RELATIVE BEHAVIORAL ATTRIBUTES: FILLING THE GAP BETWEEN SYMBOLIC GOAL SPECIFICATION AND REWARD LEARNING FROM HUMAN PREFERENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating complex behaviors from goals specified by non-expert users is a crucial aspect of intelligent agents. Interactive reward learning from trajectory comparisons is one way to allow non-expert users to convey complex objectives by expressing preferences over short clips of agent behaviors. Even though this method can encode complex tacit knowledge present in the underlying tasks, it implicitly assumes that the human is unable to provide rich-form feedback other than binary preference labels, leading to extremely high feedback complexity and poor user experience. While providing a detailed symbolic specification of the objectives might be tempting, it is not always feasible even for an expert user. However, in most cases, humans are aware of how the agent should change its behavior along meaningful axes to fulfill the underlying purpose, even if they are not able to fully specify task objectives symbolically. Using this as motivation, we introduce the notion of *Relative Behavioral Attributes*, which acts as a middle ground, between exact goal specification and reward learning purely from preference labels, by enabling the users to tweak the agent's behavior through nameable concepts (e.g., decreasing the steering *sharpness* of an autonomous driving agent, or increasing the *softness* of the movement of a two-legged "sneaky" agent). We propose two different parametric methods that can potentially encode any kind of behavioral attributes from ordered behavior clips. We demonstrate the effectiveness of our methods on 4 tasks with 9 different behavioral attributes and show that once the attributes are learned, end users can effortlessly produce desirable agent behaviors, by providing feedback just around 10 times. The feedback complexity of our approach is over 10 times less than the learning-from-human-preferences baseline and this demonstrates that our approach is readily applicable in real-world applications.

1 INTRODUCTION

A central problem in building versatile autonomous agents is how to specify and customize agent behaviors. Two representative ways to specify tasks include manual specification of reward function, and reward learning from trajectory comparisons. In the former, the user needs to provide an exact description of the objective as a suitable reward function to be used by the agent. This is often only feasible when the specification can be done at a high level, e.g. in symbolic terms (Russell & Norvig, 2003) or providing a symbolic reward machine or domain model (Icarte et al., 2022; Yang et al., 2018; Illanes et al., 2020), or giving a natural language instruction (Mahmoudieh et al., 2022). Instead of requiring users to give precise task descriptions, reward learning from trajectory comparisons learns a reward function from human preference labels between two queried behavior clips (Wilson et al., 2012; Christiano et al., 2017) or from numerical ratings on trajectory segments (Knox & Stone, 2009; MacGlashan et al., 2017; Warnell et al., 2018) or from rankings over a set of behaviors (Brown et al., 2019).

One appealing feature of reward learning from trajectory comparisons is the breadth of information it can encode. Since the reward function is typically parameterized (e.g., by a neural network), it can learn to encode knowledge that cannot be easily expressed in any structural form (i.e., as in symbolic goal specification). Examples of such tacit knowledge can be how a humanoid robot can

walk *naturally* like humans or how a virtual character can move *sneakingly*. Despite its remarkable representational capacity, the applicability of reward learning from human preferences is still restricted by its need for an excessive amount of human involvement and labeling. Though numerical ratings or binary preference labels are flexible in terms of expressing complex objectives, they are not very informative about the user's intended goals. As a result, the system must gather a significant amount of human feedback to accurately capture the user's latent preference. Moreover, since the internal representations learned within neural networks are typically inscrutable and user-specific, it's unclear how a trained reward model can be reused to serve multiple users and fulfill a variety of goals.

In short, symbolic goal specification is more straightforward and intuitive to use but it offers limited expressiveness, making it more suitable for explicit-knowledge tasks. Reward learning from trajectory comparisons, in contrast, offers better expressiveness, but it is more costly and less intuitive to use since it treats any task at hand as a tacit-knowledge one and thus limits human feedback to plain numerical labels. However, in many real-world scenarios, user objectives are neither completely explicit nor purely tacit. In other words, although human users may not be able to construct a reward function symbolically, they still have some idea of how the intelligent agent can change its behavior along certain meaningful axes of variation to better serve the users. Let us note a few ways in which this assumption is more permissive: (a) the users of a legged household robot may want it to walk *more softly* and take *smaller steps* at night when people are sleeping, though they are unable to define the low-level control policy in a codified way; (b) to get a more pleasant ride experience, passengers of a self-driving car may want to adjust the *steering sharpness* during a lane change and the *smoothness* of braking and acceleration, even though they cannot fully specify an expected driving style.

Motivated by the observations above, we introduce the notion of *Relative Behavioral Attributes* to capture the relative strength of the attribute's presence in the agent's current behavior (i.e., a trajectory sampled according to the agent's policy), aiming to extract the explicit parts of any task that potentially involves tacit knowledge. Once the behavioral attributes are learned, each time the agent presents its current behavior, the user will be able to provide feedback on whether the current behavior is satisfactory. In addition, the user will be able to express the intent to increase or decrease the strength of certain attributes. The benefits of explicitly modeling relative behavioral attributes are obvious: (a) it offers a semantically richer and more natural mode of communication, such that the hypothesis space of user intents or preferences can be greatly reduced each time the system receives attribute-level feedback from the user, thereby improving the overall feedback complexity significantly; (b) since humans communicate at the level of symbols and concepts, the learned behavioral attributes can be used as a symbolic *lingua franca* (Kambhampati et al., 2022) between humans and the agent, and more importantly, such shared vocabulary can be *reused* to serve any future users and support a diverse set of objectives. Relative behavioral attributes essentially act as a middle ground between symbolic goal specification and reward learning from preference labels, and it draws on the strengths of both methods. Note that our approach is not intended to replace any of the two current methods. This is because the choice of method for goal specification depends on the nature of the underlying task. Instead, we are advocating *communications at concept level* (as long as the user requirements are describable), as a means to achieve the best user experience and system efficiency.

The concept of relative attributes has been introduced earlier in other areas like computer vision (Parikh & Grauman, 2011) and recommender systems (Goenka et al., 2022). Though they share some similarities in motivations and definition to our relative behavioral attribute, the key difference is that relative attribute only captures properties present in a single static image, while relative behavioral attribute aims to capture properties in sequential decision-making problems, wherein an attribute may be associated with either static features at a certain timestep (e.g., step size of a walking agent) or temporally-extended features spanning multiple steps (e.g., the softness of movement). Additionally, we need to consider how to connect the captured attributes to a valid reward function. In this work, we propose two generic data-driven methods to encode relative behavioral attributes within a reward function from different perspectives. On 4 domains with 9 attributes in total, we show that our methods can learn to accurately capture the attributes from approximately 200[1] labelled behavior clips. And once the attributes are learned, any end user can effortlessly produce

[1]This is not a rigorous statistic because the training data for the two proposed methods are in different forms. The number here is just to indicate that our method can learn to encode the attributes from a reasonable number of labelled samples. Please see Sec. 5 for details.

diverse desirable agent behaviors, by providing feedback just around 10 times. This shows a significant advantage over the learning-from-human-preferences baseline (Christiano et al., 2017) that requires hundreds of binary labels from the user per task.

