Learning Spatially Refined Sub-Policies for Temporal Task Composition in Continuous RL

Tim van Gelder[0009−0006−5498−9936] and Herke van Hoof[0000−0002−1583−3692]

AMLab, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands

Abstract. Traditional Reinforcement Learning (RL) methods can solve complex, long-horizon tasks but struggle to generalize to new non-Markov tasks without retraining. Recent compositional approaches address this by learning a set of sub-policies that can be composed at test time to solve unseen, temporally extended tasks—formulated as finite state automata (FSA)—in a zero-shot manner. However, existing methods are typically restricted to discrete domains or suffer from sub-optimality in stochastic environments. We address both limitations by extending compositional RL to continuous state spaces using Radial Basis Function (RBF) features and a novel regression-based value iteration algorithm that enables optimal composition over learned sub-policies. Our method supports more globally efficient planning in environments with spatially extended goals and achieves optimal behavior in both deterministic and stochastic settings, outperforming prior compositional baselines.

Keywords: Compositional Strategies · Reinforcement Learning · Temporal Composition

1 Introduction

Traditional Reinforcement Learning (RL) methods can solve complex, long-horizon tasks, even in environments that closely resemble the real world. However, they often learn a single policy for a single task, without considering any temporal or hierarchical structure. This limits generalization to tasks or environments that are slightly different. While re-training for each new task is possible, this is costly and does not exploit shared structure across tasks.

Compositional Strategies (CS) aim to address challenges such as scalability, sample inefficiency, and poor generalization by decomposing tasks, policies, or value functions into reusable components. Rather than learning a new policy from scratch, they recombine previously learned behaviors, enabling transfer, faster learning, and greater interpretability.

Hierarchical Reinforcement Learning (HRL), a class of CS, addresses limitations of standard RL by decomposing long-horizon tasks into sub-tasks solved by lower-level policies, which are coordinated by a higher-level policy. This temporal abstraction shortens horizons, improves credit assignment, simplifies sub-task

learning, and supports more efficient exploration. As a result, HRL often outperforms flat RL in domains such as continuous control and long-horizon games [9]. A central HRL framework is the options framework [12], which allows agents to act not only with primitive actions but also with temporally extended behaviors, or options. Each option is defined by an initiation set, an internal policy, and a termination condition. Options enable planning over variable-length action sequences, facilitating behavior reuse and reducing decision-making depth.

However, one of the major difficulties with HRL and compositional strategies is that the learned sub-components may not be globally optimal. That is, each sub-policy may only be locally optimal for the sub-task(s) it was trained on since it does not consider the global goal or context. Combining these sub-policies to solve the full, high-level task may then lead to a suboptimal overall solution, also called recursive optimality [4]. Stronger forms of optimality, such as hierarchical or even global optimality, are often more desirable.

Other approaches improve generalization by conditioning policies directly on goal specifications. Universal Value Function Approximators (UVFAs) extend value functions with goal descriptors, enabling transfer across goal configurations without retraining [10]. However, they are typically limited to simple goal formulations—such as target states or coordinates—and cannot capture richer temporal or logical structures.

Reward Machines (RMs) address this limitation by decomposing reward functions into Finite State Automata that encode temporal and logical dependencies between sub-goals [5, 6]. By exposing reward structure, RMs allow agents to reason over loops, conditionals, and interleavings, capturing both Markovian and some non-Markovian objectives. Q-learning over RMs accelerates learning compared to standard Q-learning or hierarchical methods, but it requires a value function per RM state and a manually defined automaton for each task, which hinders scalability.

To overcome these constraints, recent methods condition policies directly on task specifications expressed in Linear Temporal Logic (LTL) [15, 7]. Conditioning on symbolic instructions enables zero-shot generalization to unseen LTL tasks, but these approaches rely on large end-to-end neural models, limiting interpretability and scalability in complex, continuous domains.

Thus, sometimes, it is useful to take a more structured approach. Structured alternatives offer advantages in transparency, verifiability, and instructability. They allow agents to follow composite instructions not seen during training and yield more predictable behavior compared to black-box models.

Several more structured compositional strategies exist, such as the Logical Options Framework (LOF) and Skill Machines [2, 14]. LOF's composed policies are hierarchically optimal but typically not globally optimal in the case of stochastic dynamics since each option deterministically targets its sub-goal. At the same time, by extending the Boolean Task Algebra framework [13], Skill Machines support not only temporal but also spatial composition. Although their method is shown to be *satisficing*, it may yield suboptimal solutions since the final high-level policies are recursively rather than globally optimal.

Kuric et al. [8] address these limitations by combining Optimistic Linear Support (OLS) and Successor Features to build a convex coverage set of composable sub-policies. Their method guarantees globally optimal solutions for non-Markovian, high-level tasks, even under stochastic dynamics. However, it is restricted to discrete domains, where sub-goals are defined as individual states and planning over exit states remains tractable. Extending this to continuous domains is challenging, both in defining spatial sub-goals and in handling unbounded sets of exit states.

In this work, we extend structured compositional methods to continuous domains by building on the SF-OLS framework. Our main contributions are:

- We define spatial sub-goals using Radial Basis Functions (RBFs), enabling sub-policy learning over continuous regions.
- We introduce a regression-based value iteration method to compose sub-policies for zero-shot generalization to tasks defined by arbitrary FSAs.
- We empirically show that our method outperforms LOF and other baselines in environments with stochastic dynamics and spatially extended goals.

2 Background and Notation

2.1 Preliminaries

We begin by briefly reviewing standard reinforcement learning (RL) terminology and concepts. A Markov Decision Process (MDP) is defined as a tuple $M = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p(s' \mid s, a)$ the transition probability of reaching state s' from state s via action a, $r(s, a, s')$ the reward function, and $\gamma \in [0, 1)$ the discount factor that weights future rewards.

The agent aims to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted return:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t, S_{t+1}) \right]. \tag{1}$$

The action-value function $Q^\pi(s, a)$ gives the expected return from (s, a) under π, and satisfies the Bellman equation:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s, a, s') + \gamma \mathbb{E}_{a' \sim \pi(\cdot|s')} Q^\pi(s', a') \right]. \tag{2}$$

Furthermore, the optimal Q-function satisfies the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right], \tag{3}$$

from which an optimal policy π^* can be derived [11].

2.2 The Successor Features Framework

Successor Features (SFs) decompose the reward and value functions into environment features and a task-specific weight vector. The reward is assumed to be a linear product of a feature vector and weight vector:

$$r(s, a, s') = \boldsymbol{\phi}(s, a, s')^\top \mathbf{w}, \tag{4}$$

where $\boldsymbol{\phi}(s, a, s') \in \mathbb{R}^d$ are features of the transition and $\mathbf{w} \in \mathbb{R}^d$ encodes task-specific preferences [3]. Under this formulation, the action-value function becomes:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_{i+1}^T \mid (S_t = s, A_t = a) \right] \mathbf{w} \\ &= \boldsymbol{\psi}^\pi(s, a)^T \mathbf{w}, \end{aligned} \tag{5}$$

where $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t, a_t, s_{t+1})$. The vector $\boldsymbol{\psi}^\pi(s, a)$ is the *Successor Feature*, representing the expected discounted feature vector under policy π:

$$\boldsymbol{\psi}^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_i \mid (S_t = s, A_t = a) \right]. \tag{6}$$

This representation naturally leads to Generalized Policy Improvement (GPI), which allows for transfer over a family of Markov Decision Processes (MDPs) [3]. If we consider all MDPs in which all components are the same except the reward functions induced by different weight vectors $\mathbf{w}$, we can define that as a family of MDPs:

$$\mathcal{M}^\phi \equiv \{M(\mathcal{S}, \mathcal{A}, p, r, \gamma) \mid r_{\mathbf{w}}(s, a, s') = \boldsymbol{\phi}(s, a, s')^T \mathbf{w}, \, \forall \mathbf{w} \in \mathbb{R}^d\}.$$

Given policies $\mathit{\Pi} = \{\pi_1, \ldots, \pi_n\}$ learned on tasks in $\mathcal{M}^\phi$, a GPI policy for a new task with reward weights $\mathbf{w}'$ is defined as:

$$\pi_{GPI}(s) \in \operatorname*{argmax}_a \max_{\pi \in \mathit{\Pi}} Q^\pi_{\mathbf{w}'}(s, a), \tag{7}$$

and is guaranteed to perform at least as well as the best policy in $\mathit{\Pi}$ on the new task [3, 8].

