Residual Q-Learning: Offline and Online Policy Customization without Value

Chenran Li[*]
Department of Mechanical Engineering
University of California Berkeley
chenran_li@berkeley.edu

Chen Tang[*]
Department of Mechanical Engineering
University of California Berkeley
chen_tang@berkeley.edu

Haruki Nishimura
Toyota Research Institute, USA
haruki.nishimura@tri.global

Jean Mercat
Toyota Research Institute, USA
haruki.nishimura@tri.global

Masayoshi Tomizuka
Department of Mechanical Engineering
University of California Berkeley
tomizuka@berkeley.edu

Wei Zhan
Department of Mechanical Engineering
University of California Berkeley
wzhan@berkeley.edu

Abstract

Imitation Learning (IL) is a widely used framework for learning imitative behavior from demonstrations. It is especially appealing for solving complex real-world tasks where handcrafting reward function is difficult, or when the goal is to mimic human expert behavior. However, the learned imitative policy can only follow the behavior in the demonstration. When applying the imitative policy, we may need to *customize* the policy behavior to meet different requirements coming from diverse downstream tasks. Meanwhile, we still want the customized policy to maintain its imitative nature. To this end, we formulate a new problem setting called *policy customization*. It defines the learning task as training a policy that inherits the characteristics of the prior policy while satisfying some additional requirements imposed by a target downstream task. We propose a novel and principled approach to interpret and determine the trade-off between the two task objectives. Specifically, we formulate the customization problem as a Markov Decision Process (MDP) with a reward function that combines 1) the inherent reward of the demonstration; and 2) the add-on reward specified by the downstream task. We propose a novel framework, *Residual Q-learning*, which can solve the formulated MDP by leveraging the prior policy *without knowing the inherent reward or value function of the prior policy*. We derive a family of residual Q-learning algorithms that can realize offline and online policy customization, and show that the proposed algorithms can effectively accomplish policy customization tasks in various environments. Videos are available on our website: `https://sites.google.com/view/residualq-learning`.

1 Introduction

Imitation learning (IL) is a widely used framework for learning imitative behavior from demonstrations. It is especially appealing for solving complicated real-world tasks where handcrafting reward

---

[*]Equal Contribution

function is difficult, or when the goal is to mimic expert behavior. For instance, IL has been widely studied for synthesizing human-like agents from large-scale human demonstrations on challenging real-world tasks, such as autonomous driving [4, 7, 2] and robot manipulation [50, 33, 34]. However, when applying the imitative policy in open real-world scenarios, we may encounter different circumstances that impose diverse requirements beyond merely following the behavior in the demonstration, such as reaching particular goals [39], enforcing safety [32], or complying with certain behavior preferences [51]. In this case, we need to *customize* the imitative policy to meet those task-specific objectives. Meanwhile, in many applications, we may still want the customized policy to possess some inherent characteristics of the expert demonstration (e.g., human-like).

To this end, we formulate a new problem setting we refer to as *policy customization*. In the policy customization problem, we are given a pre-trained policy, and the task is to train a new policy that 1) inherits the properties of the prior policy; 2) satisfies some additional requirements imposed by a given downstream task. At first glance, this newly formulated problem may seem to be seamlessly solved by existing reinforcement learning (RL) fine-tuning algorithms [38, 35, 51, 43], where RL is applied to fine-tune a pre-trained policy obtained from behavior cloning (BC) or offline RL [27, 25, 30]. In particular, some RL fine-tuning methods combine different forms of IL objectives with RL to accelerate policy training, e.g., maximizing the policy likelihood on demonstration [38], regularizing the KL-divergence between the fine-tuned and prior policies [35, 51]. Different from the policy customization problem we consider, the IL objectives are introduced heuristically as regularization in RL fine-tuning. Solving policy customization instead requires a *principled* way to design the learning objective to jointly optimize the policy's performance on the imitative and downstream tasks, which has not been well-studied in the RL fine-tuning literature.

In this work, we propose such a principled approach to interpret and design the policy customization objective with a solid theoretical basis. Formally, we formulate policy customization as a Markov decision process (MDP) whose reward function is a linear combination of 1) the inherent reward of the demonstration; and 2) an add-on reward specified by the downstream task. The reward weight directly determines the trade-off between the imitative and downstream tasks for policy optimization. However, we cannot solve this MDP with existing RL algorithms, since we can only access the imitative policy without knowing its underlying reward.

To tackle this challenge, we propose a novel *Residual Q-Learning* framework that can solve the formulated MDP *without knowing the inherent reward or value function of the prior policy*. Specifically, we define a residual Q-function that, together with the log-likelihood of the prior policy, can conveniently construct the maximum-entropy policy solving the target MDP. We show that the residual Q-function can be iteratively learned with an update rule derived from soft Bellman equation [16]. Hence, the proposed residual Q-learning method can be applied on top of any value-based maximum-entropy RL algorithms. In particular, we derive residual soft Q-learning and residual soft actor-critic—two model-free policy customization algorithms through RL fine-tuning—which are adapted from soft Q-learning [16] and soft actor-critic [17] respectively. In addition, we introduce a model-based policy customization algorithm adapted from maximum-entropy Monte Carlo Tree Search (MCTS) [47], which enables *zero-shot online* policy customization when a dynamics model is available. In our experiments, we show that the proposed residual Q-learning algorithms can effectively customize the policies toward the additional task objective while maintaining their performance on the basic tasks for which the prior policies are trained.

2 Policy Customization with Residual Q-Learning

2.1 Problem Formulation: Policy Customization

We consider a MDP defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Without loss of generality, we introduce the formulation under continuous state and action spaces. The state transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ represents the probability density of the next state $\boldsymbol{s}_{t+1} \in \mathcal{S}$ given the current state $\boldsymbol{s}_t \in \mathcal{S}$ and action $\boldsymbol{a}_t \in \mathcal{A}$. While the following formulation applies whether the reward function r is given or not, we are primarily interested in the case when r remains *unknown* and, instead, expert demonstrations are available to indicate desirable behaviors. We assume that we have access to a maximum-entropy policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, \infty)$ pre-trained with imitation learning [14, 21] methods. The policy π maximizes the entropy-augmented cumulative reward that is inherent in the

demonstrations. The resulting optimal policy follows a Boltzmann distribution [16] as follows:

$$\pi\left(\boldsymbol{a}|\boldsymbol{s}\right) = \frac{1}{Z_s}\exp\left(\frac{1}{\alpha}Q^*(\boldsymbol{s},\boldsymbol{a})\right),\tag{1}$$

where $Q^*(\boldsymbol{s},\boldsymbol{a})$ is the soft Q-function as defined in [16] and it satisfies the soft Bellman equation:

$$Q^*(\boldsymbol{s},\boldsymbol{a}) = r(\boldsymbol{s},\boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{s}'\sim p(\cdot|\boldsymbol{s},\boldsymbol{a})}\left[\alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^*(\boldsymbol{s}',\boldsymbol{a}')\right)d\boldsymbol{a}'\right],\tag{2}$$

and Z_s is the normalization factor defined as $\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^*(\boldsymbol{s},\boldsymbol{a})\right)d\boldsymbol{a}$. The temperature coefficient α determines the relative importance of entropy and reward in the maximum-entropy policy [16]. It is straightforward to derive from Eqn. (1) that:

$$Q^*(\boldsymbol{s},\boldsymbol{a}) = \alpha\log\pi\left(\boldsymbol{a}|\boldsymbol{s}\right) + \alpha\log Z_s.\tag{3}$$

Instead of directly deploying the policy π, we aim to *customize* it to satisfy some additional requirements for a given downstream task, specified by an add-on reward function $r_R : \mathcal{S}\times\mathcal{A}\to\mathbb{R}$. Meanwhile, we want the customized policy to inherit the characteristics of the original policy. Formally, we aim to leverage the pre-trained prior policy π to train a new policy for the MDP defined by $\hat{\mathcal{M}} = (\mathcal{S},\mathcal{A},\omega r + r_R, p)$. The new reward function is defined as a weighted sum of the basic reward r and the add-on reward r_R so that the customized policy meets both requirements. Under this setting, the main challenge lies in how to find the optimal policy for the new task without knowing the prior reward r but only the imitative policy π.

2.2 Residual Q-Learning

Since the reward r is unknown, we cannot directly train the new policy to maximize the total reward through RL. Instead, it turns out that we can leverage the imitative policy π to construct the new policy *without knowing π's underlying reward or value function*. The proposed method can be applied on top of any *value-based maximum-entropy* RL algorithms [16, 17, 47], which rely on the soft Bellman update operator defined as:

$$\hat{Q}_{t+1}(\boldsymbol{s},\boldsymbol{a}) = r_R(\boldsymbol{s},\boldsymbol{a}) + \omega r(\boldsymbol{s},\boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{s}'\sim p(\cdot|\boldsymbol{s},\boldsymbol{a})}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\hat{Q}_t(\boldsymbol{s}',\boldsymbol{a}')\right)d\boldsymbol{a}'\right],\tag{4}$$

where $\hat{Q}_t$ is the estimated soft Q-function for $\hat{\mathcal{M}}$ at the tth iteration. Instead of directly applying the soft Bellman update operator to iterate $\hat{Q}_t$ during policy learning, we define a *residual* Q-function as $Q_{R,t} := \hat{Q}_t - \omega Q^*$ and attempt to iterate this residual Q-function during policy learning. We then derive an update rule for $Q_{R,t}$ that does not depend on the reward or value function of π:

$$Q_{R,t+1}(\boldsymbol{s},\boldsymbol{a})$$

$$= r_R(\boldsymbol{s},\boldsymbol{a}) + \omega r(\boldsymbol{s},\boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{s}'}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\hat{Q}_t(\boldsymbol{s}',\boldsymbol{a}')\right)d\boldsymbol{a}'\right] - \omega Q^*(\boldsymbol{s},\boldsymbol{a}),\tag{5a}$$

