Residual Policy Gradient: A Reward View of KL-regularized Objective

Pengcheng Wang, Xinghao Zhu, Yuxin Chen, Chenfeng Xu, Masayoshi Tomizuka, Chenran Li
Department of Mechanical Engineering, University of California, Berkeley

***Abstract*— In this work, we first derive a concise form of *Soft Policy Gradient* as a preliminary. Building on this, we introduce *Residual Policy Gradient* (RPG), which extends RQL to policy gradient methods, allowing policy customization in gradient-based RL settings. With the view of RPG, we rethink the KL-regularized objective widely used in RL fine-tuning. We show that under certain assumptions, KL-regularized objective leads to a maximum-entropy policy that balances the inherent properties and task-specific requirements on a reward-level. Our experiments in MuJoCo demonstrate the effectiveness of *Soft Policy Gradient* and *Residual Policy Gradient*.**

I. Introduction

Residual Q-Learning (RQL) [1] provides a principled approach to policy customization by formulating the problem as a Markov Decision Process (MDP), which jointly optimizies the reward of the prior and the add-on task-sepecift reward. However, it has primarily been applied in value-based RL and has not yet been extended to policy gradient methods, which restricts the applicability of RQL. This limitation becomes more pronounced when large-scale parallel environments are available, where policy gradient methods tend to outperform value-based methods [2]. Therefore, it is important to derive the policy gradient extension of RQL.

In this work, we first derive a concise form of maximum-entropy policy gradient, *Soft Policy Gradient*, as the preliminary and compare it with similar forms in the related literature. Based on the derived form, we further propose *Residual Policy Gradient* (RPG) to solve the policy customization in policy gradient approach. Our experiments in MuJoCo demonstrate the effectiveness of *Soft Policy Gradient* and *Residual Policy Gradient*. Moreover, with the view of RPG, we rethink the widely used KL-regularized objective in RL fine-tuning. We show that: (1) If the prior policy models an optimal Bolzmann distribution, the KL-regularized objective induces a maximum-entropy policy that balances inherent properties and task-specific requirements at the reward level. (2) This formulation can be further refined through a simple decoupling, leading to improved optimization efficiency. In Appendix V-B, we elaborate these insights with RL-based LLM fine-tuning, demonstrating their implications for both theoretical understanding and future directions.

II. Preliminaries: Residual Q-Learning

Consider a MDP defined by $\mathcal{M} = (\mathcal{X}, \mathcal{U}, r, p)$, whose corresponding objective of policy customization is to derive a new policy that optimally solves a modified MDP, $\hat{\mathcal{M}} = (\mathcal{X}, \mathcal{U}, \omega r + r_R, p)$, where r_R is the add-on task-specified reward and ω is the weight parameter. [1] proposed the residual Q-learning (RQL) framework to solve the policy customization. Given the prior policy π, RQL updates the Residual Q function $Q_R = Q^* - \omega Q$, where Q^* and Q is the optimal value function on $\mathcal{M}$ and $\hat{\mathcal{M}}$, RQL updates the Q_R using the following equation:

$$Q_R(\boldsymbol{s}, \boldsymbol{a}) \leftarrow r_R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{s'} \left[\hat{\alpha} \log \int_{\mathcal{A}} \exp\left(\frac{1}{\hat{\alpha}} \left(Q_R(\boldsymbol{s}', \boldsymbol{a}') + \omega \alpha \log \pi\left(\boldsymbol{a}' | \boldsymbol{s}'\right) \right) \right) d\boldsymbol{a}' \right], \tag{1}$$

where α and $\hat{\alpha}$ are the entropy-encouragement coefficients of maximum-entropy policy on prior and full task, and γ is the discounted factor. As shown in the RQL [1] appendix, by adding $\omega\alpha \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ to both side of Eqn. 1 and defining $Q^{aug}(\boldsymbol{s}, \boldsymbol{a}) = Q_R(\boldsymbol{s}, \boldsymbol{a}) + \omega\alpha \log \pi(\boldsymbol{a}|\boldsymbol{s})$, we can reformulate Eqn. 1 into:

