the terminal goal state.

In Figure 2 we plot the reconstructed rewards following Eq. 3 for $K \in \{5, 10, 25, 50, 75, 100, 104\}$. Note that the reconstruction error decreases as $K$ increases. An exact reconstruction is possible when all eigenvectors are used, which is infeasible in large MDPs. Therefore, we typically rely on a smaller subset of smooth eigenvectors, which still capture essential reward structure (Mahadevan, 2005). This effectively imposes a prior that rewards vary smoothly over the state space.

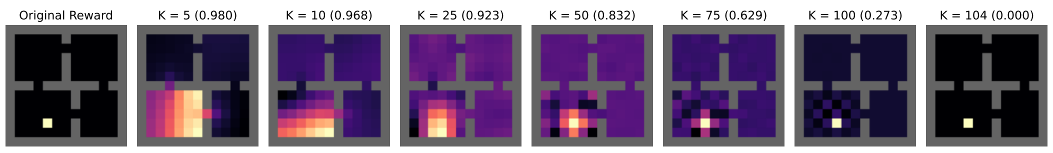

Figure 2: **Reward reconstruction using the first $K$ eigenvectors:** The leftmost image shows the original goal-reaching reward function, $R$. The subsequent images depict its reconstruction, $R_K$, using the top $K$ eigenvectors, as defined in Eq. 3. The value in parentheses indicates the mean squared reconstruction error between $R$ and $R_K$.

[2]We slightly abuse notation: unlike a standard policy mapping states to action distributions, $\pi_{\text{OK}}$ maps each state to unconstrained real-valued weights over options.

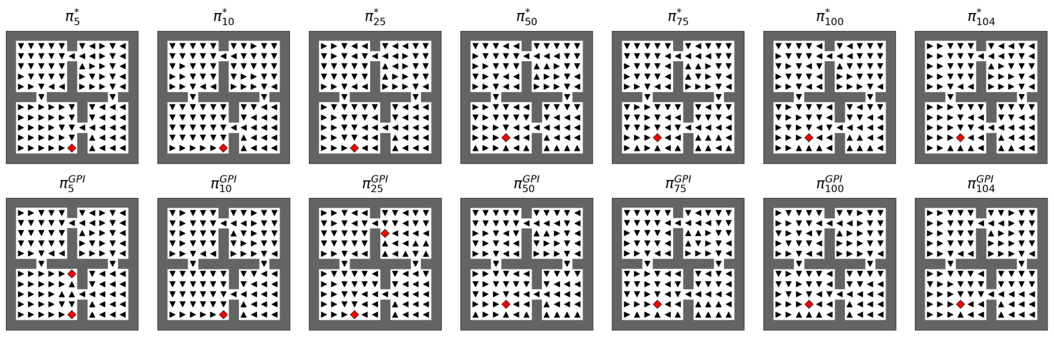

Figure 3: Behavior generated by GPE and GPI over $K$ eigenoptions for various $R_K$ in Figure 2.