$$= r_R(\boldsymbol{s},\boldsymbol{a}) + \omega Q^*(\boldsymbol{s},\boldsymbol{a}) - \omega\gamma\mathbb{E}_{\boldsymbol{s}'}\left[\alpha\log\int_{\mathcal{A}}\exp\left(\frac{1}{\alpha}Q^*(\boldsymbol{s}',\boldsymbol{a}')\right)d\boldsymbol{a}'\right]$$

$$+ \gamma\mathbb{E}_{\boldsymbol{s}'}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\hat{Q}_t(\boldsymbol{s}',\boldsymbol{a}')\right)d\boldsymbol{a}'\right] - \omega Q^*(\boldsymbol{s},\boldsymbol{a}),\tag{5b}$$

$$= r_R(\boldsymbol{s},\boldsymbol{a}) - \omega\alpha\gamma\mathbb{E}_{\boldsymbol{s}'}\log Z_{s'}$$

$$+ \gamma\mathbb{E}_{\boldsymbol{s}'}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\left(Q_{R,t}(\boldsymbol{s}',\boldsymbol{a}') + \omega\alpha\log\pi\left(\boldsymbol{a}'|\boldsymbol{s}'\right) + \omega\alpha\log Z_{s'}\right)\right)d\boldsymbol{a}'\right],\tag{5c}$$

$$= r_R(\boldsymbol{s},\boldsymbol{a}) - \omega\alpha\gamma\mathbb{E}_{\boldsymbol{s}'}\log Z_{s'} + \omega\alpha\gamma\mathbb{E}_{\boldsymbol{s}'}\log Z_{s'}$$

$$+ \gamma\mathbb{E}_{\boldsymbol{s}'}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\left(Q_{R,t}(\boldsymbol{s}',\boldsymbol{a}') + \omega\alpha\log\pi\left(\boldsymbol{a}'|\boldsymbol{s}'\right)\right)\right)d\boldsymbol{a}'\right],\tag{5d}$$

$$= r_R(\boldsymbol{s},\boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{s}'}\left[\hat{\alpha}\log\int_{\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\left(Q_{R,t}(\boldsymbol{s}',\boldsymbol{a}') + \omega\alpha\log\pi\left(\boldsymbol{a}'|\boldsymbol{s}'\right)\right)\right)d\boldsymbol{a}'\right],\tag{5e}$$

3

where Eqn. (5b) results from expressing the reward r with the soft Q-function Q^* based on Eqn. (2). Eqn. (5c) results from the definition of the residual Q-function and Eqn. (3). In each iteration, we can then define the policy corresponding to the current estimated $\hat{Q}_t$ without computing $\hat{Q}_t$:

$$\hat{\pi}_t(\boldsymbol{a}|\boldsymbol{s}) \propto \exp\left(\frac{1}{\hat{\alpha}}\left(Q_{R,t}(\boldsymbol{s},\boldsymbol{a}) + \omega\alpha\log\pi(\boldsymbol{a}|\boldsymbol{s})\right)\right). \tag{6}$$

One remaining issue is that we do not have access to the underlying temperature coefficient α of the prior policy π. Note that α is multiplied upon the reward weight ω in Eqn. (5) and Eqn. (6). In practice, the reward weight is a hyperparameter to tune. Hence, we can define $\omega' = \omega\alpha$ and directly treat ω' as the hyperparameter without the need of knowing the true α.

3 Practical Algorithms

In this section, we present three practical algorithms for policy customization under different settings based on the update rule of residual Q-function in Eqn. (5) and the maximum-entropy policy defined in Eqn. (6). In these algorithms, the goal is to find a policy that solves the MDP $\hat{\mathcal{M}}$ given a prior policy π and the add-on reward function r_R. We first introduce two model-free algorithms to learn the residual Q-function and customize the policy through additional RL steps, which are adapted from soft Q-learning [16] and soft actor-critic [17] respectively. Then we introduce a model-based policy customization algorithm adapted from maximum-entropy MCTS [47], which enables *zero-shot online* policy customization when a dynamics model is available.

Residual Soft Q-Learning. We adapt the soft Q-learning [16] algorithm to a residual soft Q-learning algorithm for offline policy customization (i.e., policy customization via additional RL training steps) when the action space is discrete. The residual algorithm is essentially the same as the soft Q-learning. The only difference is that we use a function approximator parameterized by θ to model the residual Q-function instead of the soft Q-function and denote it as $Q_{R,\theta}(\boldsymbol{s},\boldsymbol{a})$. At each iteration, the parameters are updated by taking gradient steps to minimize the temporal-difference (TD) error:

$$J_{Q_R}(\theta) = \mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t,r_{R,t},\boldsymbol{s}_{t+1})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_{R,\bar{\theta}}^{\text{target}}(\boldsymbol{s}_t,\boldsymbol{a}_t) - Q_{R,\theta}(\boldsymbol{s}_t,\boldsymbol{a}_t)\right)^2\right], \tag{7}$$

where $\mathcal{D}$ is the replay buffer and the target residual Q-value $Q_{R,\bar{\theta}}^{\text{target}}(\boldsymbol{s}_t,\boldsymbol{a}_t)$ is computed as:

$$Q_{R,\bar{\theta}}^{\text{target}}(\boldsymbol{s}_t,\boldsymbol{a}_t) = r_{R,t} + \gamma\hat{\alpha}\log\sum_{\boldsymbol{a}'\in\mathcal{A}}\exp\left(\frac{1}{\hat{\alpha}}\left(Q_{R,\bar{\theta}}(\boldsymbol{s}_{t+1},\boldsymbol{a}') + \omega'\log\pi(\boldsymbol{a}'|\boldsymbol{s}_{t+1})\right)\right), \tag{8}$$

with $\bar{\theta}$ being the target parameters. With a discrete action space, we can compute the policy distribution exactly given the residual Q-function and the prior policy as in Eqn. (6).

Residual Soft Actor-Critic. For tasks with continuous action spaces, we adapt the soft actor-critic [17] algorithm to a residual soft actor-critic algorithm. We aim to train a parameterized residual Q-function $Q_{R,\theta}(\boldsymbol{s},\boldsymbol{a})$ and a policy network $\hat{\pi}_\phi(\boldsymbol{a}|\boldsymbol{s})$ by alternating between policy evaluation and improvement. In the policy evaluation step, we update the residual Q-function for evaluating the current policy. The loss function is essentially the one in Eqn. (7) but with the target residual Q-value computed with respect to the current policy $\hat{\pi}_\phi$:

$$Q_{R,\bar{\theta}}^{\text{target}}(\boldsymbol{s}_t,\boldsymbol{a}_t) = r_{R,t} + \gamma\mathbb{E}_{\boldsymbol{a}'\sim\hat{\pi}_\phi}\left[Q_{R,\bar{\theta}}(\boldsymbol{s}_{t+1},\boldsymbol{a}') + \omega'\log\pi(\boldsymbol{a}'|\boldsymbol{s}_{t+1}) - \hat{\alpha}\log\hat{\pi}_\phi(\boldsymbol{a}'|\boldsymbol{s}_{t+1})\right] \tag{9}$$

The formula for the residual Q-value is derived from the modified Bellman update operator of soft actor-critic [17] following a similar procedure as in Eqn. (5). The detailed derivation is attached to Appendix A for completeness. In the policy improvement step, we improve the policy $\hat{\pi}_\phi$ leveraging the current estimated residual Q-value $Q_{R,\theta}(\boldsymbol{s},\boldsymbol{a})$ and the prior policy π by minimizing the expected KL-divergence between $\hat{\pi}_\phi$ and the desired maximum-entropy policy given the current $Q_{R,\theta}(\boldsymbol{s},\boldsymbol{a})$, which is equivalent to minimizing the loss function below:

$$J_{\hat{\pi}}(\phi) = \mathbb{E}_{\boldsymbol{s}_t\sim\mathcal{D}}\left[\mathbb{E}_{\boldsymbol{a}\sim\hat{\pi}_\phi}\left[\hat{\alpha}\log\hat{\pi}_\phi(\boldsymbol{a}|\boldsymbol{s}_t) - Q_{R,\theta}(\boldsymbol{s}_t,\boldsymbol{a}) - \omega'\log\pi(\boldsymbol{a}|\boldsymbol{s}_t)\right]\right]. \tag{10}$$

In our experiments, we implement the algorithm based on the practical soft actor-critic algorithm presented in [18], which incorporates auto entropy adjustment and double residual Q-functions [13].

4

Residual Maximum-Entropy MCTS. The last algorithm we introduce is a model-based policy customization algorithm adapted from the maximum-entropy MCTS [47] algorithm, which we refer to as residual maximum-entropy MCTS. Our primary interest is in applying this algorithm in the online setting with an offline learned dynamics model. MCTS essentially solves an online planning problem leveraging the prior policy and add-on reward function, which enables *zero-shot online* policy customization. Such a paradigm is particularly useful for problem domains like autonomous driving, where large-scale human driving datasets [11, 49, 45] are available for learning good imitative policies [1, 40] as well as accurate traffic predictive models for online planning [10, 29]. Note that MCTS-based methods inherently assume a discrete action space. In practice, we may extend it to tasks with continuous action space by discretizing the action space [41, 48].