$$Q^{aug}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow r_R(\boldsymbol{s}, \boldsymbol{a}) + \omega\alpha \log \pi(\boldsymbol{a}|\boldsymbol{s}) + \gamma \mathbb{E}_{s'} \left[\hat{\alpha} \log \int_{\mathcal{A}} \exp\left(\frac{1}{\hat{\alpha}} \left(Q^{aug}(\boldsymbol{s}', \boldsymbol{a}') \right) \right) d\boldsymbol{a}' \right], \tag{2}$$

which is a standard soft-Q update on an augmented MDP problem $\mathcal{M}^{\text{aug}} = (\mathcal{X}, \mathcal{U}, \omega' \log \pi(\boldsymbol{u}|\boldsymbol{x}) + r_R, p)$. This indicates that we can find the maximum-entropy policy solving the full MDP problem $\hat{\mathcal{M}}$ by solving an augmented MDP problem $\mathcal{M}^{\text{aug}}$, where $\omega' = \omega\alpha$ is a parameter that balances the optimality between basic and add-on tasks.

III. Residual Policy Gradient

In this section, we first derive a more netural form of the maximum-entropy policy gradient methods *Soft Policy Gradients*, which leads to *Residual Policy Gradient* (RPG). Furthermore, we propose *Soft PPO* and *Residual PPO* as two practical algorithms for deployment.

A. Soft Policy Gradient

As discussed in Sec. II, to utilize the augmented MDP problem $\mathcal{M}^{\text{aug}}$, we need to extend the policy gradient methods to optimize the maximum-entropy policy. For simplicity in deriviation, we remove the $\mathbb{E}_{\tau \sim p_\theta(\tau)}[\cdot]$ when presenting the equations. Let $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_t, \boldsymbol{a}_t, \ldots)$ represents a complete trajectory, and $p_\theta(\tau)$ represents the probability density of τ generated by a paramterized policy π_θ and the environment dynamics $p(\cdot|\boldsymbol{s}, \boldsymbol{a})$. The maximum-entropy objective $J(\pi_\theta)$ can be written as:

$$\sum_{t=0} \gamma^t \left(r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha \mathbb{E}_{\boldsymbol{a}_t \sim \pi_\theta(\cdot|\boldsymbol{s}_t)} \left[-\log \pi_\theta\left(\boldsymbol{a}_t | \boldsymbol{s}_t\right) \right] \right). \tag{3}$$

The estimation of its gradient $\nabla_\theta J(\pi_\theta)$ can be derived as:

$$\sum_{t=0} \nabla_\theta \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) \times \left(\sum_{t'=t} \gamma^{t'-t} \left(r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) - \alpha \log \pi_\theta \left(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'}\right)\right)\right) \tag{4}$$

This is equivalent to applying policy gradient over a reformulated reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)$. A detailed derivation can be found in Appendix V-A. Although some previous works [3], [4], [5], [6] also focused on deriving the maximum-entropy policy gradient, their formulations come with certain inconsistencies. With a unified notation, we compare and highlight their similarities and differences.

Naive Entropy-regularized PG [3], [4]. In the naive entropy-regularized policy gradient , the gradient estimator is encouraged with only an entropy term at the end, which is: $\sum_{t=0} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\sum_{t'=t} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'})\right) - \alpha \nabla_\theta \mathcal{H}\left(\pi_\theta\left(\cdot|\boldsymbol{s}_t\right)\right)$. However, such updating mechanism only considers the first-step ($t' = t$) entropy bonus when estimating the gradients, since $\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0} \nabla_\theta \mathcal{H}\left(\pi_\theta\left(\cdot|\boldsymbol{s}_t\right)\right)\right] = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_t \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \left(\log \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)\right)\right]$.