2 Related Works

In general, direct reward specification is the most commonly employed approach. It is used in almost all simulators with engineered reward functions and in industrial applications such as self-driving vehicle control systems. Yet, there are still some challenges associated with it, such as extensive requirement for sensory instrumentation, incomplete (sub)goal specification (Guan et al., 2022), reward exploitation (Lee et al., 2021) and the expressiveness issue mentioned in the previous section. These impediments motivate the idea of learning reward from states or trajectory comparisons (Warnell et al., 2018; Christiano et al., 2017; Lee et al., 2021), which leverages the exceptional representational capacity of neural networks but tends to suffer from high data complexity.

The introduction of relative behavioral attributes aligns with the broader intent of building symbolic interfaces as a middle layer for humans to communicate effectively with the agent (Bobu et al., 2021; Guan et al., 2021; Silver et al., 2022) or for the agent to explain to humans (Kim et al., 2018; Sreedharan et al., 2022; Kambhampati et al., 2022). Lee et al. (2020) utilize relative-attribute information in robot skill learning, but their GAN-based formulation is restricted to static visual attributes and is not applicable to temporally-extended concepts.

This paper adopts a similar setup to works that learn diverse skills or motion styles from large-scale offline behavior datasets or demonstrations (Lee & Popović, 2010; Wang et al., 2017; Zhou & Dragan, 2018; Peng et al., 2018b; Luo et al., 2020; Chebotar et al., 2021; Peng et al., 2021). These works emphasize on modeling a variety of reusable motor skills by learning a low-level controller conditioned on skill latent codes. Since the latent codes are inscrutable to humans, for each new task, the user must specify the desirable agent behavior by constructing an engineered *symbolic reward* and use it to train a separate high-level policy that controls the low-level controller. In this paper, we tackle the problem of easily constructing such behavioral rewards through nameable concepts. Our methods are complemented by existing diverse-skill learning methods, because skill priors (i.e., pre-trained low-level controllers) allow us to optimize the behavioral reward more efficiently.

3 Problem Setup

Personalizing agent behaviors at skill level. We assume the users are interested in customizing agent behaviors at the level of skills (e.g., making a lane change, taking a step, picking up an object). A skill corresponds to a solution that solves an episodic task in an indefinite-horizon discounted MDP $\mathcal{M} = \langle S, A, R, P, \gamma \rangle$, where S is the set of states, A is the set of actions, $R : S \times A \to \mathbb{R}$ is a provided reward function, $P : S \times A \times S \to [0, 1]$ is the transition function and γ is the discount factor. An episode terminates whenever the agent reaches an absorbing terminal state or exceeds a fixed number of time steps T. As a common practice, we assume the absorbing states are determined by the environment, though it is also possible to learn such terminal conditions (Sutton et al., 1999). Note that the terminal conditions typically contain little or no information about the user's objectives and the relevant attributes. An agent M is supposed to find a policy ($\pi : S \to A$) that maximizes the expected return $E[\sum_{t=0}^{T} \gamma^t r_t], r \in \mathbb{R}$. When performing a skill, the agent is initialized to some initial state $s_0 \in \mathcal{S}$ at the beginning of each episode; and then at each succeeding time step t, the agent interacts with the environment $\mathcal{E}$ by taking an action $a_t \in \mathcal{A}$ and receiving the next state $s_{t+1} \in \mathcal{S}$. In order to personalize agent behavior, rather than assuming that the reward function R is supplied by the environment, we require the reward function to be constructed or specified by the end user.

Learning to encode relative behavioral attributes from offline behavior datasets. Relative behavioral attributes are essentially semantically capturing different ways to carry out a task. Ideally, the training data for the attribute encoder should contain clips of diverse skill behaviors. Although we used synthetic data in this study, there are many publicly accessible behavior corpora, such as the Waymo Open Dataset for autonomous driving tasks (Ettinger et al., 2021) and large-scale motion clips data for character control (Peng et al., 2018a; 2022). Note that we don't expect the offline

dataset to exhaustively cover all possible skill behaviors, as the attribute encoder should generalize to unseen configurations.

Considering that action information is not always available, to be more general, we assume our attribute encoder is trained on state-only data. More specifically, the training dataset $\mathcal{D}$ for relative behavioral attributes consists of a collection of state-only trajectories $\{\tau_s\}$ (i.e., behavior clips), where $\tau_s = \{s_0, ..., s_l\}$ is a l-length trace (l may be varying) and s_t is the state at timestep t. For image-based datasets, states can be constructed, for example, by unsupervisedly training a VAE encoder (Kingma & Welling, 2013) and stacking the image embeddings of consecutive frames (Mnih et al., 2013). Last but not least, as a common setting, we assume the embodiment of the agent is not significantly different from that in $\mathcal{D}$.

Reward learning from trajectory comparisons. Reward learning from trajectory comparisons methods construct a reward function by learning to infer the user's latent preference from a set of ranked trajectories $\{(\tau^1, \tau^2)\}$, where $\tau^i = ((s_0^i, a_0^i), ..., (s_{l-1}^i, a_{l-1}^i))$ and l is the length of the trajectory. Typically, the user preference is modelled according to the Bradley-Terry model (Bradley & Terry, 1952):

$$P\left[\tau^1 \succ \tau^2\right] = \frac{\exp \sum_t r\left(s_t^1, [a_t^1]\right)}{\sum_{i\in\{1,2\}} \exp \sum_t r\left(s_t^i, [a_t^i]\right)}, \tag{1}$$

where $\tau^1 \succ \tau^2$ denotes the event that trajectory 1 is preferable over trajectory 2, r is a parametric learnable reward function, and $[a_t^i]$ means the preference model can be state-only (the action information is optional). A cross-entropy loss is typically used for optimization:

$$Loss_p(r) = -\sum_{\left(\tau^1,\tau^2,y\right)\in\mathcal{D}_p} y(1)\log P\left[\tau^1 \succ \tau^2\right] + y(2)\log P\left[\tau^2 \succ \tau^1\right], \tag{2}$$

where $y \in \left\{(1,0),(0,1),(0.5,0.5)\right\}$ are possible preference labels indicating $\tau^1 \succ \tau^2$, $\tau^2 \succ \tau^1$, or τ^1 and τ^2 are equally preferred, respectively. The set of preference labels $\mathcal{D}_p$ can be collected beforehand (Brown et al., 2019) or by active queries to the user in an online manner (Wilson et al., 2012; Christiano et al., 2017).

4 METHODOLOGY

4.1 PERSONALIZING AGENT BEHAVIOR VIA RELATIVE BEHAVIORAL ATTRIBUTES

The goal of modeling relative behavioral attributes is to construct an attribute parameterized reward function, such that the human user can use high-level concepts to control the agent behavior. Overall, this involves two phases, namely, modeling relative behavioral attributes and interacting with the end user.

Learning attribute parameterized reward function (no interaction with the end user). This phase is supposed to learn an attribute conditioned reward function that *internally* learns a family of rewards that correspond to behaviors with diverse attribute strengths. By varying the input attribute configurations, end users will be able to find the preferred "reward member" and produce desirable behaviors. In Section 4.2 and Section 4.3, we will present two methods that can learn such a reward function given a subset of labelled trajectories from an offline behavior dataset $\mathcal{D}$. We assume that the *agent builders* are the ones who provide the training labels (e.g., the engineers of autonomous vehicles, and the developer of virtual characters). In some extreme cases, if a novel concept has to be learned, the training labels may also come from the end user. Also note that, this step *can be skipped* if the relevant behavioral attributes have already been captured by a learned reward function.