2.3 Convex Coverage Set

A key question is which task vectors $\mathbf{w}_1, \ldots, \mathbf{w}_n$ to train policies for, such that the resulting set $\mathit{\Pi}$ leads to strong GPI performance on unseen tasks $\mathbf{w}'$. Alegre et al. [1] address this by converting the MDP family M^ϕ into a multi-objective MDP (MOMDP) and applying multi-objective RL to compute a Convex Coverage Set (CCS) of non-dominated successor features:

$$\begin{aligned} CCS &= \{\boldsymbol{\psi}^\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \boldsymbol{\psi}^{\pi'}, \, \boldsymbol{\psi}^\pi \cdot \mathbf{w} \geq \boldsymbol{\psi}^{\pi'} \cdot \mathbf{w}\} \\ &= \{\boldsymbol{\psi}^\pi \mid \exists \mathbf{w} \text{ s.t. } \forall \pi', \, V^\pi_{\mathbf{w}} \geq V^{\pi'}_{\mathbf{w}}\}. \end{aligned} \tag{8}$$

We define Π_{CCS} as a set of policies corresponding to SFs in the CCS, i.e., for every $\boldsymbol{\psi}^{\pi} \in CCS$, there exists a $\pi' \in \Pi_{CCS}$ such that $\boldsymbol{\psi}^{\pi'} = \boldsymbol{\psi}^{\pi}$. Each policy in Π_{CCS} is optimal for at least one task $\mathbf{w}$, and for any new task vector $\mathbf{w}'$, an optimal policy exists in Π_{CCS}. As a result, applying GPI to a complete CCS yields an optimal policy for all $\mathbf{w} \in \mathcal{W}$ [1].

To construct Π_{CCS}, we can use an algorithm like Successor Features Optimistic Linear Support (SF-OLS), an extension of the Optimistic Linear Support algorithm to the SF framework [1]. SF-OLS incrementally builds the CCS by proposing new task weights $\mathbf{w}$, selected from a priority queue based on expected improvement. For each selected $\mathbf{w}$, a new policy is trained on the induced MDP $M_{\mathbf{w}}$ and added to the set. This process continues until convergence.

2.4 Finite State Automata Tasks and Product MDP

Complex temporal tasks can be specified using propositional logic and Finite State Automata (FSA). We assume a set of boolean propositional symbols $\mathcal{P}$ representing (sub-)goals in the environment. Each symbol is associated with one or more low-level states $\varepsilon \in \mathcal{E}$ in the MDP $\mathcal{M}$, referred to as exit states. Entering an exit state produces a truth assignment in $2^{\mathcal{P}}$ under a known observation mapping $\mathcal{O} : \mathcal{S} \to 2^{\mathcal{P}}$ that is known to the agent [8].

A high-level task is then defined as an FSA $\mathcal{F} = \langle \mathcal{U}, u_0, \mathcal{T}, \mathcal{L}, \delta \rangle$, where $\mathcal{U}$ is a finite set of FSA states (distinct from MDP states), u_0 is the initial FSA state, $\mathcal{T} \subseteq \mathcal{U}$ is the set of terminal (accepting) FSA states, $\mathcal{L} : \mathcal{U} \times (\mathcal{U} \cup \mathcal{T}) \to 2^{\mathcal{P}}$ is a labeling function that maps FSA transitions to truth values for the propositional symbols, and $\delta : \mathcal{U} \to \{0, 1\}$ is a sparse reward function non-zero only at terminal FSA states [8]. Each transition corresponds to a propositional symbol and is enabled by the associated exit states; all other symbols cause the FSA to remain in the current state (see Fig. 1 or App. B for a graphical representation). Intuitively, the FSA captures progress in the high-level task, not every transition in the underlying MDP. A transition between FSA states only occurs when the agent reaches exit states whose propositional assignments satisfy the condition for that transition. Exit states that do not satisfy any such condition leave the FSA unchanged, reflecting no progress toward task completion.

Combining a high-level FSA task with the low-level MDP family yields a product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^{\phi}$ with augmented state space $\mathcal{U} \times \mathcal{S}$ [8]. We assume zero intermediate reward and a terminal reward $R = 1$ only in FSA terminal states $u_T \in \mathcal{T}$. Policies over this product MDP take the form $\mu : \mathcal{U} \times \mathcal{S} \to \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ is the probability simplex over $\mathcal{A}$. The corresponding augmented Q-function is:

$$Q^{\mu}(u, s, a) = \mathbb{E}_{\mu}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \mathcal{R}_i \,\middle|\, U_t = u, S_t = s, A_t = a\right]. \tag{9}$$

2.5 Solving the Product MDP with SF-FSA-VI

Since the agent's objective is to reach an FSA terminal state u_T to receive reward, it suffices to find optimal weight vectors $\mathbf{w}^*(u)$ for each FSA state u to solve the product MDP. These weights summarize the agent's desired behavior in each FSA state. Assuming we have access to the CCS, we can use GPI to retrieve an optimal policy for any $\mathbf{w}(u) \in \mathcal{W}$. The resulting optimal Q-function is:

$$Q^*_{\mathbf{w}^*}(u, s, a) = \max_{\pi \in \Pi_{CCS}} \mathbf{w}^*(u)^T \boldsymbol{\psi}^\pi(s, a), \tag{10}$$

where $\mathbf{w}^*(u)$ reflects the globally optimal objective for being in FSA state u given the full FSA structure.

To compute $\mathbf{w}^*(u)$, Kuric et al. [8] propose SF-FSA-VI, a dynamic programming algorithm similar to value iteration. They allocate a single weight element $\mathbf{w}_j$ and feature element $\boldsymbol{\phi}_j$ to each unique exit state, where the features are simple indicator functions that are 1 if the agent is in the respective exit state ε_j and 0 otherwise. To obtain $\mathbf{w}^*(u)$, $\forall u \in \mathcal{U}$, they use the following update rule:

$$\begin{aligned} \mathbf{w}_j^{k+1}(u) &= \max_a Q^*_{\mathbf{w}}(\tau(u, \mathcal{O}(\varepsilon_j)), \varepsilon_j, a) \\ &= \max_{a,\pi} \mathbf{w}^k(\tau(u, \mathcal{O}(\varepsilon_j)))^T \boldsymbol{\psi}^{\boldsymbol{\pi}}(\varepsilon_j, a), \end{aligned} \tag{11}$$

where $\tau(u, \mathcal{O}(\varepsilon_j)) \in \mathcal{U}$ is the FSA state that results from achieving the valuation $\mathcal{O}(\varepsilon_j)$ in u. Thus, each $\mathbf{w}(u)$ encodes the Q-values for all augmented state pairs (u, ε) consisting of an exit state.

3 Method

3.1 Learning by Defining Features over the Continuous Domain

Extending SF-OLS and SF-FSA-VI to continuous spaces introduces key challenges. In particular, sub-goals in continuous domains correspond to infinitely many exit states, making the discrete approach of assigning one feature per exit state intractable. In discrete settings, goal-related exit states are finite, allowing each to be assigned a feature index and corresponding weight in $\boldsymbol{\phi}$ and $\mathbf{w}$. In contrast, in continuous domains, the infinite number of such states would lead to unbounded feature and weight vectors and make planning over all possible exit states infeasible.

To enable smooth generalization across space, we define features using Radial Basis Functions (RBFs) over the continuous domain. These features softly activate based on spatial proximity to predefined centers. Formally, each feature at index i in the feature vector $\boldsymbol{\phi}$ is defined as:

$$\boldsymbol{\phi}_i(s) = \delta(s \in \mathcal{E}_{p_i}) \cdot \exp\left(-\frac{(s_x - x_i)^2 + (s_y - y_i)^2}{2\sigma^2}\right), \tag{12}$$

where (s_x, s_y) is the current 2D state, (x_i, y_i) the RBF center, σ the bandwidth, and $\delta(s \in \mathcal{E}_{p_i})$ an indicator for whether s is an exit state for p_i and p_i is the

propositional variable linked to the feature at index i in the feature vector $\boldsymbol{\phi}$. Using this formulation, we can define one or more RBF features for each propositional variable (see App. D, Fig. 8).

Due to the multiplication with the indicator function, each feature is associated with a single propositional symbol $p \in \mathcal{P}$ and is non-zero only on exit states corresponding to that goal type. Feature placement can be done manually—leveraging prior knowledge about where goal regions are located—or automatically, for example, by defining a uniform grid of RBF centers.

In principle, any feature function over the continuous domain can be used to construct the feature vector $\boldsymbol{\phi}$. For instance, we validated that Fourier features can be integrated into our framework and exhibit comparable qualitative behavior. However, for the experiments presented in this work, we focus solely on RBF features to maintain consistency.