Repeat Entropy-regularized PG [6, Eqn.(80)]. In [6], the author introduced the proper entropy bonus on every step to the expected return but retained the $-\alpha \nabla_\theta \mathcal{H}\left(\pi_\theta\left(\cdot|\boldsymbol{s}_t\right)\right)$ term, which repeats the entropy bonus on the first step.

B. Practical Algorithm

By applying advantage function, we can reform the gradient$\nabla_\theta J(\pi_\theta)$:

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0} \nabla_\theta \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) A^{\mathrm{SPG}}_{\pi_\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right], \tag{5}$$

where $A^{\mathrm{SPG}}_{\pi_\theta}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1}\sim p(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)}[V^{\pi_\theta}(\boldsymbol{s}_{t+1})] - V^{\pi_\theta}(\boldsymbol{s}_t)$, and $V^{\pi_\theta}(\boldsymbol{s}_t) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t'=t} \gamma^{t'-t} \left(r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) - \alpha \log \pi_\theta \left(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'}\right)\right)\right]$. This is equivalent to applying classic policy gradient over $r^{\mathrm{SPG}} = r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)$. However, vallina policy gradient methods have several limitations, such as poor data-efficiency and robustness [7]. To enhance practical performance, we incorporate this reformulated reward into the PPO framework, leading to the Soft PPO method:

$$\min\{r_t(\pi_\theta) A^{\mathrm{SPG}}_{\pi_\theta}, \mathrm{clip}(r_t(\pi_\theta), 1-\epsilon, 1+\epsilon) A^{\mathrm{SPG}}_{\pi_\theta}\}. \tag{6}$$

With the derived Soft Policy Gradient in Sec. III-A, the residual policy can be obtained by directly applying it on the augmented MDP $\mathcal{M}^{\mathrm{aug}} = (\mathcal{S}, \mathcal{A}, r_R + \omega' \log \pi, p)$:

$$\min\{r_t(\pi_\theta) A^{\mathrm{RPG}}_{\pi_\theta}, \mathrm{clip}(r_t(\pi_\theta), 1-\epsilon, 1+\epsilon) A^{\mathrm{RPG}}_{\pi_\theta}\}. \tag{7}$$

where A^{RPG}_t is estimator of the advantage over the reformulated reward r^{RPG} at timestep t:

$$r^{\mathrm{RPG}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \omega' \log \pi \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) - \hat{\alpha} \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right). \tag{8}$$

C. A Reward View of KL-regularized Objective

1) Not Just KL-regularized Objective: The use of KL objective can be traced back to KL Control [8], which is a branch of stochastic optimal control, and was later introduced into RL [9], [10], [11]. The maximized objective follows the form [11, Eqn. 2]:

$$\mathbb{E}_{\pi_\theta}\left[\sum_{t'=t}^{\infty} r_R(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) + \beta \log \frac{\pi(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'})}{\pi_\theta(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'})}\right], \tag{9}$$

which aligns with the form of maximum-entropy RL objective in Eqn. 3 by taking $r = r_R + \log \pi$. Follow the derivation in Sec. III-A and Sec. III-B, this form could be view as a special case of RPG whose advantage is caculated on reward:

$$r^{\mathrm{KL}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \beta \log \pi \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) - \beta \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right). \tag{10}$$

Similarly in Eqn. 8, the decomposed KL divergence can be viewed as a combination of an augmentation reward $\beta \log \pi$ and an entropy-encouragement term $\beta \log \pi_\theta$. Specifically, if we further assume that prior policy π is a maximum-entropy policy encouraged by the same entropy coefficient of the customized policy $\alpha = \beta$, and the optimizing objective is the direct sum of the implicit reward and the preference reward $\omega = 1$, then the KL-regularized objective actually leads to a maximum-entropy policy on a MDP:

$$\mathcal{M}^{\mathrm{KL}} = (\mathcal{X}, \mathcal{U}, r + r_R, p). \tag{11}$$

This decoupling reveals that the KL-constrained reward has a theoretical foundation beyond its simple intuition to just prevent the fine-tuned model from drifting too far from the pretrained model. That is, it is optimizing a trade-off between basic and add-on task on a reward level.