Supporting end users in the loop. Once an attribute-conditioned reward function is learned, any incoming users can leverage it to personalize the agent behavior through multiple rounds of query. In each round of interaction, the agent presents the user a trajectory sampled according to the policy that optimizes the current reward function. Then the user provides a feedback on whether the current behavior is desirable; and if not satisfied, the user can express the intent to increase or decrease the strength of certain attributes. The concept-level feedback is a set of attribute-feedback pairs $\{(\alpha, h)\}$, where α denotes the attribute of interest, and h is a binary value indicating whether the user wants to increase or decrease α's strength. In this work, we consider two types of attribute representation, namely the index of α in a list of known attributes or a natural-language description

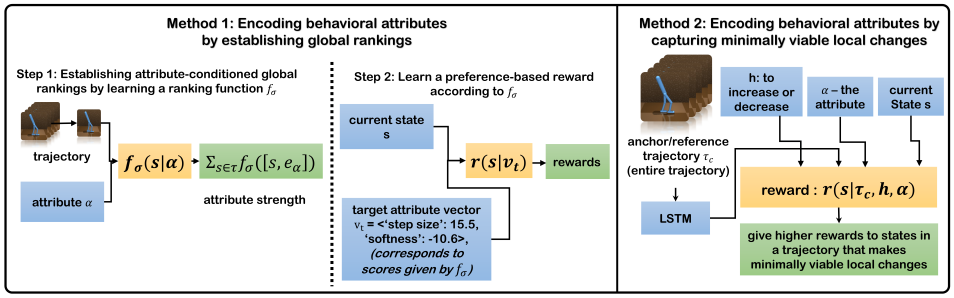

Figure 1: Overview of Method 1 and Method 2.

of α. Upon collecting feedbacks from the user, the agent will adjust the reward function and update the corresponding policy. This human-agent interaction process repeats until the user is satisfied with the latest agent behavior.

In the next two sections, we will explain: (a) the options of the computational frameworks for implementing configurable attribute parameterized reward functions; (b) how to configure the learned reward function after receiving a feedback from the user.

4.2 Method 1: Encoding Behavioral Attributes by Establishing Global Rankings

Methods in computer vision usually encode relative attributes by employing learning-to-rank techniques (Joachims, 2002; Parikh & Grauman, 2011). They construct a ranking function by learning to map each item (e.g., image) to a real-valued score that measures the relative strength of the attribute presence. Here, we refer to such ordering determined by the ranking function as *global ranking*, because it is essentially establishing a mapping from all items to a score in a universal one-dimensional real space, which represents all possible strengths of the encoded attribute. Similarly, in sequential decision-making problems, we can also encode relative behavioral attributes by establishing some global rankings.

Specifically, we suppose we are given (normally by the agent designers not the end users) a set of state-only trajectories $\{\tau\}$ and their orderings according to different attributes $\{(\tau^0 \succ \tau^1 \succ ... \succ \tau^N|\alpha)\}$, or a set of ranked trajectory pairs $\mathcal{D}_l = \{(\tau^i \succ \tau^j|\alpha)\}$, where $\alpha \in \mathcal{A}$ is one of the attributes in the domain, $(\tau^i \succ \tau^j|\alpha)$ represents the event that the attribute α has a strong presence in the trajectory τ^i than trajectory τ^j, and $N \leq |\mathcal{D}|$ is the length of the ranked trajectory sequence. Our goal is to learn a ranking function f that models the relative orderings among the labelled trajectories. One way to achieve this is to cast the ordering modeling problem to an *attribute-conditioned* preference modeling problem. In particular, we can employ a modified state-only version of Bradley-Terry model (Eq. 1), in which rather than assuming the ranking is governed by the latent user preferences, we assume the the ranking is determined by the given attribute α:

$$
\begin{aligned}
P_\sigma\left[\tau^1 \succ \tau^2 \middle| \alpha\right] &= \frac{\exp \sum_t f_\sigma\left(s_t^1|\alpha\right)}{\sum_{i\in\{1,2\}} \exp \sum_t f_\sigma\left(s_t^i|\alpha\right)} \\
&= \frac{\exp \sum_t f_\sigma\left([s_t^1, e_\alpha]\right)}{\sum_{i\in\{1,2\}} \exp \sum_t f_\sigma\left([s_t^i, e_\alpha]\right)},
\end{aligned}
\tag{3}
$$

where f_σ is the attribute conditioned ranking function with parameters σ, $[\cdot,\cdot]$ is the vector concatenation operation, and e_α is the embedding of attribute α. Recall that we consider two types of attribute representation, namely attribute index and natural language description. Accordingly, e_α can either be a one-hot vector or a sentence embedding generated by any pretrained natural language sentence encoder like Sentence-BERT (Reimers & Gurevych, 2019). Since different behaviors may result in trajectories of varying lengths, in Eq. 3 we don't require any two trajectories to have the same size.

Given the training dataset $\mathcal{D}_l$, an attribute conditioned ranking function f_σ can be trained via a cross-entropy loss similar to the one in Eq. 2. For numerical stability, in practice, we also clip the

values of $\sum_t f_\sigma\left([s_t^i, e_\alpha]\right)$ during training (we used $[-20, 20]$ for all the attributes in our experiment). With a trained f_σ, the strength of any attribute $\alpha \in \mathcal{A}$ in a trajectory τ is given as $\zeta(\tau, \alpha) = \sum_{s\in\tau} f_\sigma\left([s, e_\alpha]\right)$. And given a finite set of attributes $\mathcal{A}$, the agent behavior in any trajectory τ can be represented by an attribute vector $v(\tau) = \langle \zeta(\tau, \alpha_1), ..., \zeta(\tau, \alpha_k)\rangle$, where $k = |\mathcal{A}|$ is the number of attributes.

It remains to address how to produce desirable agent behaviors by leveraging f_σ. Here, we provide the most generic solution that constructs an attribute parameterized reward function such that a policy can be found by optimizing the reward with any major sequential decision problem solver, such as planning based methods and reinforcement learning based methods. Additional discussion on alternative ways can be found in Sec.A.1. The key idea is to construct a preference-based dense reward function $r_\theta([s, v_t])$ that takes the concatenation of the current state vector s and an attribute vector v_t that specifies the target attribute strengths as input. The reward function r_θ is supposed to give higher cumulative rewards to trajectories that have closer attribute strengths to the target v_t. Specifically, given v_t, for any two trajectories τ_i and τ_j from the offline behavior dataset $\mathcal{D}$, the preference label $l_p(\tau^i, \tau^j, v_t)$ is given as:

$$l_p(\tau^i, \tau^j, v_t) = \begin{cases} \tau^i \succ \tau^j & \text{if } \|v(\tau^i) - v_t\|_2 < \|v(\tau^j) - v_t\|_2 - \xi_r \\ \tau^i \prec \tau^j & \text{if } \|v(\tau^i) - v_t\|_2 > \|v(\tau^j) - v_t\|_2 + \xi_r \\ \text{no ordering} & \text{otherwise,} \end{cases} \tag{4}$$

where ξ_r is a small slack variable. To train r_θ, we can randomly sample a set of triplets from $\mathcal{D}$, namely $\{(\tau^0, \tau^1, \tau^2)\}$. By treating τ^0 as the target behavior, we can generate a set of training labels $\{l_p(\tau^1, \tau^2, v_t = v(\tau^0))\}$ for r_θ. In short, r_θ can be trained as a standard state-only preference-based reward function according to Eq. 1 and Eq. 2 but with preference labels given by the extracted attribute strengths (i.e., by f_σ). Note that, unlike the training of f_σ, the training of r_θ does not require any human-provided labels.