3.2 Approximating Values during Planning

Sampling a Finite Set of Exit States In continuous environments, each feature $\boldsymbol{\phi}_i$ corresponds to infinitely many exit states, complicating the value iteration procedure from [8]. To make planning tractable, we approximate each region $\mathcal{E}_{p_i}$ by selecting a finite set of representative exit states. Concretely, we place a uniform grid over the state space and select the grid cell centers that lie within $\mathcal{E}_{p_i}$ (one could also use random sampling as an alternative). This enables estimating the value of spatial sub-regions associated with each feature $\boldsymbol{\phi}_i$ and its corresponding propositional symbol p_i.

Augmented Features and Weight Vectors. To enable planning in continuous domains with finite exit state approximations, we adopt a unified representation of the value function. While prior work [8] uses a separate task vector $\mathbf{w}(u)$ for each FSA state u, we instead concatenate all task vectors into a single augmented weight vector $\mathbf{w}$ and define a corresponding augmented feature vector $\boldsymbol{\phi}(\tilde{s})$, where $\tilde{s} = (s, u)$ combines the environment state and FSA state.

The augmented feature vector is defined by stacking gated copies of the base features $\boldsymbol{\phi}(s)$:

$$\boldsymbol{\phi}(\tilde{s}) = \begin{bmatrix} \delta(u_1) \cdot \boldsymbol{\phi}(s) \\ \delta(u_2) \cdot \boldsymbol{\phi}(s) \\ \delta(u_3) \cdot \boldsymbol{\phi}(s) \\ \vdots \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} \mathbf{w}(u_1) \\ \mathbf{w}(u_2) \\ \mathbf{w}(u_3) \\ \vdots \end{bmatrix}. \tag{13}$$

Since only one $\delta(u_i)$ is active at a time, this construction selects the relevant task vector $\mathbf{w}(u)$ and recovers the value estimate $R_{\mathbf{w}}(s) = \boldsymbol{\phi}(s)^T \mathbf{w}$.

Augmented Successor Features with FSA Structure. To extend our augmented representation to successor features (SFs), we define a structured vector that respects the FSA's dynamics. As before, we concatenate SFs across FSA

states, but now incorporate the structure of transitions enabled by specific propositional symbols.

Let $\rho : \mathcal{U} \times \mathcal{U} \to \mathcal{P} \cup \{\emptyset\}$ map each FSA state pair (u_i, u_j) to the unique propositional symbol that enables the transition, or $\emptyset$ if no such transition exists.[1] For each plausible transition from u_i to any other FSA state $u_0, \ldots, u_j$, we define a masked SF vector $\boldsymbol{\psi}_j^\pi(u_i, s, a)$ by masking the base SF vector $\psi^\pi(s, a)$ using two conditions:

1. $\delta\big((u_i, u_j) \in E\big)$, which is 1 if the transition from u_i to u_j exists, and
2. $\delta\big(p_k = \rho(u_i, u_j)\big)$, which is 1 if the k-th feature corresponds to the unique symbol that enables the transition.

This leads to the following definition of our masked SFs:

$$\boldsymbol{\psi}_j^\pi(u_i, s, a) = \begin{bmatrix} \delta\big((u_i, u_j) \in E\big) \cdot \delta\big(p_1 = \rho(u_i, u_j)\big) \cdot \psi_1^\pi(s, a) \\ \vdots \\ \delta\big((u_i, u_j) \in E\big) \cdot \delta\big(p_d = \rho(u_i, u_j)\big) \cdot \psi_d^\pi(s, a) \end{bmatrix} \in \mathbb{R}^d. \tag{14}$$

We then define the full augmented vector by stacking the masked sub-vectors corresponding to all possible transitions from u_i to all FSA states:

$$\tilde{\boldsymbol{\psi}}^\pi(u_i, s, a) = \begin{bmatrix} \boldsymbol{\psi}_0^\pi(u_i, s, a) \\ \boldsymbol{\psi}_1^\pi(u_i, s, a) \\ \vdots \\ \boldsymbol{\psi}_{|\mathcal{U}|-1}^\pi(u_i, s, a) \end{bmatrix} \in \mathbb{R}^{|\mathcal{U}| \cdot d}. \tag{15}$$

This structured construction enables us to associate each FSA state with the specific SFs that contribute to its possible transitions. By concatenating the masked sub-vectors across all potential transitions, we obtain a unified representation $\tilde{\boldsymbol{\psi}}^\pi(u_i, s, a)$ that preserves the FSA's compositional structure. This formulation allows us to solve for the global weight vector $\mathbf{w}$ through a single regression step during planning, capturing task dynamics across all FSA states in a consistent manner.

Planning Model. To enable high-level planning over FSA-structured tasks, we model the value function as a linear combination of augmented state features:

$$V(\tilde{s}) = \boldsymbol{\phi}(\tilde{s})^\top \mathbf{w}, \tag{16}$$

where $\tilde{s} = (s, u)$ is the combined MDP and FSA state, and $\mathbf{w}$ is the global augmented weight vector shared across tasks.

[1] A more flexible formulation might allow multiple propositional variables to enable a single transition. However, our current notation does not limit the expressivity of our FSAs: in such cases, we split the transition into multiple transitions (and intermediate FSA states), one for each proposition, and later merge the resulting states back into a single FSA state.

Following standard value iteration, we define a Bellman-like update: $V(\tilde{s}) \leftarrow \max_\pi \mathbb{E}\left[R + \gamma^k V(\tilde{s}')\right]$, where π is a sub-policy, and k is the number of low-level steps before the FSA transitions. Since we are using function approximation, directly assigning this target to $V(\tilde{s})$ is not feasible. Instead, we compute a regression target $y(\tilde{s})$, where we derive the following sequence for non-terminal states $u \notin \mathcal{T}$:

$$\begin{aligned} y(\tilde{s}) &= \max_\pi \mathbb{E}\left[R + \gamma^k V(\tilde{s}')\right] = \max_\pi \mathbb{E}\left[\gamma^k V(\tilde{s}')\right] = \max_\pi \mathbb{E}\left[\gamma^k \boldsymbol{\phi}(\tilde{s}')^\top \mathbf{w}\right] \\ &= \max_\pi \left(\mathbb{E}\left[\gamma^k \boldsymbol{\phi}(\tilde{s}')\right]^\top \mathbf{w}\right) = \max_\pi \mathbb{E}_{a\sim\pi}\left[\tilde{\boldsymbol{\psi}}^\pi(\tilde{s}, a)\right]^\top \mathbf{w}, \end{aligned} \tag{17}$$

where $\tilde{\boldsymbol{\psi}}^\pi(\tilde{s}, a)$ denotes the augmented successor feature vector for state $\tilde{s}$ and action a under policy π. Conceptually, we are equating the features of the next augmented state $\tilde{s}'$, discounted by the number of lower-level steps k it took to get there, to the SFs of the current augmented state $\tilde{s}$, where we consider only the augmented states containing an exit state $\tilde{s} = (u, \varepsilon)$. This is consistent with the definition of SFs, which represent the expected discounted sum of features encountered when following a given policy from the current state. The SFs and the CCS over which we maximize are learned as described in Sections 2.2 and 2.3, following the approaches in [1] and [8].

Since the environment gives a reward only when the task is completed (i.e., $u \in \mathcal{T}$), we define the regression target:

$$\hat{y}(\tilde{s}) = \begin{cases} 1, & \text{if } \tilde{s} \text{ is terminal } (u \in \mathcal{T}) \\ \max_\pi \mathbb{E}_{a\sim\pi}\left[\tilde{\boldsymbol{\psi}}^\pi(\tilde{s}, a)\right]^\top \mathbf{w}, & \text{otherwise.} \end{cases} \tag{18}$$

Weight Learning via Regression. To learn the weight vector $\mathbf{w}$, we fit the approximation $\boldsymbol{\phi}(\tilde{s})^\top \mathbf{w} \approx \hat{y}(\tilde{s})$, where $\hat{y}(\tilde{s})$ is the target from Equation (18). At each planning iteration, we collect feature vectors and corresponding targets for the set of augmented exit states and construct:

- a feature matrix $\boldsymbol{\Phi}$, where each row corresponds to a feature vector $\boldsymbol{\phi}(\tilde{s})^\top$, and
- a target vector $\hat{\mathbf{y}}$, whose entries are the corresponding target values $\hat{y}(\tilde{s})$.