2) Not have to be KL-regularized Objective: However, according to the derivation of RPG, such reward reparameterization is not limited to being implemented via KL divergence but can instead be decoupled into two tunable parameters ω' and $\hat{\alpha}$ as in Eqn. 8: $r^{\mathrm{RPG}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \omega' \log \pi \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right) - \hat{\alpha} \log \pi_\theta \left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)$.

In practice, the prior policy does not necessarily follow the maximum-entropy property, nor is it necessarily optimal. Meanwhile, the entropy-encouragement coefficient of the prior policy may not be accessible, and the newly trained policy does not necessarily need to maintain the same. This flexibility enables fine-tuning without strictly relying on the KL-regularized objective, allowing for a more adaptable approach. In the next section, we empirically validate this perspective through experiments by comparing Residual PPO and classic KL-regularized PPO.

In Appendix V-B, taking Reinforcement Learning for LLM fine-tuning as an example, we demonstrate how the insights in Sec. III-C.1 and Sec. III-C.2 deepen our understanding and informs future directions.

TABLE I: **Experimental Results of Entropy-Regularized PPO Variants in MuJoCo.** The evaluation results are over 500 episodes across 5 seeds, and in the form of $\mathrm{mean} \pm \mathrm{std}$.

Average Performance				
Env.	No-Entropy PPO	End-Entropy PPO	Repeat-Entropy PPO	Soft PPO
Ant	3918.9 ± 1032.9	3887.9 ± 991.4	3241.2 ± 1299.7	$\mathbf{3845.6 \pm 911.7}$
Hopper	2387.3 ± 1060.6	2696.3 ± 978.0	2611.6 ± 967.8	$\mathbf{3105.3 \pm 829.5}$
Half Cheetah	2254.9 ± 1188.7	3291.5 ± 2006.0	-114.9 ± 73.2	$\mathbf{4993.7 \pm 1262.3}$
Best Performance				
Env.	No-Entropy PPO	End-Entropy PPO	Repeat-Entropy PPO	Soft PPO
Ant	4389.6 ± 296.7	4223.4 ± 822.6	4023.9 ± 786.3	4581.1 ± 540.3
Hopper	3271.2 ± 573.4	3654.4 ± 46.4	3404.2 ± 226.0	3689.4 ± 29.2
Half Cheetah	4136.6 ± 1566.6	6049.3 ± 946.2	-38.9 ± 58.2	5896.5 ± 371.8

IV. Experiments

A. Soft PPO

In this section, we evaluate the preliminary algorithm, Soft PPO, across several benchmarks in MuJoCo.

Baselines. We choose the variants mentioned in Sec III-A:

- **No-Entropy PPO:** First, we evaluate the classic PPO without entropy term, serving as a baseline performance compared to entropy-regularized approaches.
- **End-Entropy PPO:** Next, we evaluate the form of [3], [4], which only considers the first-step entropy. By comparing against it, we investigate whether introducing the entropy term at each step enhances exploration.
- **Repeat-Entropy PPO:** Finally, we evaluate the form of [6], which repeats the entropy of the first step. By comparing against it, we investigate whether removing the repeated entropy term leads to improvements.

When modeling the policy distribution, we adopted the common implementation approach in PPO [4], using a global standard deviation instead of a state-dependent one. While this reduces the policy interpretability, it improves practical performance by simplifying learning and enhancing stability. A detailed ablation can be found in appendix V-D.3.