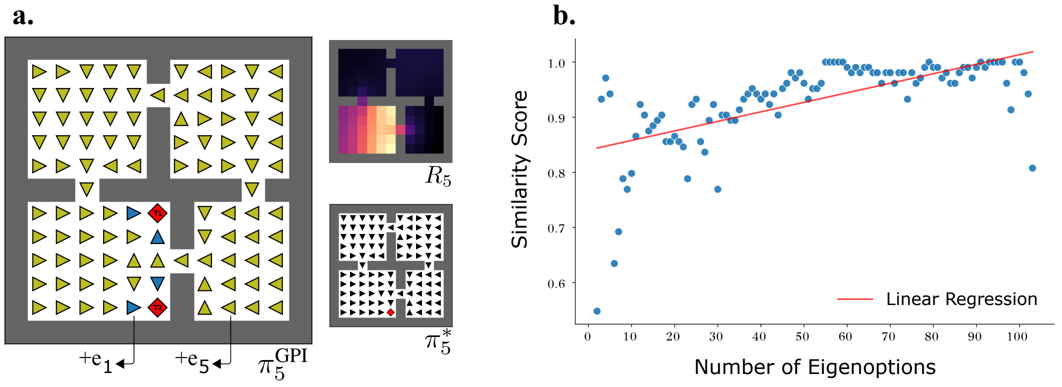

Figure 4: **Understanding the mismatch between $\pi^{\text{GPI}}_K$ and $\pi^*_K$.** (a) This panel illustrates $\pi^{\text{GPI}}_5$, where each state's action is derived from the eigenoption $\pi_i$ (indicated by blue for $+\mathbf{e}_1$ and yellow for $+\mathbf{e}_5$) that yielded the maximum value in that specific state (Eq 2). In this case, only $\mathbf{e}_1$ and $\mathbf{e}_5$ contribute to $\pi^{\text{GPI}}_5$, while the other options are considered but do not affect the final policy. The two terminal states are marked as $\top 1$ and $\top 2$. (b) Proportion of goal states where $\pi^{\text{GPI}}_K$ matches $\pi^*_K$, as a function of basis size $K$. The linear trend emphasizes that the optimal performance for GPE and GPI increases with basis size.

Figure 3 shows the options induced by $\pi^{\text{GPI}}_K$ for the same $K$ components. As $K$ increases, both $\pi^*_K$ and $\pi^{\text{GPI}}_K$ converge toward optimality. Note that for $K = \{5, 25\}$, the policies $\pi^*_K$ and $\pi^{\text{GPI}}_K$ do not align perfectly. Figure 4a visualizes $\pi^{\text{GPI}}_5$ and the GPI step behind it. While $\pi^*_5$ exclusively guides the agent to terminal state $\top 2$ (where the reconstructed reward $R_5$ is maximum), $\pi^{\text{GPI}}_5$ additionally terminates in state $\top 1$. This discrepancy arises because none of the first five eigenoptions (in either direction) take the agent from $\top 1$ to $\top 2$, thus restricting the representational capacity of the composed GPI policy. The single downward action near $\top 2$ is optimal and originates from the first eigenoption, which is an *always-terminate* policy. All other actions direct the agent from $\top 2$ to $\top 1$, causing a deviation from $\pi^*_5$.

To quantify this deviation across all possible goals, Figure 4b shows the proportion of goal positions where $\pi^{\text{GPI}}_K$ exactly matches $\pi^*_K$. As $K$ increases from 1 to $|\mathcal{S}|$, the match rate improves, with an average accuracy of 93.14%. The linear trend confirms that a richer basis yields more accurate policies. These results suggest that eigenoptions, combined with GPE and GPI, have the potential to provide an effective mechanism for solving a wide variety of goal-reaching tasks. However, to make stronger claims on the ability of eigenoptions to be a global and compact basis for any environment requires further experiments in more diverse environments and complex reward settings. Moreover, the fact that $\pi^{\text{GPI}}_K$ does not always match $\pi^*_K$ indicates that the current eigenoptions do not yet form an optimal option basis. If they did, GPI would recover the optimal policy for any goal and any value of $K$.

## 4 Laplacian Keyboard

We introduce the Laplacian Keyboard (LK), which uses eigenoptions as the basis in the OK framework, providing a more expressive alternative to GPE and GPI with eigenoptions. While GPE and GPI can produce diverse behaviors given a rich set of eigenoptions, LK enables a meta-policy to flexibly combine a smaller set of options, resulting in a broader and more adaptable range of behaviors. This allows the LK to solve tasks that GPE and GPI could not solve using the same bases.

To understand LK's effectiveness, we evaluate its performance on similar goal-reaching tasks. Eigenoptions are computed using tabular Q-learning (Watkins & Dayan, 1992), while the LK agent is trained using TD3 (Fujimoto et al., 2018). The actor and critic networks are both 2-layer feed-forward networks with 128 hidden units per layer. The input to the actor is the $K$-dimensional eigenvector of the current state, and the output is a $2 \times K$-dimensional weight vector used to combine the corresponding eigenoptions. Both networks are optimized using Adam (Kingma & Ba, 2015), with a step size of 0.001 and a batch size of 512. We find that performance improves when the actor is updated less frequently—specifically, once every 20 critic updates, rather than the default interval of 2 used in standard TD3.