We then solve the following regularized linear system using ridge regression:

$$\begin{aligned} &\boldsymbol{\Phi}\mathbf{w} \approx \hat{\mathbf{y}} \\ &\mathbf{w}^* = \left(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \epsilon I\right)^{-1} \boldsymbol{\Phi}^\top \hat{\mathbf{y}}, \end{aligned} \tag{19}$$

where $\epsilon > 0$ is a small regularization constant that ensures numerical stability and helps prevent overfitting.

This procedure resembles approximate dynamic programming, but rather than predicting the value of low-level actions, it estimates the value of high-level actions—namely, completing sub-goals and advancing through the FSA. See Appendix C for pseudo-code of our planning algorithm.

Execution. After having estimated $\mathbf{w}$, we extract the sub-vector $\mathbf{w}(u)$ corresponding to the current FSA state u from the full augmented weight vector $\mathbf{w}$ and perform GPI during execution:

$$\pi_{\text{GPI}}(u, s) \in \arg\max_a \max_{\pi \in \Pi} \boldsymbol{\psi}^{\pi}(s, a)^{\top} \mathbf{w}(u). \tag{20}$$

4 Experiments

4.1 Environments and Tasks

We evaluate our method in a continuous version of the Office environment introduced by [5], which involves three propositional sub-goals: $\mathcal{P} = \{$☕, ✉, o$\}$. In our adaptation, the discrete grid is replaced by a continuous 2D space. The agent moves in one of four cardinal directions, matching the original setting. To encourage generalization, we add zero-mean Gaussian noise (std. 0.05) to each step. The step size is set to 0.8, which approximates the original tile-based movement.

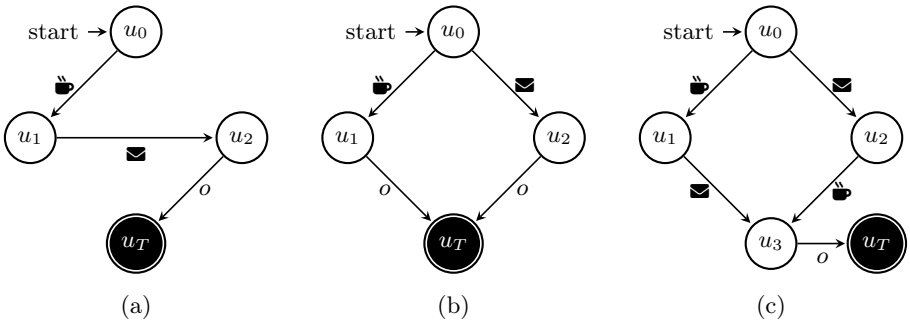


Fig. 1: FSA tasks for the Office environment: Sequential (a), Disjunctive (b), and Composite (c)

Tasks and Layouts. To test generalization and planning robustness, we extend the original Office environment by introducing new layouts with varied goal and obstacle configurations (see App. A, Fig. 3).

Original Office. This layout includes six goal regions—two per propositional variable in $\mathcal{P} = \{$☕, ✉, o$\}$. Each region occupies a 1×1 tile, and its associated symbol becomes true when the agent enters it. We place one Radial Basis Function (RBF) feature at the center of each tile with a fixed bandwidth $\sigma = 1$. This setup validates that our approach generalizes from the original discrete domain to the continuous setting. We evaluate on three canonical tasks adapted from [8]:

a sequential task ("Go to coffee, then to mail, then to an office"), a disjunctive task ("Go to coffee or mail, then to an office"), and a composite task ("Go to both coffee and mail in any order, then to an office")—see Fig. 1.

To increase complexity, we introduce a second set of tasks—one of each type—with longer logical sequences (see App. B, Fig. 4). These extended tasks highlight the limits of conventional Q-learning and the growing benefits of compositional methods like SF-OLS as task complexity and diversity increase.

Office Areas. Unlike the Original Office layout, where each propositional goal corresponds to a single tile, the Office Areas layout defines each goal as a continuous region spanning multiple 1×1 tiles, introducing spatial ambiguity and requiring more nuanced planning. The propositional set is $\mathcal{P} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$.

The B region is split into two disjoint sub-areas on opposite sides of a central obstacle wall. The left sub-area is closer to A, while the right is nearer to C. Notably, the right B area is large enough that the agent's exact entry point affects the cost of reaching C. For example, in the task "go to A, then B, then C," the agent must not only select the appropriate B sub-area but also reason about which entry point minimizes the cost of reaching C thereafter.

To support such spatial reasoning, we assign two RBF features to the right B area (one near C, one farther) and one RBF to each of the remaining smaller goal areas. All RBFs use a fixed bandwidth $\sigma = 3$. We evaluate on the same three FSA tasks as before, now adapted to $\mathcal{P} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$ (see App. B, Fig. 5).

Teleport. We introduce a stochastic variant of the Office environment with teleportation tiles—special tiles that instantly transport the agent to one of two predefined destinations at random. This setup evaluates the agent's planning under uncertainty. The layout includes two goal areas, A and B, placed on opposite sides of a long vertical wall. The propositional set is $\mathcal{P} = \{A, B\}$. Each goal region is represented by a 1×1 tile with a single RBF feature at its center ($\sigma = 1$). A teleporter at the bottom of the map randomly sends the agent—with equal probability—to a tile just above either goal, creating a high-risk, high-reward shortcut, offering the potential for significant time savings at the cost of destination uncertainty.

This mechanism poses a planning trade-off: for example, in a task requiring the agent to reach goal A specifically, the agent should avoid the teleporter to prevent being sent near B and needing a long detour. In contrast, for a disjunctive task like "Go to A or B," the teleporter becomes optimal, as either destination satisfies the goal and yields a shorter expected path. We evaluate two tasks in this layout: a disjunctive and a composite one (see App. B, Fig. 6). This tests the agent's capacity for goal-conditioned reasoning in stochastic settings.

4.2 Evaluation

We evaluate our method and baselines on the FSA tasks defined for each environment variant. Periodically during training, we measure the number of steps required to complete each task, assigning a reward of -1 per timestep—so lower

cumulative rewards indicate more efficient completion. We refer to this metric as *Mean Negative Step Reward*, which is used only for evaluation to quantify task efficiency, and differs from the reward used during training. Results are averaged over multiple independent runs (five by default) with different seeds.

Baselines. We compare against two baselines: a Flat Deep Q-Network (Flat DQN) and the Logical Options Framework (LOF). All models take the agent's normalized (x, y) coordinates as input (scaled to $[0, 1]$ based on observation bounds).

Flat DQN. The Flat DQN baseline is trained directly on the product MDP $\mathcal{M}' = \mathcal{F} \times \mathcal{M}^\phi$, where $\mathcal{F}$ is the FSA task and $\mathcal{M}^\phi$ the low-level MDP. In addition to spatial input, the agent receives a one-hot encoding of the current FSA state (length $|\mathcal{U}|$), preserving the Markov property.

Since Flat DQN learns a single policy for a specific FSA task and does not generalize across tasks, we train a separate agent for each task. Given a fixed total budget of training steps per method, this budget is distributed evenly across the different agents. As a result, increasing the number of FSA tasks reduces the number of training steps allocated to each Flat DQN agent.

Logical Options Framework. LOF is a compositional baseline similar to our method. It learns a library of options, each targeting a specific sub-goal, and constructs a meta-policy by treating these options as macro-actions. The resulting option-level MDP is solved via value iteration, yielding a policy that maps states to options without requiring further environment interaction at execution time. Unlike SF-OLS, LOF learns only one option per sub-goal and does not support options that reach combinations of sub-goals. This limits its flexibility and can lead to suboptimal performance in stochastic environments.[2]

Model Architectures and Training Details All methods use the same DQN backbone: a feedforward network with two hidden layers of 256 units and ReLU activations. We apply standard components, including a uniform replay buffer and target networks updated every 1000 steps. Exploration follows an ϵ-greedy strategy with ϵ linearly decaying from 1.0 to 0.1 over the first 50% of training. For full hyperparameter settings and schedules, see Appendix E or refer to our codebase [3].