Once a reward function r_θ is learned, the end user can use it to specify agent behavior by simply tuning the attribute strength values in the input target attribute vector v_t. As an example, for a walking virtual character domain with attributes $\{\alpha_0$: "softness of movement", α_1 : "step size"$\}$, suppose the agent's current behavior is governed by $v_t = \langle \zeta_0, \zeta_1\rangle$, to increase the movement softness while maintaining current step size, the user can simply increase the α_0's score by some scalar Δ, and set the target attribute vector to $v_t' = \langle \zeta_0 + \Delta, \zeta_1\rangle$, with which the agent can update its behavior by optimizing $r_\theta([\cdot, v_t'])$. In our realization, we implement the process of finding the expected attribute score by taking user feedbacks as a process of performing binary search in real space (details can be found in Sec. A.2). The process of personalizing agent behavior through the learned attribute parameterized reward function r_θ is highly intuitive, because r_θ handles the complex and tacit parts of the problem internally (e.g., how to walk naturally and realistically with the constraints of softness and step size) and only relies on the end user to set the explicit parts.

4.3 Method 2: Encoding Behavioral Attributes by Capturing Minimally Viable Local Changes

One limitation of Method 1 in Section 4.2 is that, it implicitly requires the total number of encoded attributes to be finite because its reward function r_θ relies on the target attribute vector v_t as input to enable personalization, but the size of v_t can't grow indefinitely. This may limit the scalability of Method 1.

In this section, we will introduce a more extensible method that can potentially encode an arbitrary number of attributes. The key motivation is that whenever an end user expresses an intention to adjust the strength of a certain attribute, what he/she expects is essentially a change in behavior along the specified direction, even if this change is small (because the user can repeat this process until a desirable behavior is produced). Hence, given current behavior presented in a trajectory τ_c and the corresponding human feedback (α, h) on an attribute α, our goal is to construct a reward function that gives higher cumulative rewards to trajectories that have some small but noticeable change in α in the direction specified by h while keeping other unmentioned attributes unchanged (or minimally changed). We refer to such small but noticeable changes as *minimally viable local changes*, and the queried trajectory τ_c as the reference or *anchor trajectory*.

Specifically, different from Method 1, here we assume we are provided (again, normally by the agent builder, not the end user) a set of trajectory pairs $D_l = \{(\tau_c, \tau_t, \alpha, h)\}$, where τ_t is a trajectory

that reflects some minimally viable local changes to behavior in the anchor trajectory τ_c in terms of the specified attribute α and the direction h (h is a binary label indicating whether to increase or decrease the attribute strength). Given a tuple of $(\tau_c, \tau_t, \alpha, h) \in D_l$, a special reward function r_θ is supposed to learn to give higher cumulative rewards to τ_t over any other negative samples (such as trajectories that make excessive changes to τ_c, or trajectories that are not significantly different from τ_c, or trajectories that make changes to unspecified attributes). In practice, we select the negative samples by randomly sampling from the (mostly unlabelled) behavior dataset $\mathcal{D}$. Again, the reward function r_θ can be formulated as a modified Bradley-Terry model. It is supposed to capture the preferences between negative samples and $\{\tau_t\}$ and omit the orderings among negative samples:

$$\begin{aligned} P_\theta\left[\tau_t \succ \tau_n \middle| \alpha, h, \tau_c\right] &= \frac{\exp \sum_{s \in \tau_t} r_\theta\left(s|\alpha, h, \tau_c\right)}{\exp \sum_{s \in \tau_n} r_\theta\left(s|\alpha, h, \tau_c\right) + \exp \sum_{s \in \tau_t} r_\theta\left(s|\alpha, h, \tau_c\right)} \\ &= \frac{\exp \sum_{s \in \tau_t} r_\theta\left([s, e_\alpha, h, \phi(\tau_c)]\right)}{\exp \sum_{s \in \tau_n} r_\theta\left([s, e_\alpha, h, \phi(\tau_c)]\right) + \exp \sum_{s \in \tau_t} r_\theta\left([s, e_\alpha, h, \phi(\tau_c)]\right)}, \end{aligned} \tag{5}$$

where τ_n is a negative sample (i.e., trajectory), $[\cdot]$ is the vector concatenation operation, e_α as in Method 1 can be either an one-hot representation of attribute α or a sentence embedding output by Sentence-BERT, and $\phi(\tau_c)$ is a sequence encoder (e.g., an LSTM (Hochreiter & Schmidhuber, 1997)) that encodes the state sequence of the anchor trajectory τ_c to a compact latent representation. Note that $\phi(\cdot)$ is a sub-module of r_θ and it's jointly optimized with r_θ. Since r_θ is essentially a preference-based reward function, it can be optimized by employing a cross-entropy loss as in the one in Eq. 2. We note that our computational framework is similar to the Prompt-DT (Xu et al., 2022) in a sense that we both take a reference trajectory as a prompt to the model to obtain conditioned outputs. But instead of trying to replicate the behavior in the "prompt" trajectory as in Prompt-DT, our reward function learns to modify the "prompt" in a controlled way.

The use of r_θ is straightforward: at each round of interaction, the agent demonstrates its current behavior by presenting a trajectory τ to the user and collects some attribute-level feedbacks (α, h). Then the agent sets the latest trajectory τ as the anchor trajectory and feeds it along with the feedback information into the reward function $r_\theta(\cdot|\alpha, h, \tau_c)$. By optimizing r_θ, the agent will be able to make local changes to its behavior in the direction specified by the user. This process repeats until the desirable behavior is produced. Compared to Method 1 (Sec. 4.2) which requires a full specification of all the attributes' strengths, the reward function in Method 2 only takes one attribute as input at a time, because it is encoding how to get closer to the target by making local changes rather than directly capturing any specified behavior. This design is appealing as it offers better scalability and it affords the development of a big universal behavioral concept encoder. Nevertheless, Method 2 still has the following shortcomings: (a) it is less efficient than Method 1 in searching for the target behavior because it can only make minimally viable changes to the presented behavior; (b) The training data for Method 2 is harder to collect. Unlike Method 1, where any random subset of trajectories exhibiting distinct behaviors can be used as the training samples, in Method 2, the agent builder must carefully pick pairs of trajectories that reflect local changes. Also, the judgement of how much variation constitutes a minimally viable change can be fairly subjective.