Does the proposed formulation achieve empirical performance improvements? As shown in Table I, compared to other variants of entropy-encouraged PPO, the proposed Soft PPO exhibits alike or potentially better average performance. Specifically, in the HalfCheetah task, we observed significant performance differences among variants with different entropy encouragement. This indicates that PPO can be very sensitive to the way to encourage entropy-regularized learning: although Repeat-Entropy PPO shares the same parameters and similar updating mechanism with Soft PPO, they can behave completely different during learning.

Notably, the best-performing seeds in the compared variants achieve similar performance to Soft PPO, whose overall performance declines are generally due to more seeds converging to the local sub-optimal. This further indicates that the concise formulation of Soft PPO can potentially better encourage the agent to explore the environment effectively. However, it is also worth noting that since End-Entropy PPO considers entropy encouragement in fundamentally different ways compared to Repeat-Entropy PPO and Soft PPO, they adopt different entropy coefficients, which remains room for better performance with more fine-grained tuning. In practices, we consider all these variants to be selectable based on the specific characteristics of the problem.

B. Residual PPO

In this section, we evaluate the proposed Residual PPO in the same MuJoCo environments but with extra add-on rewards, which are designed to illustrate customized specifications during practical deployments [12]. In HalfCheetah, an add-on penalty on the angle of a certain joint is applied to simulate a common issue in deployment when the corresponding motor is broken. In Hopper and Ant, the extra rewards are assigned on height and velocity for moving along the y-axis. The configurations of environments and training can be found in Appendix V-D.1 and Appendix V-D.2.

Metrics. We evaluate a policy's performance with the total reward $R = r + r_R$, since it represents the objective of the policy customization task. However, when the add-on task conflicts with the basic task, the optimal policy may sacrifice performance on the basic task to maximize the total reward. To better monitor this trade-off during customization, we separately measure the basic and add-on rewards, allowing us to assess how well the customized policy preserves the basic task's performance while optimizing for the add-on task [12].

Baselines. We mainly compare against four baselines:

- **Prior Policy:** First, we evaluate the best-performing Soft PPO checkpoint in Sec. IV-A on the full task, which serves as a baseline to show the effectiveness of policy customization.
- **Greedy PPO:** Next, we finetune the prior policy towards the add-on reward solely, which can be achieved by setting $\omega' = 0$, which serves as a trival baseline when basic reward is unknown.

TABLE II: **Experimental Results of Residual PPO in MuJoCo.** The evaluation results are over 500 episodes with best-performing checkpoints across 5 seeds. The results are in the form of $\mathrm{mean} \pm \mathrm{std}$.

Env.	Policy	Full Task	Basic Task	Add-on Task	
		Total Reward	Basic Reward	$\lvert\bar{\theta}\rvert$	Add-on Reward
Half Cheetah	Prior Policy	5357.0 ± 377.8	5896.5 ± 371.8	0.54 ± 0.01	-539.8 ± 8.7
	Greedy PPO	-1314.0 ± 252.4	-1150.2 ± 236.0	0.16 ± 0.02	-163.7 ± 21.8
	KL PPO	5418.7 ± 150.1	5776.3 ± 133.7	0.36 ± 0.02	-357.6 ± 17.7
	Residual PPO	$\mathbf{5488.3 \pm 75.6}$	5845.1 ± 69.5	$\mathbf{0.36 \pm 0.01}$	$\mathbf{-356.3 \pm 8.6}$
	Full Policy	5556.2 ± 57.8	6011.4 ± 57.2	0.46 ± 0.01	-455.2 ± 5.8
Env	Policy	Total Reward	Basic Reward	$\bar{z}$	Add-on Reward
Hopper	Prior Policy	5133.7 ± 33.2	3689.4 ± 29.2	1.44 ± 0.01	1444.3 ± 6.0
	Greedy PPO	3714.5 ± 7.1	2224.2 ± 6.5	1.49 ± 0.00	1490.2 ± 0.9
	KL PPO	3864.4 ± 50.10	2400.3 ± 55.2	1.46 ± 0.01	1464.1 ± 6.1
	Residual PPO	$\mathbf{4401.4 \pm 505.3}$	2951.6 ± 333.5	$\mathbf{1.50 \pm 0.03}$	$\mathbf{1449.5 \pm 177.0}$
	Full Policy	5142.7 ± 34.8	3669.0 ± 26.8	1.47 ± 0.00	1473.6 ± 8.4
Env.	Policy	Total Reward	Basic Reward	$\bar{v}_y$	Add-on Reward
Ant	Prior Policy	4875.2 ± 589.4	4581.1 ± 540.3	0.29 ± 0.13	294.1 ± 133.9
	Greedy PPO	4340.6 ± 1464.5	726.4 ± 277.1	3.86 ± 0.96	3613.9 ± 1246.4
	KL PPO	4019.4 ± 1103.4	368.6 ± 151.8	3.74 ± 0.92	3650.6 ± 1047.6
	Residual PPO	$\mathbf{5568.3 \pm 852.4}$	1340.8 ± 220.0	$\mathbf{4.27 \pm 0.63}$	$\mathbf{4227.4 \pm 705.3}$
	Full Policy	6185.1 ± 1142.0	3543.0 ± 621.2	2.70 ± 0.41	2641.6 ± 542.8