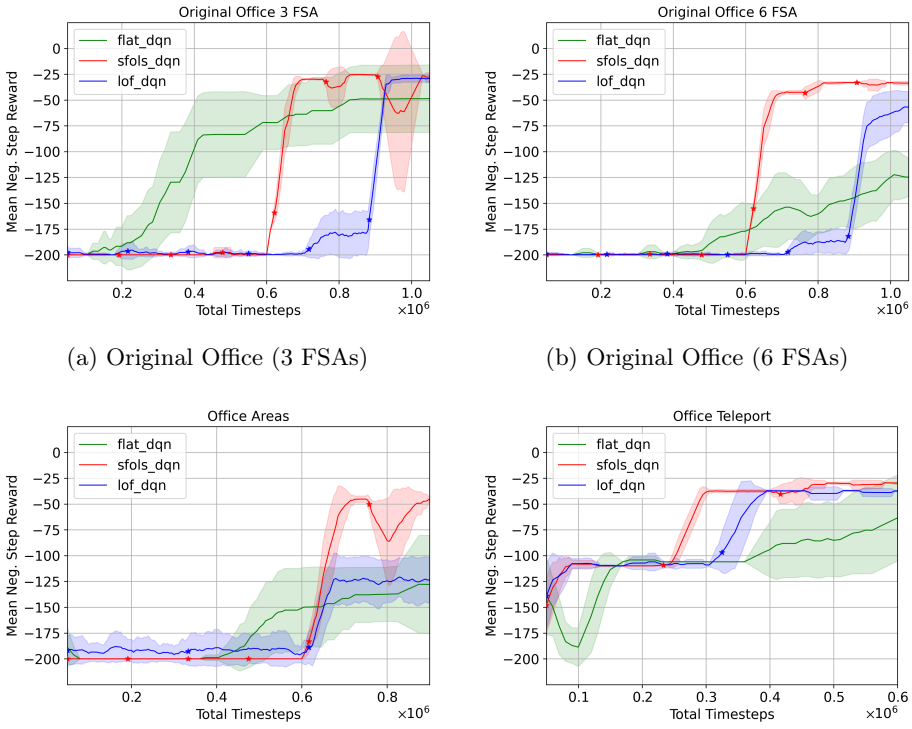
5 Results

Figure 2 shows the average performance of the three methods on sets of FSA tasks, evaluated periodically during learning. For SF-OLS and LOF, this corre-

[2] See App. E for implementation details and how the number of options varies per environment.

[3] https://github.com/timtimtim3/sf-fsa-vi-regression

sponds to learning the low-level policy set Π_{CCS} and the option base, respectively. The Flat DQN curve reflects a setup where separate networks are trained per FSA task and updated in an interleaved manner—i.e., as if training all networks concurrently by cycling through them and incrementing a shared global step counter.



(a) Original Office (3 FSAs)

(b) Original Office (6 FSAs)

(c) Office Areas (3 FSAs)

(d) Teleport (2 FSAs)

Fig. 2: Evaluation curves across Office layouts. Each plot shows the average number of steps to complete a task, averaged over all tasks per layout and 5 seeds. Shaded areas denote one standard deviation. Curves are smoothed (moving average, window size 10). Episodes end after 200 steps. Red and blue stars indicate when a new policy is added for SF-OLS and LOF, respectively.

5.1 Original Office

In the Original Office environment, both compositional methods converge to optimal behavior on the simpler set of three FSA tasks, while Flat DQN achieves near-optimal performance (Figure 2, a). This confirms that our continuous extension of SF-OLS performs on par with established baselines.

However, when the task set expands to six FSA tasks—three of which are more complex—a clear performance gap emerges (Figure 2, b): SF-OLS and LOF maintain high performance, while Flat DQN degrades significantly.

This drop stems from two main factors. First, Flat DQN trains a separate policy per task, so more tasks split the training budget across more networks. Increasing the training time for Flat DQN can partially mitigate this issue, allowing it to recover near-optimal performance (see Fig. 7 in App. D). Nevertheless, its performance still lags behind SF-OLS and, in contrast to SF-OLS and LOF, Flat DQN lacks any generalization capabilities to new FSA tasks.

Second, the added tasks are longer and more complex, with sparser rewards. These tasks require a longer sequence of correct actions before any reward is observed, making them harder to learn from scratch via monolithic RL. In contrast, for compositional methods like SF-OLS and LOF, increased task complexity does not directly affect the difficulty of the learning phase. This is because sub-policy learning is decoupled from the global task specification—each sub-policy is learned independently of the final FSA structure, and thus the overall training complexity remains constant.

5.2 Office Areas

In the Office Areas environment, SF-OLS consistently outperforms both LOF and Flat DQN (Figure 2, c). Flat DQN fails to fully converge within the training budget, while LOF's is limited by learning only one option per propositional goal.

Specifically, for the propositional variable B in $\mathcal{P} = \{A, B, C\}$, LOF learns a single option that does not differentiate between the left- and right-side sub-regions of goal area B. As a result, its policy navigates uniformly to the nearest reachable part of either region—regardless of the downstream impact on future goals (see Q-values in App. D, Fig. 9). This behavior leads to suboptimal performance in tasks such as "Go to A, then to B, then to C." For such tasks, LOF may select the left-side B region after reaching A, inadvertently increasing the distance to C, even though the right-side B region lies adjacent to C and would offer a more efficient trajectory.

In contrast, SF-OLS overcomes this limitation by learning distinct sub-policies for different spatial regions within the same goal area (see Q-values of sub-policies in App. D, Fig. 10). This allows it to selectively target the B sub-region closest to C, enabling more globally efficient plans (see value iterated Q-values in App. D, Fig. 11). This illustrates a key advantage of SF-OLS over LOF—namely, its capacity to represent and utilize spatially refined behaviors within goal regions for improved long-horizon planning.

5.3 Teleport

In the stochastic Teleport environment, SF-OLS achieves optimal performance, while LOF is slightly suboptimal (Figure 2, d). The gap stems from how sub-policies are learned. LOF constructs only two options—one for each goal (A and B)—both of which follow deterministic paths that avoid the teleporter. While

this satisfies both the disjunctive and composite tasks, it results in suboptimal trajectories; in the disjunctive task, the teleporter offers a faster expected route, since reaching either goal suffices. For the composite task, teleporting to one goal and walking to the other is optimal. LOF's fixed sub-policies miss this opportunity.

In contrast, SF-OLS learns an additional policy targeting both sub-goals. This policy takes the teleporter and proceeds to the nearest goal, allowing SF-OLS to exploit stochastic shortcuts, resulting in lower step counts. Though the performance difference is modest here, it would likely grow in larger environments—where the shortcut yields greater time savings—or under increased stochasticity—e.g., with multiple teleporters or more complex tasks. This highlights SF-OLS's advantage in handling stochastic dynamics.

6 Discussion and Conclusion

This work addresses the challenge of solving temporally extended, non-Markovian tasks in continuous state spaces by extending the SF-OLS framework to this setting. SF-OLS builds a reusable set of low-level policies using successor features (SFs), enabling zero-shot generalization to new finite state automata (FSA) tasks. Our main contribution is adapting SF-OLS to continuous domains by defining features that capture goal conditions over continuous spaces, and developing a regression-based value iteration algorithm to compute optimal task vectors for arbitrary FSA tasks.

We demonstrate that our continuous variant of SF-OLS outperforms both a Flat DQN baseline and a compositional method, LOF, particularly in environments where goals span spatial regions rather than singular points or tiles. The key advantage of SF-OLS over LOF stems from its capacity to learn multiple distinct sub-policies that target different parts of a goal area. This enables globally efficient planning, as SF-OLS can select sub-policies that consider both the immediate goal and downstream transitions. Furthermore, we show that SF-OLS maintains optimal behavior under stochastic dynamics, whereas LOF leads to suboptimal decisions in such settings.

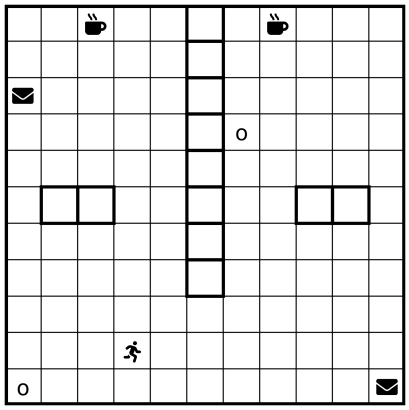
Despite its strong empirical performance, a limitation of our approach could be its reliance on hand-designed, linear feature representations, which may require domain knowledge and tuning to work effectively across different settings. A second limitation concerns the handling of FSA tasks with conditional effects. The policies in our basis are trained to reach each goal region directly, irrespective of such effects, which may lead to suboptimal behavior for tasks where entering certain regions causes adverse consequences—such as triggering FSA transitions that require the agent to visit additional regions. Notably, this limitation is not unique to SF-OLS; other compositional methods like LOF exhibit similar shortcomings. For a more extensive discussion of this limitation, see Appendix F.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