For evaluation, we select five goal positions and train a separate LK agent for each goal. Each agent is trained for 20,000 environment steps across twenty random seeds, with episodes initialized from random starting states and capped at 100 timesteps. To assess performance, we measure the average number of steps the trained agent takes to reach its target goal from all possible starting positions, and then average this metric across all five goal positions. We report the mean and the standard error of the average episodic length for varying values of $K \in \{5, 15, 25, 35, 45, 55\}$, and we compare the results for both LK and GPI.

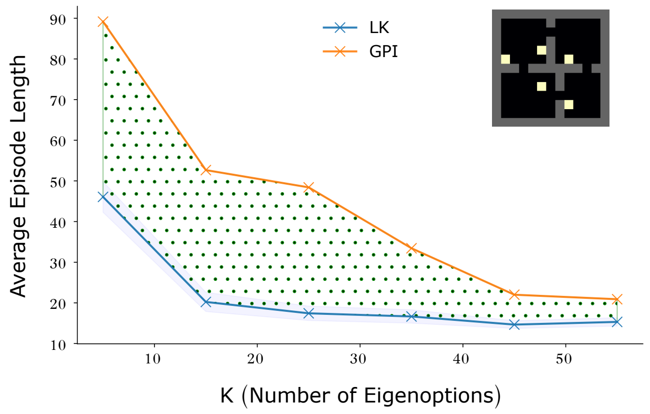

Figure 5: **Comparison of performance of LK vs GPI:** The green dotted region highlights the performance gap between GPI and LK, where LK achieves lower average episode lengths, indicating improved efficiency. The blue shaded area represents the standard error of LK's mean performance, averaged over all starting positions and five goal positions across 20 random seeds. The inset displays the five individual goal positions.

Figure 5 shows the average duration the agent takes to reach the goal positions, and it is clear that the LK consistently outperforms GPI across different values of $K$. The green dotted region highlights the performance gap, marking the area between the average episode lengths achieved by LK and GPI. This visual cue highlights the degree to which the LK outperforms GPI in goal-reaching tasks. This suggests that LK is more effective, even with fewer eigenoptions, highlighting its ability to learn and solve tasks more efficiently.

Figure 6 illustrates the dynamic capabilities of LK in comparison to GPE and GPI, particularly highlighting its advantage when using a small option basis. 6a shows the GPI policy, $\pi_{10}^{\text{GPI}}$, for the reward function $R_{10}$ (refer to Figure 2). In this case, $\pi_{10}^{\text{GPI}}$ coincides exactly with the optimal policy