5 Empirical Evaluation

As a proof of concept, we demonstrate the effectiveness of our methods in a varied set of 4 domains with 9 behavioral attributes:

Walker. This environment corresponds to a scenario that involves a 2-legged home-service robot walking around the house to perform household tasks. The users may want the robot to walk more softly at night. This environment is also related to physical character control scenarios wherein we want the character to move in a sneaky way. Two attributes are considered here: (a) step size; (b) softness of movement. To synthesize dataset and simulate human feedbacks, we use the moving speed and landing speed of the feet as proxies to measure the abstract concept softness or "sneaky". The environment is implemented based on the Walker-2d domain in the DeepMind control suite (Tunyasuvunakool et al., 2020).

Manipulator. We consider a virtual character control scenario wherein we want a simulated arm to mimic the ways of a human lifting objects. When humans, especially elders and children, are lifting heavy objects, their movement can be unstable. Hence, we consider two attributes: (a) moving speed of the arm; (b) instability of the movement. The environment is implemented based

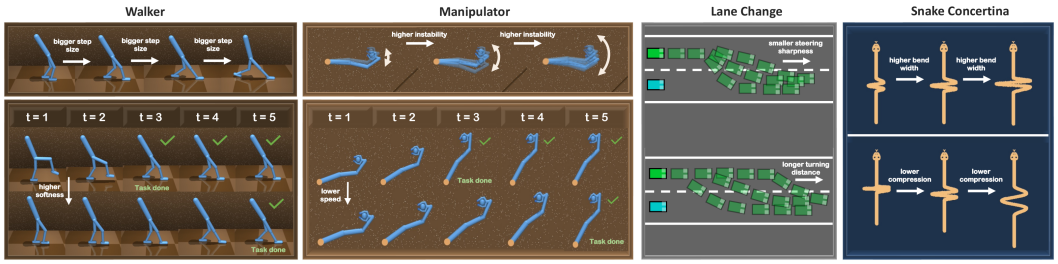

Figure 2: Visualizations of the domains and behavioral attributes.

on the Manipulator domain in the DeepMind control suite. We synthesize the behavior clips by hard coding the arm motion and by adding random noises in a controlled way. The purpose of conducting experiments in this domain is to demonstrate that our method can capture not only regular behavior patterns but also the irregularities. A Markov state is constructed by stacking five consecutive raw states of the environment to ensure it contains information about the irregularities.

Lane Change. We consider a day-to-day driving scenario wherein the rider would want to change the lane-changing behavior of the cab to get a more pleasant experience. Two attributes are used for evaluation: (a) the sharpness of steering: this attribute corresponds to how sharp a turn the agent makes while changing lanes; (b) distance to the following vehicle: this attribute is about the distance between our agent and the following car at the moment when our agent starts making the lane change. This environment is built on the highway environment in (Leurent, 2018). Note that the environment is an image based domain, so the objective here is to verify that our methods can be scaled to image inputs.

Snake Concertina. In this task, the agent is supposed to control a virtual snake character and imitate diverse concertina styles of a real snake's locomotion. There are three relevant attributes in concertina locomotion: (a) Width of the bend (i.e., the maximal width that the snake occupies); (b) Compression (i.e., how much the snake's body is compressed when it is moving); (c) Speed of movement.

In our experiment, all the behavior clips are generated either by hard programming the motions or by using reinforcement learning with sophisticated reward designs and hard-coded constraints.

Method	Lane-Change		Manipulator		Snake		Walker	
	SR	AF (std)	SR	AF (std)	SR	AF (std)	SR	AF (std)
RA-Global	0.95	3.95 *(2.43)*	1.0	2.8 *(1.21)*	**0.85**	**4.17 *(1.85)***	**1.0**	**3.75 *(1.47)***
RA-Global-L	**1.0**	**3.05 *(2.06)***	**1.0**	**2.5 *(1.32)***	0.8	6.38 *(5.03)*	0.95	3.78 *(2.25)*
RA-Local	0.7	7.07 *(3.49)*	0.7	12.23 *(5.75)*	0.55	6.45 *(3.82)*	0.95	5.47 *(3.52)*
RA-Local-L	0.9	5.78 *(4.04)*	0.8	10.75 *(6.35)*	0.6	4.17 *(2.11)*	0.9	5.16 *(3.08)*
PbRL	1.0	162.3 *(184)*	0.6	159.5 *(188.87)*	0.05	N/A	1.0	84.6 *(79.87)*

Table 1: SR - Success Rate; AF - Average Feedback (when success); RA - Relative Attribute; L - Language

5.1 Baseline and Results

For comparisons, we use the PbRL algorithm proposed in (Christiano et al., 2017; Lee et al., 2021), which learns a reward function from human preference labels collected by making active queries. Considering that our methods assume the additional access to the offline behavior dataset $\mathcal{D}$, we made a couple modifications to make PbRL into a stronger baseline. The most important one is that, to optimize the most recent reward function and update agent behavior after each query, rather than applying reinforcement learning, we use the policies extracted from $\mathcal{D}$. Specifically, we stochastically sample at large set of rollouts by executing the policies we used to synthesize dataset $\mathcal{D}$, and

the rollout with the highest cumulative rewards is set as the agent's latest behavior. In practice, these policies can be unsupervisedly learned from $\mathcal{D}$, and this is identical to the use of skill priors as in (Peng et al., 2018b; Luo et al., 2020; Peng et al., 2022), which first learns a diverse set of natural and plausible motions from offline behavior datasets to reduce online computation. We also employ this approach for policy optimization in our methods for the purpose of experiment. Besides, for PbRL, we also experimented with different query strategies and considered reusing previously trained reward models. For the sake of simplicity, we only report the best PbRL performance achieved at different setups.

As the evaluation metric, we count the number of human feedbacks (i.e., the binary preference labels in PbRL and the attribute-level feedback in our methods) needed to produce the target behavior. For each domain, we randomly sample 20 behavior configurations as targets, and the selected targets were unseen in the behavior dataset $\mathcal{D}$. A trial is considered as a success if the generated agent behavior has a ground truth attribute strength or proxy score that falls within a certain range of the target value. We use a threshold that roughly divides the strength of each attribute into five to ten buckets. A trail is deemed unsuccessful if the agent fails to produce the target behavior within 30 feedbacks in our methods, or 1000 feedbacks in PbRL. The results are shown in Table 5, where we denote Method 1 (Sec. 4.2) as RA-Global, and Method 2 (Sec. 4.3) as RA-Local. In addition, we add "L" as suffix to the names of variants that use language embedding as attribute representation. Results show that with relative behavioral attributes, users can specify agent behavior in an extremely efficient manner. The substantial performance gap between our methods and PbRL further confirms the necessity of maintaining a symbolic lingua franca between humans and agents. Despite the superior performance, the experimental results still reveal some defects in our methods. The rare failure cases of Method 1 suggest that it suffers from a saturation problem in that the attribute-conditioned ranking function f_σ tends to assign very close scores to behaviors at the two extremes (e.g., moving very softly or very recklessly). This limits the extrapolation capability of the system. Also, Method 2 has a lower success rate, indicating that its more complex formulation and architecture (e.g., the incorporation of a sequence encoder) to some extent affects its performance.