- **KL PPO:** Next, we utilize the KL divergence to regularize the fine-tuning process, which can be achieved by setting $\omega' = \hat{\alpha}$, which serves as a widely-used practical fine-tuning baseline.
- **Full Policy:** Finally, we utilize the Soft PPO to train an optimal policy on the full task, which serves as an upper-bound performance of policy customization

Can Residual PPO effectively solve the policy customization problem? The experimental results, summarized in Table II, demonstrate the effectiveness of Residual PPO for policy customization. Across all tasks, Residual PPO achieves obvious performance improvements on the add-on task compared to the prior policy, demonstrating its capability to optimize the prior policy for new requirements. Moreover, in the HalfCheetah and Ant, Residual PPO achieves performance comparable or alike performance of the Full Policy, which indicates that by incorporating the $\log \pi$, Residual PPO can effectively balance the trade-off between the basic and the add-on task.

Does the flexibility of decoupling lead to more effective policy customization? Greedy PPO, due to optimizing towards the add-on reward solely, struggles to maintain its performance on the basic task, resulting in a lower total reward compared to Residual PPO. This issue becomes more pronounced in tasks requiring auxiliary rewards, such as action regularization, where Greedy PPO fails to train an effective policy in the HalfCheetah task. KL PPO, in theory, should achieve the best performance as it optimizes a MDP with the exact objective $R = r + r_R$ and a reasonable entropy coefficient $\hat{\alpha} = \alpha$. However, as discussed in Sec. III-C.2, the maximum-entropy and optimality assumptions of RPG do not hold in practical scenarios, which can lead to even worse performance, as shown in Ant results. In contrast, the decoupling of the KL-regularized term in RPG provides greater tuning flexibility, which can lead to superior performance in practical applications. However, when this flexibility encounters excessively large assumption misalignments, it can also lead to suboptimal customization results, as observed in the Hopper task. This also suggests that when building a foundation model that is more adaptable, adequate modeling of complex distributions is an important direction for future work.

V. Conclusion

In this work, we derived a concise form of the maximum-entropy policy gradient and, based on this foundation, developed the *Residual Policy Gradient* as an important complement to RQL. With MuJoCo experiments, we validated the theoretical correctness of the proposed algorithms. Moreover, RPG reveals that the KL-constrained reward has theoretical foundation beyond its simple intuition. This enables us to establish clear connections between many successful RLHF methods and further explain certain experimental phenomena observed in them. The success of these works also serves as a strong empirical validation for our theoretical framework, laying a solid foundation for future RLHF and broader policy customization research.