To solve the online planning problem, maximum-entropy MCTS constructs a look-ahead tree $\mathcal{T}$ incrementally via Monte Carlo simulation. In the tree $\mathcal{T}$, each node $n(\boldsymbol{s}) \in \mathcal{T}$ corresponds to a state s. For each action $a \in \mathcal{A}$, the node $n(\boldsymbol{s})$ stores an estimated soft Q-value $\hat{Q}(\boldsymbol{s}, \boldsymbol{a})$ and a visit count $N(\boldsymbol{s}, \boldsymbol{a})$. At each iteration, simulations are conducted starting from the root node of the search tree, which consists of two stages: 1) querying a *tree policy* to select actions and traversing the tree until a leaf node is reached; 2) querying an *evaluation function* (e.g., Monte Carlo simulations leveraging a roll-out policy) at the leaf node to estimate the return-to-go. After simulation, the estimated returns are propagated backward to update all the nodes along the simulated paths. Afterward, the tree is grown by expanding the leaf nodes reached during the simulations. Different from a typical MCTS algorithm [8], maximum-entropy MCTS exploits maximum-entropy policy optimization. Hence, the tree policy used in maximum-entropy MCTS is defined as:

$$\hat{\pi}(\boldsymbol{a}|\boldsymbol{s}) = (1 - \lambda_s) \frac{\exp(\frac{1}{\hat{\alpha}} \hat{Q}(\boldsymbol{s}, \boldsymbol{a}))}{\sum_{\boldsymbol{a}' \in \mathcal{A}} \exp(\frac{1}{\hat{\alpha}} \hat{Q}(\boldsymbol{s}, \boldsymbol{a}))} + \lambda_s \frac{1}{|\mathcal{A}|}, \tag{11}$$

where $\lambda_s = {}^{\epsilon|\mathcal{A}|}/\log(\sum_{\boldsymbol{a}} N(\boldsymbol{s},\boldsymbol{a})+1)$ with ϵ as a hyperparameter. Given a simulated trajectory from the root node to a leaf node, i.e., $\{\boldsymbol{s}_0, \boldsymbol{a}_0, \cdots, \boldsymbol{s}_T, \boldsymbol{a}_T\}$, the Q-values of all the visited nodes are updated with a *softmax backup*:

$$\hat{Q}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \begin{cases} r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \omega r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \hat{R} & t = T - 1, \\ r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \omega r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \hat{\alpha} \log \sum_{\boldsymbol{a}' \in \mathcal{A}} \exp\left(\frac{1}{\hat{\alpha}} \hat{Q}(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\right) & t < T - 1, \end{cases} \tag{12}$$

where $\hat{R}$ is the return-to-go estimated by querying an evaluation function on the leaf node $n(\boldsymbol{s}_T)$. To adapt the algorithm to residual maximum-entropy MCTS, we instead have each node store an estimated residual Q-value $Q_R(\boldsymbol{s}, \boldsymbol{a})$. The tree policy is revised as:

$$\hat{\pi}(\boldsymbol{a}|\boldsymbol{s}) = (1 - \lambda_s) \frac{\exp(\frac{1}{\hat{\alpha}} (Q_R(\boldsymbol{s}, \boldsymbol{a}) + \omega' \log \pi(\boldsymbol{a}|\boldsymbol{s})))}{\sum_{\boldsymbol{a}' \in \mathcal{A}} \exp(\frac{1}{\hat{\alpha}} (Q_R(\boldsymbol{s}, \boldsymbol{a}) + \omega' \log \pi(\boldsymbol{a}|\boldsymbol{s})))} + \lambda_s \frac{1}{|\mathcal{A}|}. \tag{13}$$

During backward propagation, the residual Q-value is updated as follows:

$$Q_R(\boldsymbol{s}_t, \boldsymbol{a}_t) = \begin{cases} r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma R & t = T - 1, \\ r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V_R(\boldsymbol{s}_{t+1}) & t < T - 1, \end{cases} \tag{14}$$

with $V_R(\boldsymbol{s}_{t+1}) = \hat{\alpha} \log \sum_{\boldsymbol{a}' \in \mathcal{A}} \exp\left(\frac{1}{\hat{\alpha}} (Q_R(\boldsymbol{s}_{t+1}, \boldsymbol{a}') + \omega' \log \pi(\boldsymbol{a}'|\boldsymbol{s}_{t+1}))\right)$. In particular, we use Monte Carlo simulations to estimate the return-to-go R. The roll-out policy is set as the prior policy π and R is computed as the cumulative return of the add-on reward r_R over the roll-out trajectories. While the estimated R is biased, it serves as a heuristic to guide the tree search to select nodes with future trajectories collecting high add-on rewards while staying close to the prior policy. The effect of this biased estimation will gradually mitigate as the search tree expands to the terminal time step.

4 Related Work

Our residual Q-learning framework can be considered a principled approach to combine the objectives of IL and RL. Prior works have proposed different approaches for this purpose. In the IL literature, reward augmentation [9] has been introduced as a general framework to provide additional incentives from prior knowledge into the imitation learning process. Under the Generative Adversarial Imitation Learning (GAIL) framework, a surrogate reward can be conveniently incorporated to augment the

learned discriminator for policy optimization [31, 3]. The augmented imitative reward is essentially the same as the total reward in our framework. The difference is that ours aims to further customize a trained imitative policy for downstream tasks, instead of regularizing IL with prior knowledge.

There are also a variety of works that directly design the objective function as a weighted mixture of IL and RL objectives. In some methods, like DQfD [20], a supervised classification loss is combined with a TD loss to train the Q-function. It regularizes the learned Q-function to assign higher values to demonstrated actions. Other methods, such as DAPG [38] and BC-SAC [32], use a weighted mixture of behavior cloning loss and RL objectives in their loss function. SHAIL [23] formulates the imitation problem as a constrained RL problem with a constraint on the lower bound of policy likelihood on demonstrations. These approaches have shown that IL can help accelerate RL learning, especially under sparse rewards [20, 38], and vice versa [23]. As discussed in Sec. 1, similar objectives are also adopted in some RL fine-tuning algorithms [38, 35, 51, 43]. The pre-trained policy is not only used to initialize RL but also leveraged consistently over the fine-tuning stage as regularization. The introduction of RL has also been shown to effectively robustify behavior cloning [32] for challenging and safety-critical autonomous driving tasks. Conversely, IL regularization is used in offline RL approaches, such as TD3+BC [12] and CQL [27], to prevent overestimating the values on out-of-distribution actions and improve the robustness of offline RL policies. Different from these works where the additional objective is treated as regularization, our framework formulates the learning task as a new MDP and provides a principled way to interpret and design the combined objective.

One work in the literature that aligns with our motivation behind MCTS online customization is Deep Imitative Model [39], which aims to direct an imitative policy to arbitrary goals during online usage. It is formulated as a planning problem solved online to find the trajectory that maximizes its posterior likelihood conditioned on the goal. Compared to theirs, we formulate the additional requirements as rewards instead of goals, which are more flexible to specify. Also, residual Q-learning is a unified framework for both offline and online customization.

5   Experiments

We evaluate the proposed algorithms on four environments selected from different domains: *CartPole* and *Continuous Mountain Car* environments from the OpenAI gym classic control suite [5], and *Highway* and *Parking* from the *highway-env* environments [28]. In our experiments, we implemented our algorithms upon Stable-Baselines3 [37] and its imitation library [15]. In Sec. 5.1, we provide the configurations of our experiments, including the settings of policy customization tasks in different environments, baselines, and evaluation metrics. In Sec. 5.2, we present and analyze the experimental results of RL offline policy customization. In Sec. 5.3, we demonstrate an example of zero-shot online customization leveraging the residual maximum-entropy MCTS algorithm.
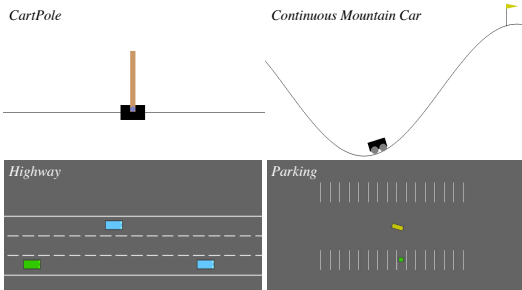

Figure 1: Four tested environments: *CartPole* and *Continuous Mountain Car* from the gym classic control suite, and *Highway* and *Parking* from the *highway-env* environments.

5.1   Environmental Setup

Environments. The detailed configurations of the environments can be found in Appendix B. We mainly describe the policy customization task settings here. In each environment, we design the basic and add-on rewards to illustrate a practical application scenario, where the basic reward corresponds to some basic task requirements and the add-on reward reflects customized specifications such as behavior preference or additional task objectives.

- **CartPole balancing:** In the cartpole environment, the basic task is to balance the pole as long as possible. During policy customization, we demand an additional task that requires the cart to stay at the center of the rack. To validate if the additional objective is satisfied, we

monitor the average absolute position error of the cart which is defined as $\bar{e}_x = \sum_{t=1}^{T} |x_t|/T$, where x_t is the coordinate of the cart with respect to the center of the rack.

- **Continuous mountain car:** In the mountain car environment, the basic task is to climb the mountain with the least energy consumption. During policy customization, we enforce an additional preference to avoid negative actions whenever possible. This corresponds to use cases where certain actions are less favorable. To monitor the change in action distribution, we count the number of negative actions executed during each episode, denoted as n_{neg}.

- **Highway navigation:** In the highway environment, the basic task is to navigate the vehicle safely and efficiently through the traffic. The vehicle can switch to arbitrary lanes to overtake the other vehicles. During policy customization, we enforce an additional preference to stay on the rightmost lane whenever possible. This reproduces driving habits that human drivers may have in real life. For example, drivers may prefer to drive on the rightmost lane to avoid missing the exits or when they want to drive the car slowly. We compute the average normalized lane index over an episode to validate if the desired behavior emerges, which is defined as $\bar{I}_{\text{lane}} = \sum_{t=1}^{T} I_{\text{lane},t}/2T$, where $I_{\text{lane},t} \in \{0, 1, 2\}$ is the lane index with the leftmost lane as the lane 0 and the rightmost lane as the lane 2.

- **Parking:** In the parking environment, the basic task is to park the car in the target parking slot. During policy customization, we add an additional requirement to avoid touching the boundaries of the parking slots during parking, which is considered good practice to avoid scratching other cars and to leave sufficient space for neighboring parking slots. We compute the non-violation rate to monitor if the requirement is met, which is defined as the percentage of episodes where the constraint is not violated, denoted as $\gamma_{\text{no-violation}}$.