5.2 Additional Discussion

To get more insights into the number of labels needed to learn an accurate attribute ranking function or reward function in our methods, we conduct an extra experiment in which we train the model with different number of samples uniformly sampled from the training set, and evaluate each model on a held-out testing set. Results (Sec. A.4) show that to simultaneously learn two attributes, Method 1 needs around 200 labelled trajectories (i.e., the orderings among the trajectories) and Method 2 needs around 200 (τ_c, τ_t) pairs, which is a reasonable number. Also, the cost of learning a reward function is amortized when we continue to use it to support incoming users over its lifetime.

Recall that in the main experiment, we consider a trial as a successful one if the difference between the agent behavior and the target behavior is lower than a threshold. One interesting thing to see is whether our methods can achieve even higher-precision control over the agent behavior. Hence, we additionally experiment with a threshold value that roughly divides the attribute strengths into five to ten times more buckets. Surprisingly, PbRL fails to find the target behavior even once within 1000 feedbacks, while our methods can still succeed occasionally. Results are shown in Table 2.

6 Conclusion

In this paper, we introduced the notion of relative behavioral attributes which attempts to fill the gap between the two extremes of goal specification (i.e., exact specification and reward learning from feedback on trajectories). Users can provide symbolic feedback (i.e., their intent to increase/decrease attribute strength) to tweak and get desired agent behavior. We proposed two approaches and demonstrated their effectiveness through experiments in a varied set of domains. For future works, apart from the limitations we discussed earlier, it would be interesting if we could develop methods that combine the strengths of the two approaches proposed in this work. Also, currently, we use the sentence embedding of each attribute only as an alternative to the one-hot vector. However, it would be beneficial to make better use of the semantic structure inside the sentence embeddings.

REFERENCES

Junwen Bai, Weiran Wang, and Carla P Gomes. Contrastively disentangled sequential variational autoencoder. *Advances in Neural Information Processing Systems*, 34, 2021.

Andreea Bobu, Chris Paxton, Wei Yang, Balakumar Sundaralingam, Yu-Wei Chao, Maya Cakmak, and Dieter Fox. Learning perceptual concepts by bootstrapping from human queries. *arXiv preprint arXiv:2111.05251*, 2021.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pp. 783–792. PMLR, 2019.

Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9710–9719, 2021.

Sonam Goenka, Zhaoheng Zheng, Ayush Jaiswal, Rakesh Chada, Yue Wu, Varsha Hedau, and Pradeep Natarajan. Fashionvlp: Vision language transformer for fashion retrieval with feedback. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14105–14115, 2022.

Lin Guan, Mudit Verma, Suna Sihang Guo, Ruohan Zhang, and Subbarao Kambhampati. Widening the pipeline in human-guided reinforcement learning with explanation and context-aware data augmentation. *Advances in Neural Information Processing Systems*, 34:21885–21897, 2021.

Lin Guan, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging approximate symbolic models for reinforcement learning via skill diversity. *arXiv preprint arXiv:2202.02886*, 2022.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pp. 540–550. AAAI Press, 2020.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142, 2002.

Subbarao Kambhampati, Sarath Sreedharan, Mudit Verma, Yantian Zha, and Lin Guan. Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems. In *AAAI*, 2022.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pp. 2668–2677. PMLR, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16, 2009.

Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. *arXiv preprint arXiv:2106.05091*, 2021.

Lisa Lee, Ben Eysenbach, Russ R Salakhutdinov, Shixiang Shane Gu, and Chelsea Finn. Weakly-supervised reinforcement learning for controllable behavior. *Advances in Neural Information Processing Systems*, 33:2661–2673, 2020.

Seong Jae Lee and Zoran Popović. Learning behavior styles with inverse reinforcement learning. *ACM transactions on graphics (TOG)*, 29(4):1–7, 2010.

Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

Ying-Sheng Luo, Jonathan Hans Soeseno, Trista Pei-Chun Chen, and Wei-Chao Chen. Carl: Controllable agent with reinforcement learning for quadruped locomotion. *ACM Transactions on Graphics (TOG)*, 39(4):38–1, 2020.

James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L Roberts, Matthew E Taylor, and Michael L Littman. Interactive learning from policy-dependent human feedback. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2285–2294. JMLR. org, 2017.

Parsa Mahmoudieh, Deepak Pathak, and Trevor Darrell. Zero-shot reward specification via grounded natural language. In *ICML*, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Devi Parikh and Kristen Grauman. Relative attributes. In *2011 International Conference on Computer Vision*, pp. 503–510. IEEE, 2011.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018a.

Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions On Graphics (TOG)*, 37(6): 1–14, 2018b.

Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Trans. Graph.*, 40 (4), July 2021. doi: 10.1145/3450626.3459670. URL `http://doi.acm.org/10.1145/3450626.3459670`.

Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *arXiv preprint arXiv:2205.01906*, 2022.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL `https://arxiv.org/abs/1908.10084`.

Stuart J. Russell and Peter Norvig. Artificial intelligence - a modern approach, 2nd edition. In *Prentice Hall series in artificial intelligence*, 2003.

Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic skills for bilevel planning. *arXiv preprint arXiv:2206.10680*, 2022.

Sarath Sreedharan, Utkarsh Soni, Mudit Verma, Siddharth Srivastava, and Subbarao Kambhampati. Bridging the gap: Providing post-hoc symbolic explanations for sequential decision-making problems with inscrutable representations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=o-1v9hdSult.

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.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. *Advances in Neural Information Processing Systems*, 30, 2017.

Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. *Advances in neural information processing systems*, 25, 2012.

Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *International Conference on Machine Learning*, pp. 24631–24645. PMLR, 2022.

Fangkai Yang, Daoming Lyu, Bo Liu, and Steven Gustafson. Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. *arXiv preprint arXiv:1804.07779*, 2018.

Allan Zhou and Anca D Dragan. Cost functions for robot motion style. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3632–3639. IEEE, 2018.

A Appendix

A.1 Alternative ways to use the attribute-conditioned ranking function in Method 1

Recall that in Method 1, given a trained attribute-conditioned ranking function f_σ and a finite set of attributes $\mathcal{A}$, the agent behavior in any trajectory τ can be represented by an attribute vector $v(\tau) = \langle \zeta(\tau, \alpha_1), ..., \zeta(\tau, \alpha_k) \rangle$, where $k = |\mathcal{A}|$ is the number of attributes. If we assume the optimal policy is approximated via some parametric model (e.g., neural networks), we can actually skip the learning of the attribute parameterized reward function by viewing the attribute vectors as skill latent codes and learning a versatile policy conditioned on them. This is similar to the operations in (Wang et al., 2017; Peng et al., 2022) but with a more structured latent code. Also, one might create a sparse reward given at the end of episode by computing the distance between the extracted attribute vector and the target attribute vector. But such a reward is usually hard to optimize.