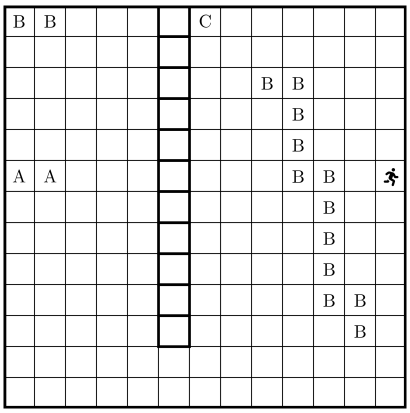
References

1. Alegre, L.N., Bazzan, A., Da Silva, B.C.: Optimistic linear support and successor features as a basis for optimal policy transfer. In: International conference on machine learning. pp. 394–413. PMLR (2022)
2. Araki, B., Li, X., Vodrahalli, K., DeCastro, J., Fry, M., Rus, D.: The logical options framework. In: International Conference on Machine Learning. pp. 307–317. PMLR (2021)
3. Barreto, A., Dabney, W., Munos, R., Hunt, J.J., Schaul, T., van Hasselt, H.P., Silver, D.: Successor features for transfer in reinforcement learning. Advances in neural information processing systems **30** (2017)
4. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of artificial intelligence research **13**, 227–303 (2000)
5. Icarte, R.T., Klassen, T., Valenzano, R., McIlraith, S.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. vol. 80, pp. 2107–2116. PMLR (10–15 Jul 2018)
6. Icarte, R.T., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. Journal of Artificial Intelligence Research **73**, 173–208 (2022)
7. Kuo, Y.L., Katz, B., Barbu, A.: Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 5604–5610. IEEE (2020)
8. Kuric, D., Infante, G., Gómez, V., Jonsson, A., van Hoof, H.: Planning with a learned policy basis to optimally solve complex tasks. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 34, pp. 333–341 (2024)
9. Pateria, S., Subagdja, B., Tan, A.h., Quek, C.: Hierarchical reinforcement learning: A comprehensive survey. ACM Computing Surveys (CSUR) **54**(5), 1–35 (2021)
10. Schaul, T., Horgan, D., Gregor, K., Silver, D.: Universal value function approximators. In: International conference on machine learning. pp. 1312–1320. PMLR (2015)
11. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
12. Sutton, R.S., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2), 181–211 (1999)
13. Tasse, G.N., James, S., Rosman, B.: A boolean task algebra for reinforcement learning. Advances in Neural Information Processing Systems **33**, 9497–9507 (2020)
14. Tasse, G.N., Jarvis, D., James, S., Rosman, B.: Skill machines: Temporal logic skill composition in reinforcement learning. arXiv preprint arXiv:2205.12532 (2022)
15. Vaezipoor, P., Li, A.C., Icarte, R.A.T., Mcilraith, S.A.: LTL2Action: Generalizing LTL instructions for multi-task RL. In: International Conference on Machine Learning. pp. 10497–10508. PMLR (2021)

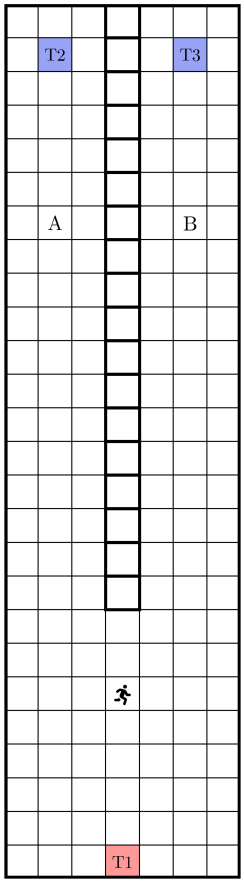
A Environments



(a) Original Office layout



(b) Office Areas layout



(c) Teleport layout

Fig. 3: Environment layouts used in our experiments. In the teleport layout, entering the red region (T1) teleports the agent with 50% probability to either of the two blue regions (T2 or T3). The 🚶 symbol marks the agent's evaluation start location. Thick-outlined cells denote impassable obstacles.

B FSA Tasks

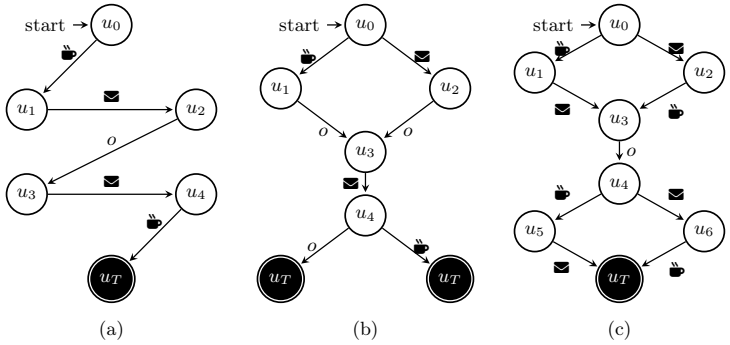


Fig. 4: Complex set of FSA tasks for the Office environment: Sequential (a), Disjunctive (b), and Conjunctive (c)

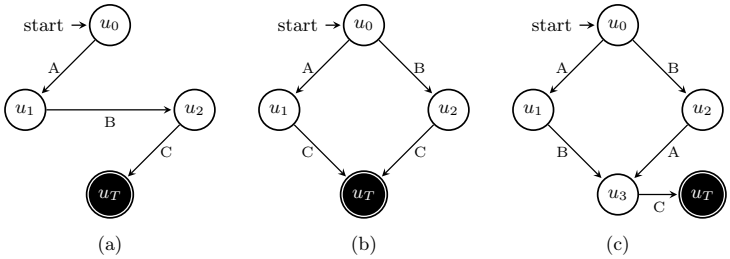


Fig. 5: FSA tasks for the Office Areas environment: Sequential (a), Disjunctive (b), and Composite (c)

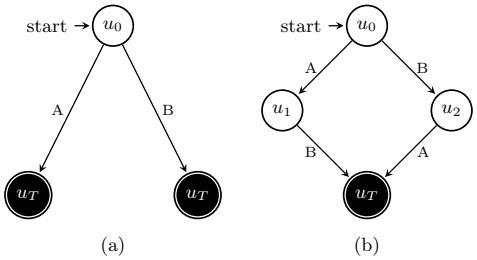


Fig. 6: FSA tasks for the Teleport Office environment: Disjunctive (a), and Composite (b)

C Algorithms

Algorithm 1 SF-FSA-VI Regression

Input: Low-level MDP M_ϕ, task specification $\mathcal{F}$

Obtain Π_{CCS} on $\mathcal{M}_\phi$
Initialize $\mathbf{w} = \mathbf{0} \in \mathbb{R}^{|\mathcal{U}|\cdot d}$, and ϵ to some small value like $1e-5$.
Sample a set of exit states $\mathcal{E}$ *// e.g., by random sampling or grid layout over sub-goal areas*
while not done **do**
    Initialize $\Phi = \mathbf{0} \in \mathbb{R}^{(|\mathcal{E}|\cdot|\mathcal{U}|)\times(|\mathcal{U}|\cdot d)}$, and $\hat{\mathbf{y}} = \mathbf{0} \in \mathbb{R}^{|\mathcal{E}|\cdot|\mathcal{U}|}$
    for each $u \in \mathcal{U}$ **do**
        for each $\varepsilon \in \mathcal{E}$ **do**
            Compute $\hat{y}(\varepsilon, u)$ using Equation (18) and $\boldsymbol{\phi}(\varepsilon, u)$ using the Definition in (13)
            Store $\hat{y}(\varepsilon, u)$ and $\boldsymbol{\phi}(\varepsilon, u)$ in $\hat{\mathbf{y}}$ and Φ, respectively
        end for
    end for
    Solve for $\mathbf{w}^*$ using Equation (19)
end while
return $\{\mathbf{w}^*(u)\ \forall u \in \mathcal{U}\}$

D Additional Figures

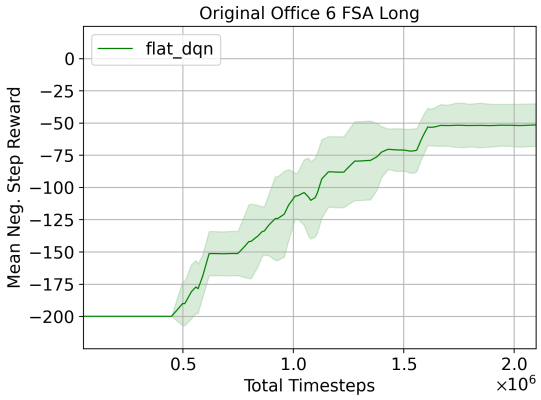


Fig. 7: Evaluation curve for Flat DQN on the 6-task variant of the Original Office layout. The plot shows the average number of steps (reported as negative rewards) needed to complete a task, averaged over all FSA tasks and across 3 seeds. Shaded regions denote one standard deviation. The curve is smoothed (moving average, window size 10), and episodes terminate after 200 steps.