Baselines. In our experiments, we mainly compare the performance of four categories of policies:

- **RL prior policy**: We train an RL policy optimizing the basic reward. It serves as the prior policy for policy customization. For environments with discrete action space (i.e., *CartPole* and *Highway*), we train the RL prior policy using soft Q-learning. For environments with continuous action space (i.e., *Continuous Mountain Car* and *Parking*), soft actor-critic is used. At test time, we also evaluate its performance on the overall task without customization, which serves as a baseline to show the effectiveness of the proposed algorithms.

- **IL prior policy**: We train another IL prior policy that imitates the RL prior with GAIL [21]. As discussed in Sec. 2, the proposed residual Q-learning algorithms are primarily motivated to customize imitative policy without knowledge of its inherent reward. The IL prior policy serves as a baseline for comparison with the residual-Q policy customized from the IL prior policy. In GAIL, we use soft Q-learning as the policy learning algorithm when the action space is discrete, and use soft actor-critic when the action space is continuous.

- **Residual-Q customized policies (Ours)**: In each environment, we train two residual-Q customized policies leveraging the RL and IL prior policies respectively. They are mainly compared against the corresponding prior policies to validate the effectiveness of policy customization. We apply residual soft Q-learning for environments with discrete action space and apply the residual soft actor-critic for environments with continuous action space. Besides, we evaluate the zero-shot customization ability of the proposed residual maximum-entropy MCTS algorithm in the *Highway* environment.

- **RL with total reward**: We also train an RL policy with the total reward (i.e., $r + \omega r_R$). Note that we do not intend to treat it as a baseline for fair comparison under the policy customization setting. We aim to compare the residual-Q policies with it to validate that the customized policies indeed solve the overall task $\hat{\mathcal{M}}$.

Metric. We are mainly interested in validating the performance of the proposed policy customization methods from two perspectives: 1) whether the policy is customized toward the add-on reward; 2) whether the customized policy still possesses the characteristics of the prior policy, in other words, maintains the same level of performance on the basic task. Hence, we use the average basic reward and the success rate of completing the basic task as metrics to evaluate the policies' performance on the basic task. Meanwhile, we use the average add-on reward and task-specific metrics introduced above as the metrics for evaluating the policies' performance with respect to the customized objective.

5.2 RL Offline Customization

The experimental results of RL offline customization are summarized in Table 1. In all the environments and with either RL or IL prior policies, the customized policies significantly outperform the prior policies on the add-on task objectives. Meanwhile, the customized policies maintain the same level of performance as the prior policies on the basic tasks. It validates that the proposed residual-Q algorithms can tailor the policies for customized requirements while maintaining their original characteristics. One interesting observation is that the policy customization improves the performance of the IL policy on the basic task in the *Parking* environment. One factor is that GAIL does not perform well in the first place. The *Parking* environment is indeed more challenging than the other environments given the continuous action space and, more importantly, the nature of the task—navigation in tight space under the non-holonomic constraint. Another aspect we think can explain this phenomenon is that the add-on reward *implicitly* guides the policy to improve its basic task performance. As shown in Appendix E, it is difficult for the imitative policy to stop the vehicle accurately at the target parking slot. Since GAIL merely learns to match the trajectory distributions under the expert and learned policies without the knowledge of the actual task objective, it is challenging for the imitative policy to precisely localize such a narrow parking space. In contrast, the add-on boundary violation constraint confines the vehicle trajectory within the parking space once the vehicle approaches the target parking slot. Hence, the policy is able to reach the goal point faster and receives higher returns even on the basic parking task.

In Table 1, we also compare the customized policies against the RL policies trained with the total reward function, named *RL Full Policy* in the table. In environments with discrete action spaces, the residual soft-Q learning policies behave very similarly to the RL full policies. Meanwhile, the residual soft actor-critic agents have different reward patterns compared to the RL full policies. We think the behavior difference is mainly due to the approximation errors in value and policy networks. The proposed residual-Q framework relies on the prior policy to acquire knowledge of the basic task. Thus, a suboptimal prior will affect the performance of the customized policy, especially on the basic task. It also explains the performance difference between the policies customized from RL and IL priors. Inspired by the observation mentioned above on IL policy customization, we think a practical solution to mitigate this issue is partially incorporating some basic task objectives into the add-on reward if available. It is reasonable for many practical use cases even with an imitative prior. While it is difficult to precisely describe the underlying reward function of human experts, we can comfortably assume that they follow some obvious commonsense objectives (e.g., collision avoidance). In this way, our method can be considered a more principled framework for the general synergy between IL and RL beyond policy customization where the reward functions are completely decomposed.

5.3 MCTS Online Customization

We choose the *Highway* environment to demonstrate the zero-shot online customization ability of residual maximum-entropy MCTS. We implement residual maximum-entropy MCTS based on the MCTS planner provided by the *highway-env* environment. The planner leverages the ground-truth dynamics model of the environment for roll-out simulation. The results are summarized in Table 1. MCTS online customization is able to achieve similar performance as the RL offline counterpart.

6   Discussion and Limitations

In our residual Q-learning framework, we formulate policy customization as an MDP whose reward function balances the utilities of the imitative and add-on tasks. It provides the theoretical foundation to determine the synergy between the IL and RL objectives in a principled manner. As discussed in Sec. 1 and Sec. 4, different approaches have been proposed in the RL fine-tuning literature to combine IL and RL objectives. We investigate two representative methods under the context of policy customization. The detailed analysis can be found in Appendix D.

One common approach is regularizing the divergence between the trained RL and prior policies during policy updates. It has been widely used in offline RL algorithms to address the distributional shift issue [26, 46], and Advantage Weighted Actor-Critic (AWAC) [35] demonstrates the benefits of such policy regularization in stabilizing and accelerating RL fine-tuning. Note that when using KL-divergence to measure the distance between policies, the regularized optimal policy becomes a maximum-entropy policy with the advantage weighted by the prior policy [35]. We can reduce

Table 1: Experimental Results of Residual-Q Policy Customization

Env.	Policy	Basic Task		Add-on Task	
		Succ. Rate	Basic Reward	$\bar{e}_x$	Add-on Reward
CartPole	RL Prior Policy	100%	427.21 ± 2.25	0.15 ± 0.11	-31.11 ± 22.82
	RL Customized	100%	425.73 ± 2.74	$\mathbf{0.05 \pm 0.05}$	$\mathbf{-11.18 \pm 10.12}$
	IL Prior Policy	100%	422.17 ± 4.19	0.19 ± 0.12	-38.57 ± 24.26
	IL Customized	100%	423.37 ± 4.28	$\mathbf{0.07 \pm 0.06}$	$\mathbf{-14.12 \pm 12.09}$
	RL Full Policy	100%	425.27 ± 3.10	0.05 ± 0.04	-10.72 ± 7.43
Env.	**Policy**	**Succ. Rate**	**Basic Reward**	n_{neg}	**Add-on Reward**
Cont. Mt. Car	RL Prior Policy	100%	95.78 ± 0.64	42.26 ± 1.29	-4.23 ± 0.13
	RL Customized	100%	95.61 ± 0.43	$\mathbf{37.90 \pm 0.75}$	$\mathbf{-3.79 \pm 0.07}$
	IL Prior Policy	100%	92.84 ± 2.35	64.40 ± 21.03	-6.44 ± 2.10
	IL Customized	100%	94.41 ± 0.06	$\mathbf{41.08 \pm 1.01}$	$\mathbf{-4.11 \pm 0.10}$
	RL Full Policy	100%	95.14 ± 0.21	27.86 ± 3.87	-2.79 ± 0.39
Env.	**Policy**	**Succ. Rate**	**Basic Reward**	$\bar{I}_{\mathrm{lane}}$	**Add-on Reward**
Highway	RL Prior Policy	97.22%	44.67 ± 2.43	0.46 ± 0.28	9.17 ± 5.54
	RL Customized	97.19%	42.23 ± 1.55	$\mathbf{0.97 \pm 0.05}$	$\mathbf{19.42 \pm 0.99}$
	RL ME-MCTS	98.00%	42.71 ± 1.65	$\mathbf{0.96 \pm 0.07}$	$\mathbf{19.19 \pm 1.38}$
	IL Prior Policy	96.04%	40.03 ± 0.02	0.50 ± 0.40	10.06 ± 8.07
	IL Customized	93.00%	40.36 ± 0.34	$\mathbf{0.99 \pm 0.01}$	$\mathbf{19.88 \pm 0.24}$
	IL ME-MCTS	97.00%	40.68 ± 0.46	$\mathbf{0.99 \pm 0.01}$	$\mathbf{19.85 \pm 0.30}$
	RL Full Policy	96.35%	41.98 ± 1.53	0.97 ± 0.01	19.49 ± 1.24
Env	**Policy**	**Succ. Rate**	**Basic Reward**	$\gamma_{\mathrm{no\text{-}violation}}$	**Add-on Reward**
Parking	RL Prior Policy	99.09%	-7.08 ± 2.65	57.61%	-1.55 ± 3.71
	RL Customized	98.73%	-7.60 ± 3.07	$\mathbf{96.09\%}$	$\mathbf{-0.03 \pm 0.20}$
	IL Prior Policy	70.91%	-12.37 ± 9.05	11.27%	-4.94 ± 5.52
	IL Customized	$\mathbf{83.59\%}$	$\mathbf{-7.93 \pm 4.84}$	$\mathbf{66.07\%}$	$\mathbf{-0.32 \pm 0.73}$
	RL Full Policy	99.34%	-6.81 ± 2.39	74.24%	-0.38 ± 0.72

The evaluation results are computed over 4000 episodes for model-free policies and 100 episodes for maximum-entropy MCTS (ME-MCTS). The statistics with $\pm$ are in the format of mean $\pm$ std.

our residual Q-learning policy to a form equivalent to the AWAC policy by replacing the residual Q-function in Eqn. (6) with the soft Q-function corresponding to the add-on reward. It is equivalent to heuristically estimating the optimal Q-function as the sum of the optimal Q-functions corresponding to the basic and add-on rewards. Hence, the synthesized policy is not the optimal solution to the target MDP but a greedy approximation [44, 24]. We found that this greedy customized policy performed much worse than residual Q-learning in the most challenging *Parking* environment.