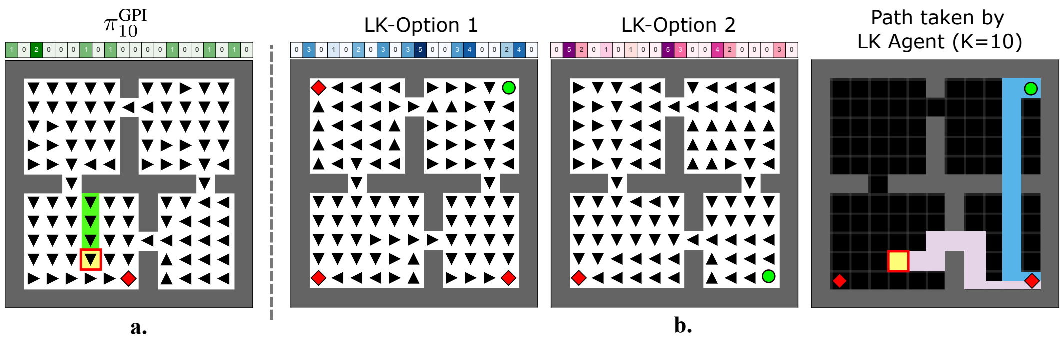

Figure 6: **Understanding the improvement of LK over GPI:** (a) This panel shows the policy $\pi_{10}^{\text{GPI}}$ for the reward function defined in Figure 2. The agent successfully reaches the goal (yellow square) only when initialized in one of the green-highlighted states; from all other initial states, it fails to reach the goal and terminates elsewhere. (b) This panel illustrates the behavior of the LK using the same set of 20 eigenoptions. When the agent (green circle) is initialized in the top-right state, LK first outputs a set of weights to combine the 20 eigenoptions to generate LK-Option 1, which takes the agent to the bottom-right state. At that point, a new set of weights is computed to generate LK-Option 2, which then guides the agent to the goal. The rightmost plot visualizes this two-step trajectory: the blue and pink paths show the agent's progress under LK-Option 1 and LK-Option 2, respectively. The weights used to generate the LK-Options are rounded to the closest integers and visualized above each option.

$\pi_{10}^*$. However, the agent can only reach the goal if initialized in one of the three green-highlighted states. Otherwise, GPE and GPI yield a static behavior that takes the agent to a non-goal state and terminates there, failing to solve the task. In contrast, LK can yield dynamic behaviors by stitching different options to solve the same task (see Figure 6b). For example, when initialized in the top-right state, LK first generates 20 weights to combine the base eigenoptions to produce LK-Option 1. Following LK-Option 1, the agent reaches the bottom-right state and terminates. The control is passed back to the LK, which then generates a new set of weights to construct LK-Option 2. The agent now follows the newly generated option to reach the goal state. This ability to generate dynamic behaviors enables LK to outperform naive GPE and GPI when using a small option basis.

## 5 Related Work

**Option Keyboard:** Prior research on learning option bases within the OK framework has explored various methodologies. Barreto et al. (2019) introduces the OK and uses options that were derived from manually engineered reward features. In contrast, Carvalho et al. (2023) focused on directly learning these reward features. However, working in a transfer RL setting, the learned reward features are limited by the choice of training tasks. More recently, Alegre et al. (2022; 2023; 2025) proposed a method for learning an optimal basis for a given MDP, applicable to any specified reward features. While this method directly addresses a key limitation of our current study—namely, how to construct an optimal basis using the eigenvectors of the graph Laplacian as reward features—it suffers from exponential complexity with respect to the number of reward features. As a result, their experiments are limited to environments with very small observation/reward feature spaces.

Another common thread across these recent works (Alegre et al., 2022; 2023; 2025; Carvalho et al., 2023) is the use of Successor Features (Barreto et al., 2017), which enable efficient evaluation of policies under varying reward functions by decoupling environment dynamics from task-specific rewards. This decoupling makes it possible to reuse learned dynamics when adapting to new tasks defined by different reward combinations. Building on this foundation, Borsa et al. (2019) extends the idea further by integrating Successor Features into the Universal Value Function Approximators framework (Schaul et al., 2015), thereby achieving broader generalization across the task space.

**Successor Measure:** Unlike the aforementioned methods that rely on pre-defined reward features, recent works such as the one by Touati & Ollivier (2021) and Agarwal et al. (2024) aim to learn features that represent the successor measure of any given policy. Specifically, Touati & Ollivier (2021) learn two distinct representation modules, $F$ and $B$, guaranteeing near-optimal performance. Under certain constraints, the representation $B$ is known to converge to the eigenvectors of the graph Laplacian (Blier et al., 2021), which establishes a close connection to our current work. While the Option Keyboard framework requires a large number of options to ensure optimality, their method overcomes this by using parameterized models that support efficient behavior generation. The scalability of this framework has been demonstrated in Tirinzoni et al., where a high-dimensional humanoid agent with a 358-dimensional state space was successfully controlled to perform zero-shot behaviors.

## 6 Discussion

In this work, we propose using the eigenvectors of an environment's graph Laplacian as reward features for learning an option basis. As a proof of concept, we evaluate the Laplacian Keyboard, which leverages eigenoptions as a basis within the Option Keyboard framework. In gridworld experiments, we show that eigenoptions serve as an effective basis: applying GPE and GPI with these eigenoptions can recover the optimal policy for goal-reaching tasks. Furthermore, we demonstrate that the Laplacian Keyboard achieves similar performance while relying on a substantially smaller set of basis options.