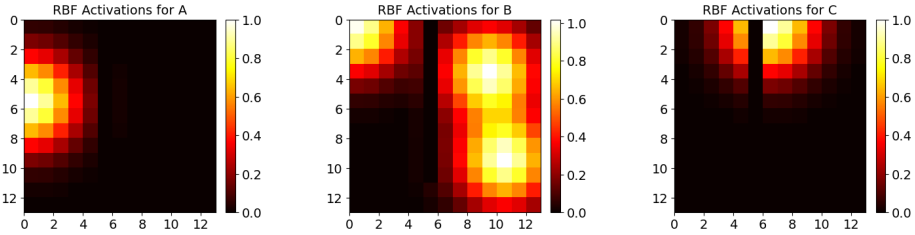
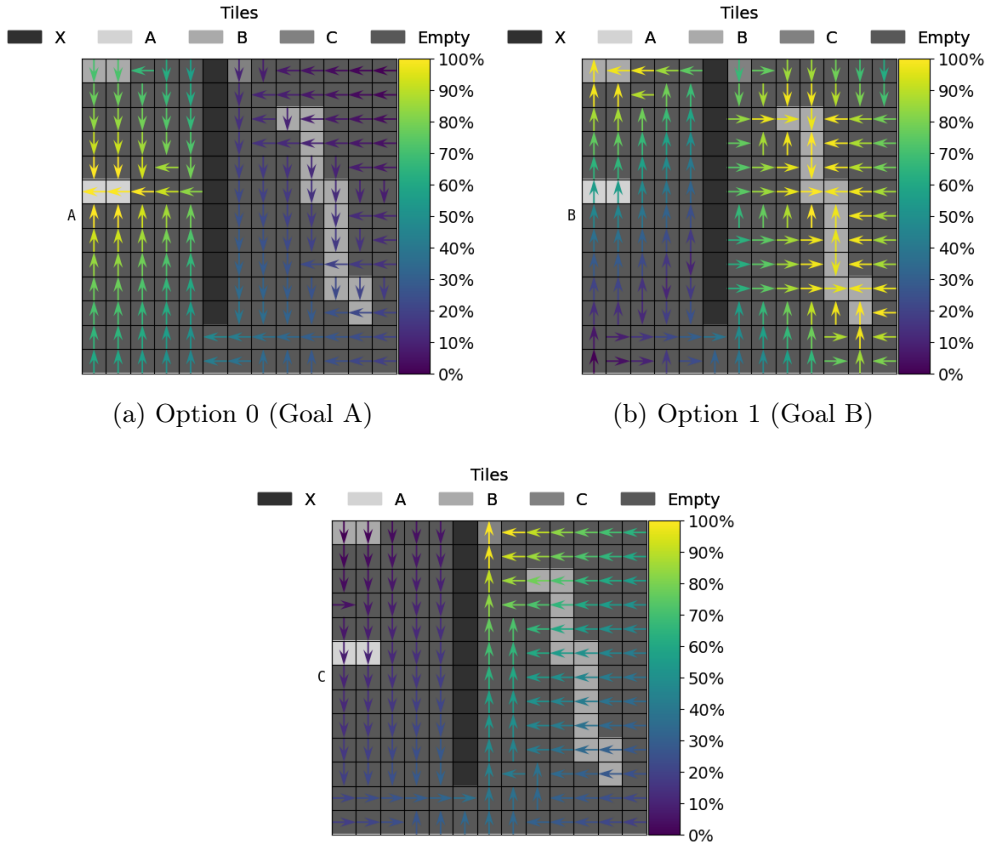


Fig. 8: RBF feature activations for propositional variables in the Office Areas layout. Sub-goal A on the left, B in the middle, C on the right. Saturation indicates activation strength.

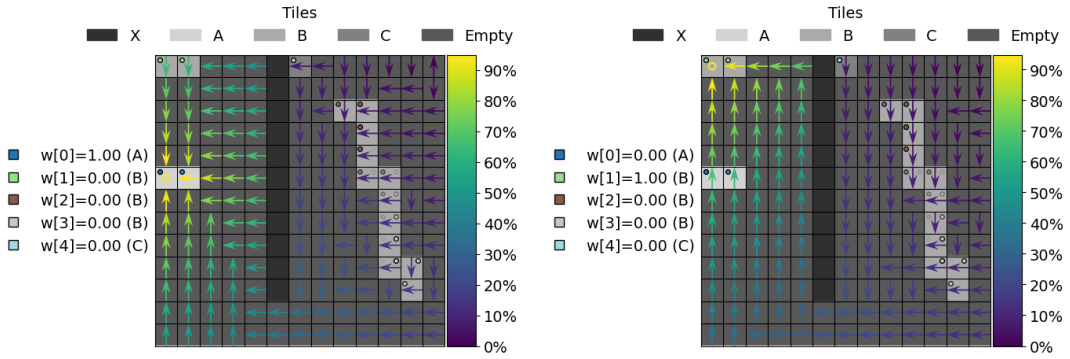


(a) Option 0 (Goal A)

(b) Option 1 (Goal B)

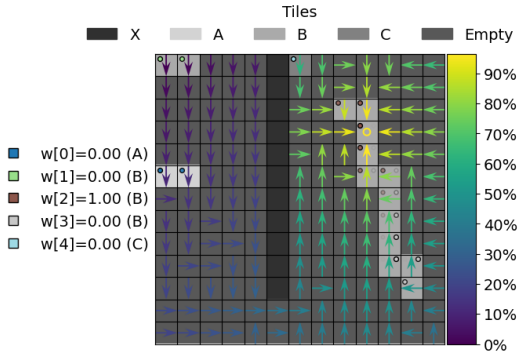
(c) Option 2 (Goal C)

Fig. 9: LOF option Q-values in the Office Areas layout. Arrows indicate greedy actions with highest Q-values; color denotes Q-value magnitude. Option 1 (Goal B) exhibits uniform behavior across all reachable parts of B, potentially leading to suboptimal outcomes.