Alternatively, some works [46, 51] have explored adding the policy divergence to the reward function. It has been shown to outperform policy regularization in offline RL setting [46]. However, we found that it made the learning task much more difficult in our policy customization problem—soft actor-critic, for example, failed to find a customized policy that at least perform well on the basic parking task. We believe that one crucial factor contributing to the failure of divergence-based regularization is that the basic and add-on rewards are mostly *orthogonal* in our setting. The add-on reward only encodes additional task requirements; thus, we heavily rely on the prior policy to embed the desirable behavior on the basic task. Merely regularizing the KL-divergence is therefore insufficient—being close to the prior policy in terms of probabilistic distance does not necessarily imply behavior that accomplishes the basic task objective. Besides, incorporating the policy likelihood into the reward function makes it more challenging to update the Q-function in practice. For example, in Appendix D.3, we show that our residual Q-learning algorithm is theoretically equivalent to directly

solving an MDP whose reward function is a weighted sum of the add-on reward and the log-likelihood of the prior policy. However, similar to the case of divergence-augmented reward, soft actor-critic failed to find a solution in *Parking* environment.

Limitations. As discussed in Sec. 5.2, one limitation of our current algorithms is that the performance of policy customization is *bottlenecked* by the imitative prior policy. Policy customization not only requires it to perform well on the trajectory distribution under the imitative prior policy itself but, more importantly, we need the imitative policy to accurately indicate desired behavior with respect to the basic task objective on the trajectory distribution under the *customized* policy. Hence, without careful design, our algorithms would suffer from the distributional shift issue. While, in practice, this issue can be mitigated by partially incorporating some basic task objectives into the add-on reward if available, a more principled and fundamental solution is to obtain a diverse imitative prior that is best suited for customization. Generative models, such as normalizing flows [42] and, more recently, diffusion models [22, 6, 19], have been shown promising in constructing such a diverse behavior prior. We will investigate their applications in the policy customization problem in future work. Also, we only demonstrate MCTS online customization with an ideal perfect dynamics model in our current experiments. In future work, we will combine residual maximum-entropy MCTS with offline learned predictive models to achieve zero-shot online customization in more practical scenarios.

References

[1] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[2] R. Bhattacharyya, B. Wulfe, D. J. Phillips, A. Kuefler, J. Morton, R. Senanayake, and M. J. Kochenderfer. Modeling human driving behavior through generative adversarial imitation learning. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[3] R. P. Bhattacharyya, D. J. Phillips, C. Liu, J. K. Gupta, K. Driggs-Campbell, and M. J. Kochenderfer. Simulating emergent properties of human driving behavior using multi-agent reward augmented imitation learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 789–795. IEEE, 2019.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[7] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[8] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[9] P. Englert, N. A. Vien, and M. Toussaint. Inverse kkt: Learning cost functions of manipulation tasks from demonstrations. *The International Journal of Robotics Research*, 36(13-14):1474–1488, 2017.

[10] J. L. V. Espinoza, A. Liniger, W. Schwarting, D. Rus, and L. Van Gool. Deep interactive motion prediction and planning: Playing games with motion prediction models. In *Learning for Dynamics and Control Conference*, pages 1006–1019. PMLR, 2022.

[11] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. 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*, pages 9710–9719, 2021.

[12] S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[13] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[14] D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon. Iq-learn: Inverse soft-q learning for imitation. In *Advances in Neural Information Processing Systems*, volume 34, pages 4028–4039, 2021.

[15] A. Gleave, M. Taufeeque, J. Rocamonde, E. Jenner, S. H. Wang, S. Toyer, M. Ernestus, N. Belrose, S. Emmons, and S. Russell. imitation: Clean imitation learning implementations. *arXiv preprint arXiv:2211.11972*, 2022.

[16] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[18] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[19] P. Hansen-Estruch, I. Kostrikov, M. Janner, J. G. Kuba, and S. Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[20] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[21] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, volume 29, 2016.

[22] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[23] K. Judah, A. Fern, P. Tadepalli, and R. Goetschalckx. Imitation learning with demonstrations and shaping rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[24] Z. Juozapaitis, A. Koul, A. Fern, M. Erwig, and F. Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[25] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[26] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

[27] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[28] E. Leurent. An environment for autonomous driving decision-making. *GitHub repository*, 2018.

[29] C. Li, T. Trinh, L. Wang, C. Liu, M. Tomizuka, and W. Zhan. Efficient game-theoretic planning with prediction heuristic for socially-compliant autonomous driving. *IEEE Robotics and Automation Letters*, 7(4):10248–10255, 2022.

[30] J. Li, C. Tang, M. Tomizuka, and W. Zhan. Dealing with the unknown: Pessimistic offline reinforcement learning. In *Conference on Robot Learning*, pages 1455–1464. PMLR, 2022.

[31] Y. Li, J. Song, and S. Ermon. Infogail: Interpretable imitation learning from visual demonstrations. *Advances in Neural Information Processing Systems*, 30, 2017.

[32] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, B. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, et al. Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios. *arXiv preprint arXiv:2212.11419*, 2022.

[33] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. Learning to generalize across long-horizon tasks from human demonstrations. *arXiv preprint arXiv:2003.06085*, 2020.

[34] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, pages 1678–1690. PMLR, 2022.

[35] A. Nair, A. Gupta, M. Dalal, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[36] A. Raffin. Rl baselines3 zoo. *GitHub repository*, 2020.

[37] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[38] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[39] N. Rhinehart, R. McAllister, and S. Levine. Deep imitative models for flexible inference, planning, and control. In *International Conference on Learning Representations*, 2020.

[40] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *Conference on Robot Learning*, pages 718–728. PMLR, 2022.

[41] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[42] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[43] I. Uchendu, T. Xiao, Y. Lu, B. Zhu, M. Yan, J. Simon, M. Bennice, C. K. Fu, C. Ma, J. Jiao, S. Levine, and K. Hausman. Jump-start reinforcement learning. In *NeurIPS 2021 Robot Learning Workshop, RSS 2022 Scaling Robot Learning Workshop*, 2022.

[44] H. Van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[45] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023.

[46] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[47] C. Xiao, R. Huang, J. Mei, D. Schuurmans, and M. Müller. Maximum entropy monte-carlo planning. *Advances in Neural Information Processing Systems*, 32, 2019.

[48] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34:25476–25488, 2021.

[49] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kummerle, H. Konigshof, C. Stiller, A. de La Fortelle, et al. Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps. *arXiv preprint arXiv:1910.03088*, 2019.

[50] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[51] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A Policy Evaluation for Residual Soft Actor-Critic

In this section, we derive the policy evaluation step of the residual soft actor-critic algorithm introduced in Sec. 3. In the soft actor-critic algorithm, the policy evaluation step aims to iteratively compute the soft Q-value of a policy $\hat{\pi}$, which relies on repeatedly applying a modified soft Bellman backup operator given by:

$$\hat{Q}_{t+1}(\boldsymbol{s}, \boldsymbol{a}) = r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s}' \sim p(\boldsymbol{s}'|\boldsymbol{a})} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}} \left[\hat{Q}_t(\boldsymbol{s}', \boldsymbol{a}') - \hat{\alpha} \log \hat{\pi}(\boldsymbol{a}'|\boldsymbol{s}') \right] \right]. \quad (15)$$

We can then derive an update rule for the residual Q-function from the modified soft Bellman backup operator following a similar procedure as in Eqn. (5):

$$\begin{aligned}
& Q_{R,t+1}(\boldsymbol{s}, \boldsymbol{a}) \\
&= r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s}'} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}} \left[\hat{Q}_t(\boldsymbol{s}', \boldsymbol{a}') - \hat{\alpha} \log \hat{\pi}(\boldsymbol{a}'|\boldsymbol{s}') \right] \right] - \omega Q^*(\boldsymbol{s}, \boldsymbol{a}), \quad (16a) \\
&= r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega Q^*(\boldsymbol{s}, \boldsymbol{a}) - \omega' \gamma \mathbb{E}_{\boldsymbol{s}'} \log Z_{s'} - \omega Q^*(\boldsymbol{s}, \boldsymbol{a}) \\
& \quad + \gamma \mathbb{E}_{\boldsymbol{s}'} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}} \left[Q_{R,t}(\boldsymbol{s}', \boldsymbol{a}') + \omega' \log \pi(\boldsymbol{a}'|\boldsymbol{s}') + \omega' \log Z_{s'} - \hat{\alpha} \log \hat{\pi}(\boldsymbol{a}'|\boldsymbol{s}') \right] \right], \quad (16b) \\
&= r_R(\boldsymbol{s}, \boldsymbol{a}) - \omega' \gamma \mathbb{E}_{\boldsymbol{s}'} \log Z_{s'} + \omega' \gamma \mathbb{E}_{\boldsymbol{s}'} \log Z_{s'} \\
& \quad + \gamma \mathbb{E}_{\boldsymbol{s}'} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}} \left[Q_{R,t}(\boldsymbol{s}', \boldsymbol{a}') + \omega' \log \pi(\boldsymbol{a}'|\boldsymbol{s}') - \hat{\alpha} \log \hat{\pi}(\boldsymbol{a}'|\boldsymbol{s}') \right] \right], \quad (16c) \\
&= r_R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s}'} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}} \left[Q_{R,t}(\boldsymbol{s}', \boldsymbol{a}') + \omega' \log \pi(\boldsymbol{a}'|\boldsymbol{s}') - \hat{\alpha} \log \hat{\pi}(\boldsymbol{a}'|\boldsymbol{s}') \right] \right]. \quad (16d)
\end{aligned}$$

B Environment Configuration

In this section, we introduce the detailed configurations of the selected environments, including the definitions of the state and action spaces, observations, and reward functions.