A.2 Finding the target attribute scores with binary search

Binary search is highly efficient because it narrows down the search space by cutting it in half at each step. In order to apply binary search in the attribute space, we need to maintain beliefs about the upper and lower bounds of each attribute. In our case, the upper and lower bounds can be initialized to the maximum and minimum attribute scores observed in the offline behavioral dataset $\mathcal{D}$, where the scores are given by the trained attribute conditional ranking function f_σ. At each

Method	Lane-Change		Manipulator		Snake		Walker	
	SR	AF (std)	SR	AF (std)	SR	AF (std)	SR	AF (std)
RA-Global	0.60	6.75 *(2.98)*	0.80	7.00 *(2.45)*	**0.55**	**8.00** ***(3.67)***	**0.70**	**7.29** ***(2.63)***
RA-Global-L	**0.7**	**6.79** ***(1.66)***	**0.75**	**5.47** ***(3.2)***	0.25	10.4 *(6.02)*	0.75	6.00 *(3.59)*
RA-Local	0.25	5.00 *(0.63)*	0.45	19.44 *(5.57)*	0.30	8.16 *(3.18)*	0.40	9.37 *(4.15)*
RA-Local-L	0.4	8.63 *(2.45)*	0.30	10.5 *(5.5)*	0.50	7.2 *(4.621)*	0.40	6.25 *(2.48)*
PbRL	0	N/A	0	N/A	0	N/A	0	N/A

Table 2: Results on controlling the agent's behavior with higher-precision. SR - Success Rate; AF - Average Feedback (when success); RA - Relative Attribute; L - Language

query step, the agent presents to the human the behavior corresponding to the median attribute value, i.e., $\frac{\alpha_{upper}+\alpha_{lower}}{2}$, and updates the beliefs accordingly after getting the feedback. In the case of extrapolation, we can also go beyond the maximum and minimum attribute strengths, but this is no longer a binary search.

A.3 Architectures and Hyperparameters

When one-hot representation is used to represent attributes, both f_σ and r_θ in Method 1 employ a 3-layer fully-connected network with 512 hidden neurons as the architecture. In Method 2, for the trajectory encoder, we use a 2-layer bi-directional LSTM with 128 as the hidden dim. The trajectory embedding, along with the input state and attribute, are fed into a 3-layer fully-connected network with 512 hidden neurons to compute the reward.

When sentence embeddings are used as attribute representation, for both Method 1 and Method 2, we increase the number of hidden neurons in fully-connected layers from 512 to 1024 due to the increase in the size of attribute embedding (size 768).

A.4 Number of training samples needed in our methods

Fig. 4 and Fig. 5 show the performance (on a held-out testing set) of Method 1 and Method 2 when different numbers of training samples are used. For Method 1, given an ordered trajectory pair $(\tau_1 \succ_\alpha \tau_2)$, the ranking function is converted into a binary classifier that predicts the ordering of the given pair and the performance is measured in terms of the accuracy of this binary classifier. Similarly for Method 2, the performance is measured by converting Equation 5 to a binary classification problem where the function predicts whether a trajectory is the target trajectory or not. Note that the sample complexity of the two methods are not directly comparable because their training samples are in different formats.

A.5 Discussion on Unsupervised Discovery of Behavioral Attribute

As a preliminary attempt in this study, we explored the possibility of unsupervised discovery of concepts/properties. We applied a state-of-the-art disentangled sequential variational autoencoder method, C-DSVAE (Bai et al., 2021), to learn to encode behaviors in the offline behavior dataset $\mathcal{D}$. Though the behavior/motion embeddings given by C-DSVAE are able to cluster visually similar traces together, it still has difficulty capturing subtle but meaningful differences in behaviors. More importantly, it fails to establish any meaningful ordering among behaviors. As an example (Fig. 3), we visualize the variations encoded in a specific dimension in the latent space of the Lane-Change domain. This observation is consistent with current trend in vision-text research, which suggests unless additional supervision signals are provided, representations developed by neural networks are not guaranteed to capture semantics in the inputs that make sense to humans. This preliminary study confirms the necessity to employ supervised learning for behavioral attribute modeling.

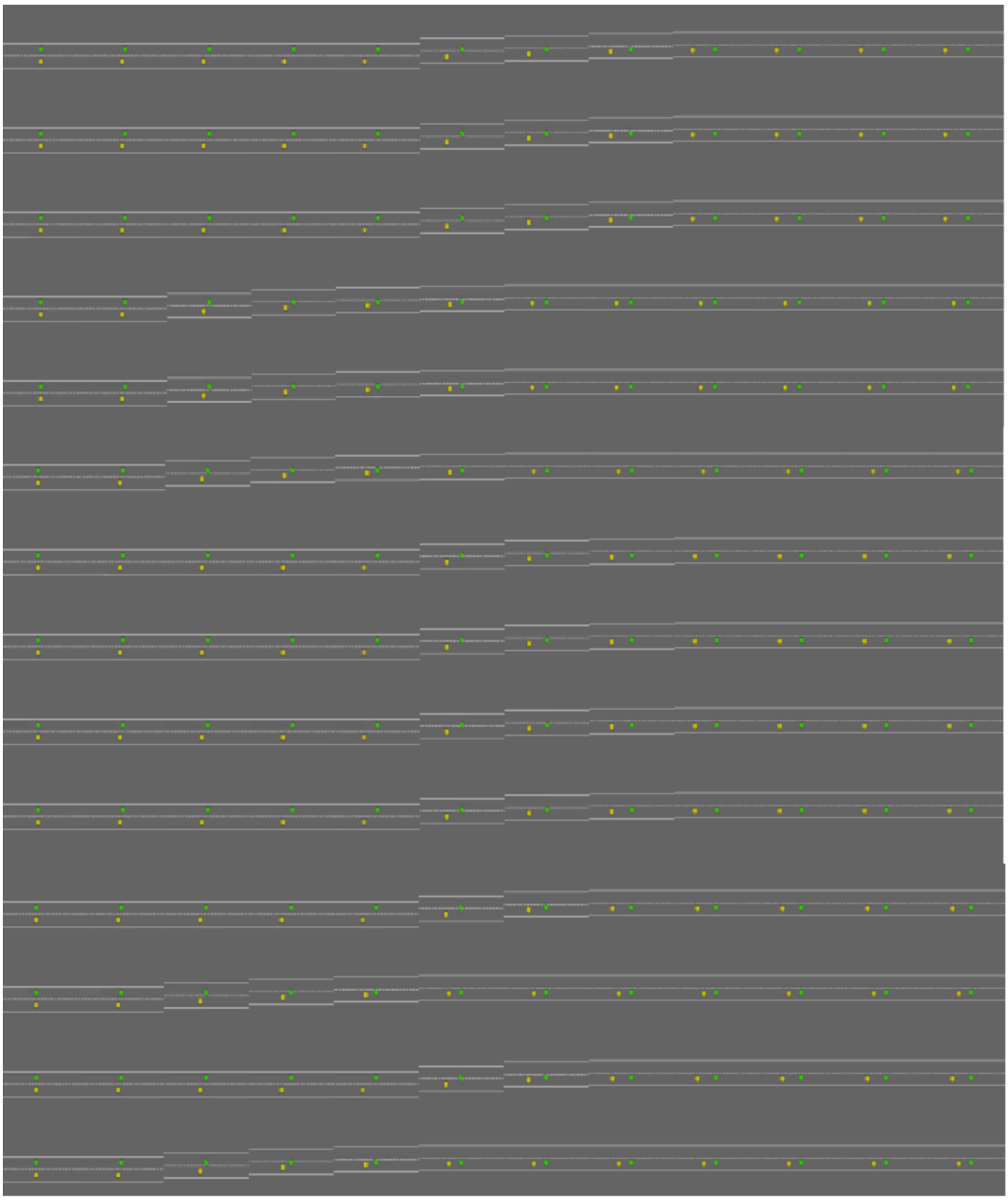

Figure 3: A failure case of unsupervised concept discovery. Each row corresponds to a behavior trace.