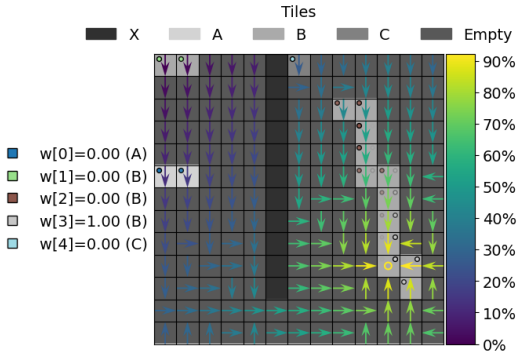


(a) Policy 0

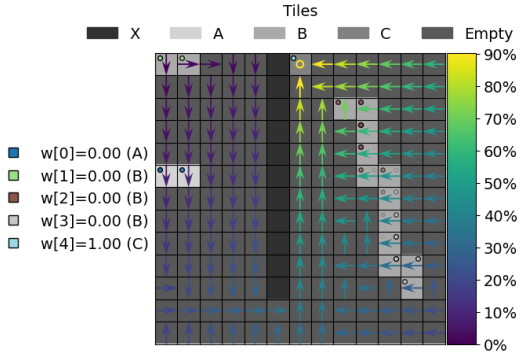
(b) Policy 1



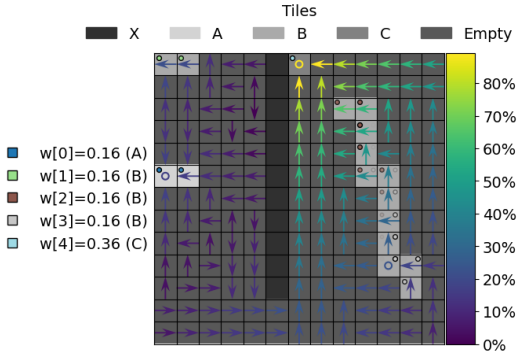
(c) Policy 2



(d) Policy 3

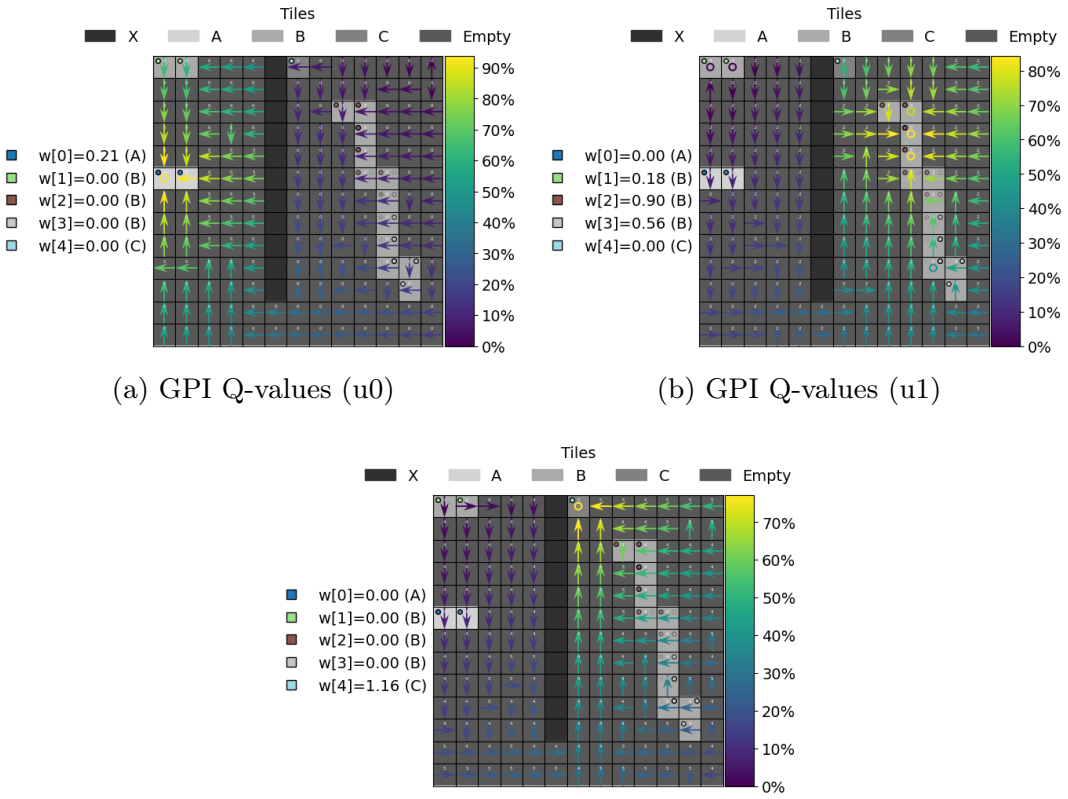


(e) Policy 4



(f) Policy 5

Fig. 10: SF-OLS sub-policy Q-values in the Office Areas layout. Weights on the left of each plot indicate which combination of features is being steered towards. Feature activations on sub-goals are indicated by colored dots. Arrows indicate greedy actions with highest Q-values; open circles denote terminal actions; arrow color encodes Q-value magnitude. Demonstrates SF-OLS's capacity to learn distinct policies for different parts of goal B.



(a) GPI Q-values (u0)

(b) GPI Q-values (u1)

(c) GPI Q-values (u2)

Fig. 11: SF-OLS Value Iterated GPI Q-values for the sequential task (A → B → C) in the Office Areas layout. Weights on the left of each plot indicate which combination of features is being steered towards. Feature activations on sub-goals are indicated by colored dots. Arrows indicate greedy actions with highest Q-values; open circles denote terminal actions; arrow color encodes Q-value magnitude; numbered tiles indicate selected sub-policy index. Highlights spatial reasoning where SF-OLS selects region of B closest to C.

E Hyper-parameters and Experimental Setup

E.1 Hyper-parameters

Table 1: Training hyperparameters used across methods and environments.

Parameter	**Value**
Network and DQN Parameters	
Architecture	2-layer MLP with 256 units per layer, ReLU activations
Optimizer	Adam
Learning rate	3×10^{-4}
Replay buffer size	1×10^{6}
Batch size	256
Target update frequency	Every 1000 steps
Exploration strategy	ϵ-greedy
Initial ϵ	1.0
Final ϵ	0.1
ϵ decay schedule	Linear over first 50% of training steps
Discount Factors	
FlatQ (FlatDQN)	$\gamma = 0.99$
Compositional methods (SF-OLS, LOF)	$\gamma = 0.95$
Training Budgets	
Original Office	1,050,000 steps
Office Areas	900,000 steps
Office Teleport	600,000 steps
FlatQ – Original Office 6-FSA-Long	2,100,000 steps
Compositional-Specific Parameters	
OLS priority threshold ε	0.2
Seeds	
Default	1001, 1002, 1003, 1004, 1005
FlatQ – Original Office 6-FSA-Long	Only 1001, 1002, 1003 used

E.2 LOF Experimental Setup

In the Original Office environment, we designate each individual exit state tile as a distinct sub-goal, leading LOF to learn a separate option for each. This results in an option library containing six options — one for each unique tile associated with the three propositional variables (i.e., two tiles per variable). For example, the two coffee tiles are treated as independent sub-goals, each with its own corresponding option. In contrast, SF-OLS treats multiple exit states associated with the same propositional variable as interchangeable: a feature defined for a given proposition (e.g., coffee) can be active on any of its associated exit states. As such, this formulation provides LOF slightly different information than SF-OLS by distinguishing each propositional variable into multiple sub-goals — one

for each exit state tile (e.g., treating ☕[1] and ☕[2] as separate sub-goals derived from ☕).

In the Office Areas environment, we instead define a single sub-goal per propositional variable, resulting in a total of three options. This design choice is motivated by the fact that the propositional variables correspond to a large number of exit state tiles. Defining a separate option for each of these tiles would lead to an impractically large option set. While it would be theoretically possible to define each exit tile as a distinct sub-goal — potentially improving performance — this approach does not scale to larger environments due to the large increase in the number of options. One could mitigate this by coarsening the spatial resolution of the grid (thereby reducing the number of distinct sub-goals) or by adopting a more flexible representation similar to that of SF-OLS, wherein features are defined over spatial regions rather than individual tiles. However, to preserve consistency with the original LOF formulation and maintain its role as a baseline, we do not pursue such modifications here, focusing instead on improving the SF-OLS framework.

F Extended Discussion

A limitation of our approach lies in its handling of FSA tasks that involve conditional effects or side effects associated with specific propositional variables. For instance, consider a task that requires reaching a goal region "G," but where passing through an intermediate region "M" (e.g., mud) causes the agent to become dirty, triggering an FSA transition that requires visiting a cleaning region "C" before completing the task. In such cases, entering the mud region leads to an overall longer trajectory compared to avoiding it altogether. However, the policies in our basis are trained to reach each goal region directly, irrespective of such side effects, since they only terminate if the terminal action is taken. Consequently, if "M" lies along the shortest path to "G," the policy targeting "G" may traverse "M," resulting in suboptimal behavior for the task at hand. One possible workaround is to exclude "M" from the reachable set during policy training—for example, by automatically terminating any episode upon entering "M"—but this introduces a trade-off: for other FSA tasks that do not distinguish between being clean or dirty, the learned policy would unnecessarily avoid "M," resulting in a longer path than necessary. Ideally, the agent should be able to solve both types of FSA tasks using a shared set of base policies, adapting its behavior through high-level planning rather than policy-level exclusions. Notably, this challenge is not unique to SF-OLS—other compositional methods like LOF are similarly limited in this regard.

One potential avenue to address this issue is to train each policy in the base set to avoid entering any goal region other than its intended target—for example, by automatically terminating episodes upon entering a goal. This would encourage policies to take detours around other goal regions unless the agent starts within one. In the context of a task like "go to G" where cleanliness is irrelevant, this approach may lead to suboptimal behavior: the agent might avoid region "M" (mud) even if passing through it would yield a shorter trajectory. To restore optimality, one could enrich the FSA by explicitly adding transitions corresponding to such intermediate goals. For instance, by adding a path to the FSA that allows visiting "M" before "G," the planning procedure could choose to first execute the policy that reaches "M" and then the one that reaches "G," resulting in the shorter route being selected. However, this approach comes with notable drawbacks. First, it requires manually augmenting the FSA with transitions that account for all relevant combinations of sub-goals. Second, to ensure globally optimal behavior, the FSA would need to include every potentially beneficial intermediate goal sequence—e.g., "X $\rightarrow$ M $\rightarrow$ G"—even if "X" lies en route to "M," but is not necessary for the task. Without such explicit transitions, the planner might avoid useful regions like "X" because the corresponding policy would steer around them. Thus, while this solution offers a path to greater flexibility, it shifts the complexity to the task design phase by requiring a carefully constructed, exhaustive FSA. As such, it remains unclear how to effectively address these challenges in continuous state spaces, and we believe this is an important direction for future work