CartPole balancing. In the cartpole environment, a pole is attached to a cart by an unactuated joint. The cart is moving along a track. The states are defined as the coordinate of the cart along the track, x, the angle of the pole, δ, the velocity of the cart, v, and the angular velocity of the pole, $\dot{\delta}$. The action is a binary variable indicating the direction of the force exerted on the cart, i.e., $\mathcal{A} = \{\text{"left", "right"}\}$. The goal of the basic task is to balance the pole by exerting forces on the cart, which is considered to be successful if the pole does not fall down for 500 steps. The basic reward function is a sum of two components, which are:

$$\text{Survival Reward}: \quad r_{\text{survival}}(\boldsymbol{s}, \boldsymbol{a}) = 1, \quad (17)$$

$$\text{Balancing Reward}: \quad r_{\text{balance}}(\boldsymbol{s}, \boldsymbol{a}) = -\frac{10|\delta|}{0.2095}. \quad (18)$$

During policy customization, we demand an additional task that requires the cart to stay at the center of the rack. The corresponding add-on reward is defined as $r_R(\boldsymbol{s}, \boldsymbol{a}) = -|x|/2.4$, which penalizes the deviation from the center of the rack at each time step.

Continuous mountain car. In the mountain car environment, the car is placed at the bottom of a sinusoidal valley with a randomized initial position. The states are defined as the horizontal coordinate of the car, x, and the velocity of the car, v. The action is the directional force f applied to the car. The goal of the basic task is to accelerate the car to reach the goal state on top of the right hill with the least energy consumption. It is considered successful if the car reaches the goal within 999 steps. The basic reward function is designed as a sum of the following two components:

$$\text{Goal Reward}: \quad r_{\text{goal}}(\boldsymbol{s}, \boldsymbol{a}) = 100 \times \mathbb{1}(\boldsymbol{s} = \boldsymbol{s_g}), \quad (19)$$

$$\text{Energy Cost}: \quad r_{\text{energy}}(\boldsymbol{s}, \boldsymbol{a}) = -0.1f^2. \quad (20)$$

During policy customization, we enforce an additional preference to avoid negative actions whenever possible. The add-on reward is defined as $r_R(\boldsymbol{s}, \boldsymbol{a}) = -0.5 \times \mathbb{1}(f < 0)$ to penalize negative force.

Highway navigation. In the highway environment, we navigate the vehicle on a three-lane highway around other vehicles. The state vector of the i-th vehicle is defined as $\boldsymbol{s}_i = \left[x_i, y_i, v_{x,i}, v_{y,i}, \theta_i, \dot{\theta}_i \right]^{\mathsf{T}}$, where x_i, y_i are the xy-coordinates, $v_{x,i}, v_{y,i}$ are the velocities along the x- and y-axes, θ_i is the yaw

angle, and $\dot{\theta}$ is the yaw rate. The ego vehicle controlled by the policy is indexed with $i = 0$. The observation collects the normalized states of all the vehicles. The discrete action space consists of five meta-actions that govern the corresponding built-in controllers provided as part of the environment. The meta-actions define different high-level behaviors in the highway environment, referred to as "switch left", "switch right", "faster", "idle", and "slower" [28]. The goal of the basic task is to drive the vehicle through the traffic safely and efficiently. The task is considered successful if the car is driven without collision over 40 steps. The basic reward function consists of three components:

$$\text{Survival Reward}: \quad r_{\text{survival}}(\boldsymbol{s}, \boldsymbol{a}) = 1, \tag{21}$$

$$\text{Velocity Reward}: \quad r_{\text{velocity}}(\boldsymbol{s}, \boldsymbol{a}) = 0.4 v_{\text{norm}}, \tag{22}$$

$$\text{Collision Cost}: \quad r_{\text{collision}}(\boldsymbol{s}, \boldsymbol{a}) = -0.5 \times \mathbb{1}(\text{isCollision}(\boldsymbol{s}) = 1), \tag{23}$$

where v_{norm} is defined as $(\sqrt{v_{x,0}^2 + v_{y,0}^2} - 20)/10$ and clipped into $[0, 1]$, and isCollision is a built-in function to determine if the ego collides with the other vehicles. During policy customization, we enforce an additional preference to stay on the rightmost lane whenever possible, which is formulated as an add-on reward function defined as:

$$r_R(\boldsymbol{s}, \boldsymbol{a}) = 0.5 \frac{I_{\text{lane}}(\boldsymbol{s})}{\text{num of lanes} - 1}, \tag{24}$$

where the function I_{lane} gives the index of the lane where the car is driving, with the leftmost lane indexed with 0.

Parking. In the parking environment, we control a vehicle in the parking lot. The states of the vehicle are defined the same as in the highway environment. The observation consists of the current state, $\boldsymbol{s}$, and the target parking state, $\boldsymbol{s}_g$. The actions are the longitudinal acceleration command, a, and the steering angle command, δ. The goal of the basic task is to park the vehicle at the target parking space within a minimal number of time steps, the task is considered successful if the error between the vehicle state and the target parking state becomes smaller than a threshold value within 100 steps. Hence, the basic reward is simply defined as $r(\boldsymbol{s}, \boldsymbol{a}) = -\boldsymbol{w}^T \|\boldsymbol{s} - \boldsymbol{s_g}\|$. During policy customization, we add an additional requirement to avoid touching the boundaries of the parking slots during parking, the add-on reward is defined as $r_R(\boldsymbol{s}, \boldsymbol{a}) = -\mathbb{1}(\text{Violation}(\boldsymbol{s}) = 1)$, where Violation is a function detecting whether the vehicle violates the enforced boundary constraint.

C   Implementation Details and Hyperparameters

In this section, we introduce the detailed implementation of the proposed algorithms and the hyperparameters we used in our experiments for each environment. We implemented the proposed algorithms and all the baseline methods upon Stable-Baselines3 [37] and its imitation library [15]. In addition, we implemented the residual maximum-entropy MCTS based on the MCTS planner provided by the *highway-env* environment. All the experiments were conducted on Ubuntu 16.04 with Intel Core i7-7700 CPU @ 3.60GHz × 8, GeForce GTX 1070/PCIe/SSE2, and 32 GB RAM.

C.1   Algorithm Implementation

Residual Soft Q-Learning was implemented upon the standard deep Q-network (DQN) from Stable-Baselines3 [37]. In specific, we substituted the target Q-value with the target residual Q-value defined in Eqn. (8) when computing the TD error for the loss function.

Residual Soft Actor-Critic was implemented upon the standard soft actor-critic (SAC) from Stable-Baselines3 [37]. Similar to the case of residual soft Q-learning, we substituted the target Q-value with the target residual Q-value defined in Eqn. (9) when computing the TD error for the critic loss. In addition, we added the log-likelihood of the prior policy into the actor loss as in Eqn. (10).

Residual Maximum-Entropy MCTS was implemented upon the MCTS planner provided by the *highway-env* environment. We first adapted it to the maximum-entropy MCTS proposed in [47] and then implemented the residual maximum-entropy MCTS upon it. The pseudo-code of the proposed algorithm is presented in the Algorithm 1.

C.2   Hyperparameters

RL prior policy. We adopted the hyperparameters from RL Baselines3 Zoo [36] for the *Cart Pole*, *Continuous Mountain Car* and *Highway* environments. For the *Parking* environment, we adopted

the hyperparameter provided in Stable-Baselines3's document [37]. We further adjusted the learning rate and increase the number of training steps for better performance. The final learning rate and the number of training steps for each environment are summarized in Table 2.

IL prior policy. The IL prior policies were trained by imitating the RL prior policies with GAIL. The same RL algorithms used to train the RL experts were used as the policy learning algorithms in GAIL. Thus, we adopted the same hyperparameters for the policy learners. For each environment, we further tuned the following hyperparameters specified in the Stable-Baselines3 imitation library [15]:

- `expert_min_episodes`: The minimum number of episodes of expert demonstration.
- `demo_batch_size`: The number of samples contained in each batch of expert data.
- `gen_replay_buffer_capacity`: The capacity of the generator replay buffer, i.e., the maximum number of state-action pairs sampled from the generator that can be stored.
- `n_disc_updates_per_round`: The number of discriminator update steps after each iteration of generator update.

The final values of these hyperparameters for each environment are summarized in Table 3.

Residual Q-learning policies. The model-free residual Q-learning policies were trained with the same hyperparameters of their corresponding RL prior policy. The additional hyperparameters are summarized in Table 4. For the *Parking* environment, we scaled ω' by 10 times at the first 10^5 training steps, so that the residual Q-learning policy was trained to copy the prior policy. We found that it could accelerate the exploration at the early stage and stabilize the training process. For residual maximum-entropy MCTS, the hyperparameters include ω', maximum iterations $Iter_{\max}$, planning horizon H, and the exploration coefficient ϵ. The final values of hyperparameters we used in the *Parking* experiment are summarized in Table 5.