### 6.1 Future Work

**Function Approximation:** In this study, we computed eigenoptions and evaluated them for different reward functions using tabular Q-learning. However, this approach does not scale to environments with large or continuous state spaces, where function approximation becomes necessary. Future work will address this limitation by leveraging recent methods for approximating the eigenfunctions of the graph Laplacian using neural networks (Gomez et al., 2023). Additionally, more efficient learning and evaluation of eigenoptions can be pursued by techniques used in prior work on successor features and option composition (Barreto et al., 2017; Carvalho et al., 2023). Finally, building on prior evidence of scalability (Gomez et al., 2023; Klissarov & Machado, 2023), we plan to extend our approach to more complex environments.

**Optimal basis of eigenoptions:** Our experiments demonstrate that while eigenoptions provide a strong basis for behavior composition, they are not optimal. As shown in Section 3, the policies $\pi^*_K$ and $\pi^{GPI}_K$ do not always align, indicating that the current basis does not fully capture the optimal structure. Recent works (Alegre et al., 2022; 2025) have proposed methods for constructing an optimal basis given knowledge of the reward features. Here we argued that the eigenvectors of the graph Laplacian should be such reward features. Building on these insights, we hypothesize that an optimal basis of eigenoptions would include options trained from weighted combinations of eigenvectors rather than one-hot / individual eigenvectors. As part of future work, we aim to extend our approach by incorporating these ideas to construct a universal and optimal basis of options to solve any given task in an environment.

## References

Siddhant Agarwal, Harshit Sikchi, Peter Stone, and Amy Zhang. Proto Successor Measure: Representing the Space of All Possible Solutions of Reinforcement Learning. *arXiv preprint arXiv:2411.19418*, 2024.

Lucas Nunes Alegre, Ana L. C. Bazzan, and Bruno C. Da Silva. Optimistic Linear Support and Successor Features as a Basis for Optimal Policy Transfer. In *International Conference on Machine Learning*, pp. 394–413, 2022.

Lucas Nunes Alegre, Ana L. C. Bazzan, Diederik M. Roijers, Ann Nowé, and Bruno C. da Silva. Sample-Efficient Multi-Objective Learning via Generalized Policy Improvement Prioritization. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 2003–2012, 2023.

Lucas Nunes Alegre, Ana L. C. Bazzan, André Barreto, and Bruno C. Da Silva. Constructing an Optimal Behavior Basis for the Option Keyboard. *arXiv preprint arXiv:2505.00787*, 2025.

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.

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.

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.

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

Diana Borsa, Andre Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal Successor Features Approximators. In *International Conference on Learning Representations*, 2019.

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.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, pp. 1587–1596, 2018.

Diego Gomez, Michael Bowling, and Marlos C. Machado. Proper Laplacian Representation Learning. In *International Conference on Learning Representations*, 2023.

Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering Options for Exploration by Minimizing Cover Time. In *International Conference on Machine Learning*, pp. 3130–3139, 2019.

Yuu Jinnai, Jee Won Park, Marlos C. Machado, and George Konidaris. Exploration in Reinforcement Learning with Deep Covering Options. In *International Conference on Learning Representations*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Martin Klissarov and Marlos C. Machado. Deep Laplacian-Based Options for Temporally-Extended Exploration. In *International Conference on Machine Learning*, pp. 17198–17217, 2023.

Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. A Laplacian Framework for Option Discovery in Reinforcement Learning. In *International Conference on Machine Learning*, pp. 2295–2304, 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.

Sridhar Mahadevan. Proto-Value Functions: Developmental Reinforcement Learning. In *International Conference on Machine Learning*, pp. 553–560, 2005.

Sridhar Mahadevan and Mauro Maggioni. Proto-Value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research*, 8(10), 2007.

Stéphane Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 1999.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015.

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.

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.

Ahmed Touati and Yann Ollivier. Learning One Representation to Optimize All Rewards. In *Neural Information Processing Systems*, pp. 13–23, 2021.

Christopher J. C. H. Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.