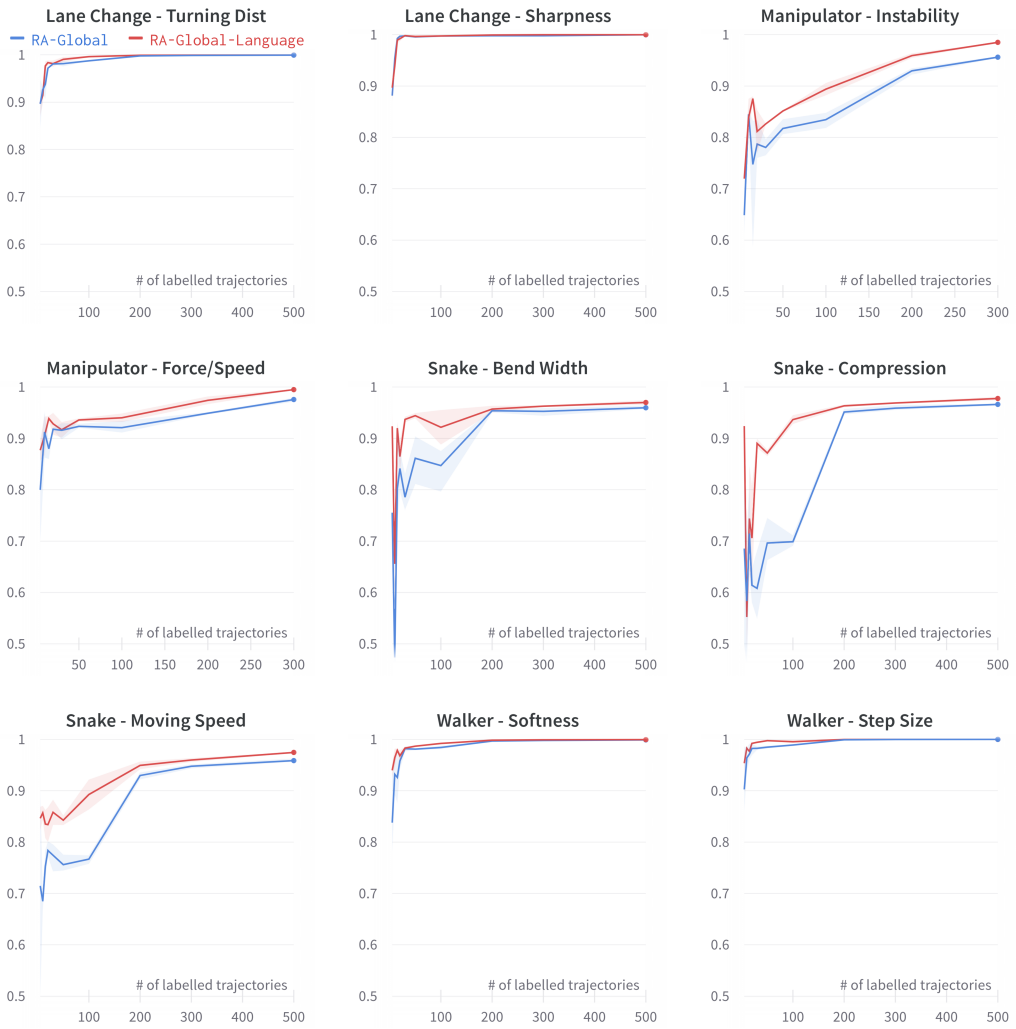

Figure 4: Performance of the ranking function in Method 1 versus # of training samples .

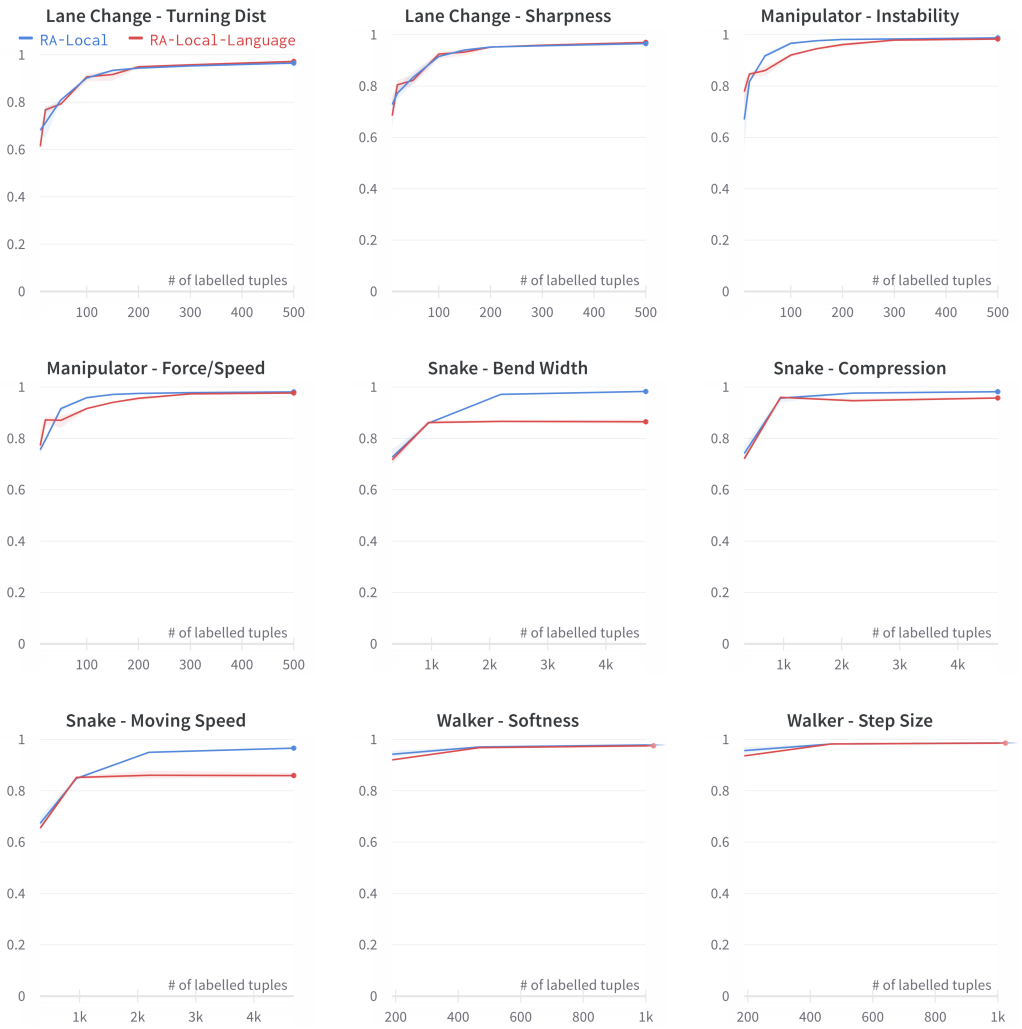

Figure 5: Performance of Method 2 versus # of training samples in Method 2.