Table 2: Learning Rate and Training Steps of RL Policies

Hyperparameter	Cart Pole	Mountain Car Continuous	Highway	Parking
Learning Rate	2.3×10^{-3}	3×10^{-4}	10^{-4}	10^{-3}
Number of Training Steps	10^5	10^5	5×10^5	8×10^5

Table 3: Hyperparameters of GAIL Imitated Policies

Hyperparameter	Cart Pole	Mountain Car Continuous	Highway	Parking
`expert_min_episodes`	10^3	10^4	10^3	10^4
`demo_batch_size`	1024	1024	1024	1024
`gen_replay_buffer_capacity`	2048	2048	2048	2048
`n_disc_updates_per_round`	4	4	4	4

Table 4: Hyperparameters of Residual Q Policies

Hyperparameter	Prior Policy	Cart Pole	Mountain Car Continuous	Highway	Parking
α	RL	1	0.1	1	0.0097
	IL	1	0.1	1	auto
$\hat{\alpha}$	RL	1	0.1	1	auto
	IL	1	0.1	1	0.0611
ω'	RL	1	0.1	1	0.0097
	IL	1	0.1	1	0.0611

Table 5: Hyperparameters of Residual Maximum-Entropy MCTS

Hyperparameter	Prior Policy	Value
ω'	RL	1
	IL	1
$Iter_{\max}$	RL	150
	IL	150
H	RL	6
	IL	6
ϵ	RL	0.1
	IL	1

D   Ablation Study

In this section, we investigate three alternative algorithm designs for policy customization to illustrate the advantages of the proposed residual Q-learning framework. In Sec. D.1 and D.2, we study two alternatives where the KL-divergence between the customized and prior policies is used to incorporate the behavior prior into the customized policy. In Sec. D.3, we study a theoretically equivalent form of residual Q-learning that can be directly solved with off-the-shelf RL algorithms.

D.1   Greedy Reward Decomposition

As discussed in Sec. 6, one common method to infuse IL objective into RL is directly regularizing the divergence between the trained RL and prior policies during policy updates. As shown in AWAC [35], when using KL-divergence to measure the distance between policies, the regularized optimal policy becomes a maximum-entropy policy with the advantage weighted by the prior policy. If we directly adapt it to solve the policy customization problem, the customized policy is essentially defined as the solution to the following optimization problem at each policy update step:

$$\tilde{\pi}_t = \arg\max_{\tilde{\pi} \in \Pi} \mathbb{E}_{\boldsymbol{a} \sim \tilde{\pi}(\cdot|\boldsymbol{s})} \left[\tilde{Q}_{R,t}(\boldsymbol{s}, \boldsymbol{a}) - \hat{\alpha} \log \tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}) \right] - \lambda \mathcal{D}_{\mathrm{KL}}(\tilde{\pi}(\cdot|\boldsymbol{s}) \| \pi(\cdot|\boldsymbol{s})), \qquad (25)$$

where $\tilde{Q}_{R,t}$ is the soft Q-function of the MDP with only the add-on reward $r_R(\boldsymbol{s}, \boldsymbol{a})$, which is defined iteratively with the following update rule:

$$\tilde{Q}_{R,t+1}(\boldsymbol{s}, \boldsymbol{a}) = r_R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s}' \sim p(\boldsymbol{s}'|\boldsymbol{a})} \left[\mathbb{E}_{\boldsymbol{a}' \sim \hat{\pi}_t} \left[\tilde{Q}_{R,t}(\boldsymbol{s}', \boldsymbol{a}') - \hat{\alpha} \log \tilde{\pi}_t(\boldsymbol{a}'|\boldsymbol{s}') \right] \right]. \qquad (26)$$

The closed-form solution to the optimization problem is given as:

$$\tilde{\pi}_t(\boldsymbol{a}|\boldsymbol{s}) \propto \exp \left(\frac{1}{\hat{\alpha} + \lambda} \left(\tilde{Q}_{R,t}(\boldsymbol{s}, \boldsymbol{a}) + \lambda \log \pi(\boldsymbol{a}|\boldsymbol{s}) \right) \right). \qquad (27)$$

We can reduce our residual Q-learning policy to the same form by simply replacing the residual Q-function in Eqn. (6) with the soft Q-function $\tilde{Q}_{R,t}$. The resulting policy solves the optimization problem in Eqn. (25) but with an entropy weight of $\hat{\alpha} - \lambda$ in the objective function. If we consider it for the policy customization problem, it is essentially equivalent to heuristically estimating the optimal Q-function as the sum of the optimal Q-functions corresponding to the basic and add-on rewards. Hence, the synthesized policy is not the optimal solution to the target MDP but a greedy approximation [44, 24]. We compare residual Q-learning with this greedily customized policy on the *Continuous Mountain Car* and *Parking* environments. As shown in Table 6, residual Q-learning outperforms the greedy policies on both the basic and add-on objectives, especially on the challenging *Parking* environment. Compared to the prior policy, greedy customization compromises the policy's success rate on the basic parking task to customize them towards satisfying the add-on objective. However, the greedy policy still has a significantly lower non-violation rate compared to the residual Q-learning policy. It validates that our framework is a more principled method to jointly optimize the IL and RL objectives for policy customization than divergence-based regularization.

Table 6: Greedy Reward Decomposition Experimental Results

Env.	Policy	Basic Task		Add-on Task	
		Succ. Rate	Basic Reward	n_{neg}	Add-on Reward
Cont. Mt. Car	RL Greedy	100%	95.59 ± 0.51	39.47 ± 0.77	-3.95 ± 0.08
	RL Residual-Q	100%	95.61 ± 0.43	$\mathbf{37.90 \pm 0.75}$	$\mathbf{-3.79 \pm 0.07}$
	IL Greedy	100%	93.96 ± 0.08	46.10 ± 1.53	-4.61 ± 0.15
	IL Residual-Q	100%	94.41 ± 0.06	$\mathbf{41.08 \pm 1.01}$	$\mathbf{-4.11 \pm 0.10}$
Env	Policy	Succ. Rate	Basic Reward	$\gamma_{\text{no-violation}}$	Add-on Reward
Parking	RL Greedy	84.07%	-7.38 ± 2.90	70.39%	-0.26 ± 0.64
	RL Residual-Q	$\mathbf{98.73\%}$	-7.60 ± 3.07	$\mathbf{96.09\%}$	$\mathbf{-0.03 \pm 0.20}$
	IL Greedy	69.06%	-7.63 ± 4.96	48.79%	-0.49 ± 0.87
	IL Residual-Q	$\mathbf{83.59\%}$	-7.93 ± 4.84	$\mathbf{66.07\%}$	$\mathbf{-0.32 \pm 0.73}$

D.2 Augmenting Reward with Policy KL-Divergence

Alternatively, some works [46, 51] have explored adding the policy divergence to the reward function as an IL objective. In this subsection, we investigate this line of approaches for policy customization. We formulate the reward function of the target MDP to solve as the sum of the add-on reward and a policy divergence regularization term. In particular, we follow [51] to augment the add-on reward with a penalty whose expectation regularizes the KL-divergence between the customized and prior policies. Formally, the reward function is defined as:

$$\check{r}_t(\boldsymbol{s}, \boldsymbol{a}) = r_R(\boldsymbol{s}, \boldsymbol{a}) - \beta \log \frac{\check{\pi}_t(\boldsymbol{a}|\boldsymbol{s})}{\pi(\boldsymbol{a}|\boldsymbol{s})}, \tag{28}$$

where $\check{\pi}_t$ denotes the customized policy after the tth iteration of policy update, and β is a hyperparameter to balance the add-on reward and the policy regularization term. We can then apply standard RL algorithms to customize the policy toward this reward function. We evaluated this method in the *Parking* environment with soft actor-critic as the learning algorithm. As shown in Figure 2, It failed to find a customized policy that at least perform well on the basic parking task—The success rate was persistently close to zero during the learning procedure. We believe that one crucial factor contributing to its failure is that the basic and add-on rewards are mostly *orthogonal* in our setting. The add-on reward only encodes additional task requirements; thus, we heavily rely on the prior policy to embed the desirable behavior on the basic task. Merely regularizing the policy divergence is therefore insufficient—being close to the prior policy in terms of probabilistic distance does not necessarily imply behavior that accomplishes the basic task objective.

D.3 Augmenting Reward with Policy Log Likelihood

The last alternative algorithm we investigate is a theoretically equivalent form of residual Q-learning. Observe that if we add $\omega' \log \pi(\boldsymbol{s}, \boldsymbol{a})$ to both sides of Eqn. (5), we obtain:

$$Q_{R,t+1}^{\text{aug}}(\boldsymbol{s}, \boldsymbol{a}) = r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega' \log \pi(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{s}'} \left[\hat{\alpha} \log \int_{\mathcal{A}} \exp \left(\frac{1}{\hat{\alpha}} Q_{R,t}^{\text{aug}}(\boldsymbol{s}', \boldsymbol{a}') \right) d\boldsymbol{a}' \right], \tag{29}$$

where $Q_{R,t}^{\text{aug}}(\boldsymbol{s}, \boldsymbol{a}) = Q_{R,t}(\boldsymbol{s}, \boldsymbol{a}) + \omega' \log \pi(\boldsymbol{s}, \boldsymbol{a})$. It is then straightforward to see that $Q_{R,t}^{\text{aug}}$ is the soft Q-function corresponding to the reward function $r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega' \log \pi(\boldsymbol{s}, \boldsymbol{a})$, which means that we can find a policy equivalent to the residual Q-learning one, simply through solving the MDP $\mathcal{M}^{\text{aug}} = (\mathcal{S}, \mathcal{A}, r_R + \omega' \log \pi, p)$ with off-the-shell RL algorithms. We test this equivalent algorithm in the *Parking* environment. As shown in Figure 2, similar to the case with divergence-augmented reward, soft actor-critic failed to find a customized policy that can complete the basic parking task. We think it is because estimating $Q_{R,t}^{\text{aug}}$ is inherently difficult since the complex policy log-likelihood function is embedded into the reward. In contrast, when estimating $Q_{R,t}^{\text{aug}}$, our residual Q-learning framework fully leverages our prior knowledge on $\log \pi$ and only estimates the residual part during policy customization. Thus, residual Q-learning can more efficiently explore and optimize the policy.
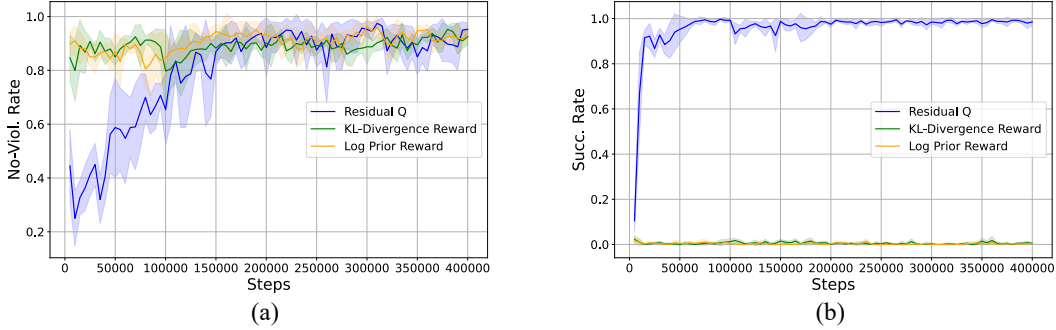
(a)

(b)

Figure 2: Learning curves in the *Parking* environment for different algorithms, including residual soft actor-critic, soft actor-critic with divergence-augmented reward (Appendix D.2), and soft actor-critic with likelihood-augmented reward (Appendix D.3). We plot the curves of the non-violation rate and success rate over the number of episodic steps in subplots (a) and (b) respectively. The error bands indicate standard deviations computed over four trials with different random seeds.
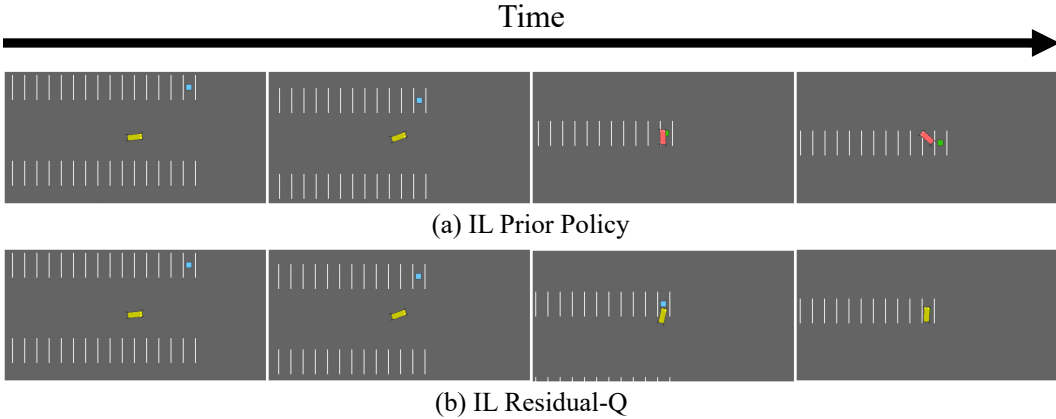
Time

(a) IL Prior Policy

(b) IL Residual-Q

Figure 3: Representative examples from the *Parking* environment comparing the results of executing the IL prior and the residual Q-learning policy customized from the IL prior.

E   Visualization

In this section, we visualize some representative examples from the *Parking* environment. As shown in Figure 3, the IL policy fails to stop the car within the target parking slot, while the customized policy is able to do so with the guidance of the boundary violation constraint. As shown in Figure 4, compared to the RL prior policy, the proposed residual Q-learning policy finds a very different and violation-free parking route, whereas the greedily customized policy gets stuck and fails to reach the target parking slot. Also, it is worth noting that the customized policy behaves differently from the RL full policy. As discussed in Sec. 5.2, we think the behavior difference is mainly due to the approximation errors in value and policy networks, which results in an imperfect prior policy.
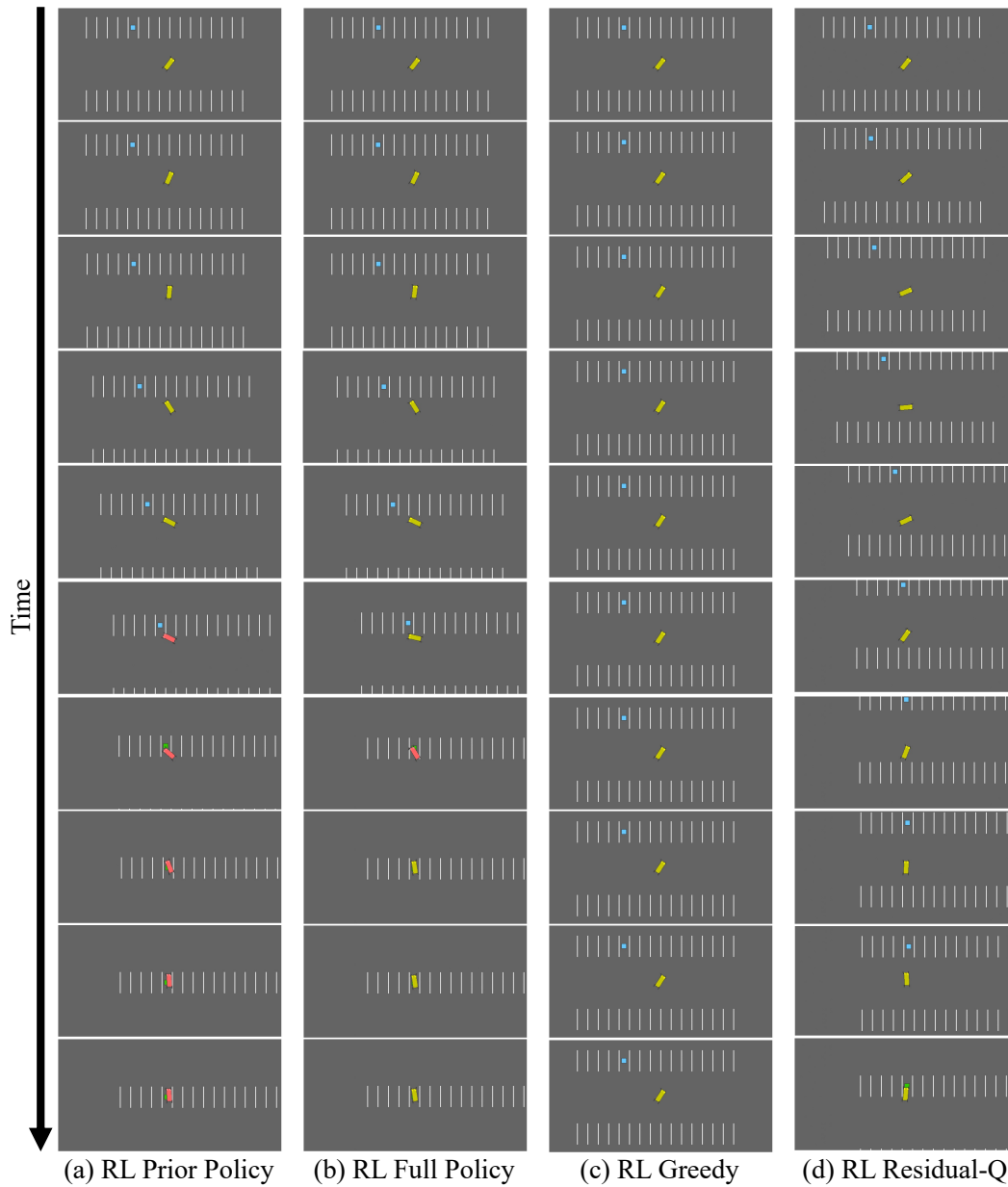
Figure 4: Representative examples from the *Parking* environment comparing the results of executing RL prior, RL full policy, and policies customized from RL prior with different approaches.

Algorithm 1 Residual Maximum-Entropy MCTS

Input: Current state s_0; Prior policy π; Hyperparameter ω'; Maximum iterations $Iter_{\max}$; Planning horizon H; Exploration coefficient ϵ.

Output: Root node n_0 of the built tree.

1: Initialize the root node $n_0(s_0)$.
2: Initialize iteration counter $Iter = 0$.
3: **while** $Iter < Iter_{\max}$ **do**
4:      $n_{T-1} \leftarrow$ SELECTION(n_0)
5:      $n_T \leftarrow$ EXPANSION(n_{T-1})
6:      $R \leftarrow$ SIMULATION&EVALUATION(n_T)
7:      BACK-PROPAGATION(n_T, R)
8: **end while**
9: **return** n_0

10: **function** SELECTION(n_0)
11:      $n_{T-1} \leftarrow n_0$
12:      **while** n_{T-1} is non-terminal **do**
13:          **if** n_{T-1} is not fully expanded **then**
14:              **return** n_{T-1}
15:          **else**
16:              $n_{T-1} \leftarrow$ child node of n_{T-1} with action sampling from Eqn. 13.
17:          **end if**
18:      **end while**
19:      **return** n_{T-1}
20: **end function**

21: **function** EXPANSION(n_{T-1})
22:      **if** n_{T-1} is non-terminal **and** $T \neq H$ **then**
23:          $n_T \leftarrow$ expand n_{T-1} with an untried action.
24:          **return** n_T
25:      **else**
26:          **return** n_{T-1}
27:      **end if**
28: **end function**

29: **function** SIMULATION&EVALUATION(n_T)
30:      **if** n_T is non-terminal **and** $T \neq H$ **then**
31:          $n_{\text{terminal}} \leftarrow$ Roll-out(n_T)
32:      **else**
33:          $n_{\text{terminal}} \leftarrow n_T$
34:      **end if**
35:      $\{s_T, a_T, \cdots, s_{\text{terminal}}, a_{\text{terminal}}\} \leftarrow$ extract from n_{terminal}.
36:      $R \leftarrow \sum_{t=T}^{\text{terminal}} \gamma^{t-T} r_R(s_t, a_t)$
37:      **return** R
38: **end function**

39: **function** BACK-PROPAGATION(n_T, R)
40:      $t \leftarrow T - 1$
41:      **while** $t \geq 0$ **do**
42:          $Q_R(s_t, a_t) \leftarrow$ Eqn. 14
43:          $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
44:          $t \leftarrow t - 1$
45:      **end while**
46: **end function**