APPENDIX

A. Soft Policy Gradient Derivation

Let $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_t, \boldsymbol{a}_t, \ldots)$ represents the complete trajectory, and $p_\theta(\tau)$ represents the probability density of τ generated by a paramterized policy π_θ and the environment dynamics $p(\cdot|\boldsymbol{s}, \boldsymbol{a})$. The maximum-entropy objective $J(\pi_\theta)$ can be written as:

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}\gamma^t\left(r(\boldsymbol{s}_t,\boldsymbol{a}_t)+\alpha\mathbb{E}_{\pi_\theta(\cdot|\boldsymbol{s}_t)}\left[-\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right]\right)\right]. \tag{12}$$

Noted the substitution of $\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right] = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right]]$, it can be reformulated:

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}\gamma^t\left(r(\boldsymbol{s}_t,\boldsymbol{a}_t)-\alpha\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right)\right]. \tag{13}$$

which is equivalent to having a reward defined as $r(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)$. The objective can be rewritten by defining $R(\tau) = \sum_{t=0}\gamma^t\left(r(\boldsymbol{s}_t,\boldsymbol{a}_t)-\alpha\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right)$:

$$J(\pi_\theta) = \int p_\theta(\tau)R(\tau)d\boldsymbol{\tau}. \tag{14}$$

By taking gradient over θ, we can obtain:

$$\nabla_\theta J(\pi_\theta) = \int \nabla_\theta p_\theta(\tau)R(\tau) + p_\theta(\tau)\nabla_\theta R(\tau)d\boldsymbol{\tau}. \tag{15}$$

Due to the Markov properties of the system and policy, the gradient of the probability density can be written as:

$$\begin{aligned}
&\nabla_\theta p_\theta(\tau)\\
=&p_\theta(\tau)\nabla_\theta\log p_\theta(\tau)\\
=&p_\theta(\tau)\nabla_\theta\left(\sum_{t=0}\log p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t,\boldsymbol{s}_t)+\log\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)\right)\\
=&p_\theta(\tau)\sum_{t=0}\nabla_\theta\log\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)
\end{aligned} \tag{16}$$

Inspired by [13], we drop γ^t for the reduced weight to increase data efficiency and ignore the rewards before time t for each $\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)$ to reduce variance . Therefore, we can simplify the second term:

$$\begin{aligned}
&\int p_\theta(\tau)\nabla_\theta R(\tau)d\boldsymbol{\tau}\\
=&\int p_\theta(\tau)\left(\sum_{t=0}-\alpha\gamma^t\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right)d\boldsymbol{\tau}\\
\approx&-\alpha\int p_\theta(\tau)\left(\sum_{t=0}\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right)d\boldsymbol{\tau}\\
=&-\alpha\int\nabla_\theta p_\theta(\tau)d\boldsymbol{\tau}\\
=&-\alpha\nabla_\theta\int p_\theta(\tau)d\boldsymbol{\tau}\\
=&-\alpha\nabla_\theta 1\\
=&0,
\end{aligned} \tag{17}$$

and then we apply the same result on Eqn. 15: $\nabla_\theta J(\pi_\theta)$

$$\begin{aligned}
=&\int\nabla_\theta p_\theta(\tau)R(\tau)d\boldsymbol{\tau}\\
=&\int p_\theta(\tau)\left(\sum_{t=0}\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\right)R(\tau)d\boldsymbol{\tau}\\
=&\int p_\theta(\tau)\sum_{t=0}\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\\
&\left(\sum_{t'=0}\gamma^{t'}\left(r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})-\alpha\log\pi_\theta\left(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'}\right)\right)\right)d\boldsymbol{\tau}\\
\approx&\int p_\theta(\tau)\sum_{t=0}\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\\
&\left(\sum_{t'=t}\gamma^{t'-t}\left(r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})-\alpha\log\pi_\theta\left(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'}\right)\right)\right)d\boldsymbol{\tau}\\
=&\mathbb{E}_{\tau\sim p_\theta(\tau)}\sum_{t=0}\nabla_\theta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)\\
&\left(\sum_{t'=t}\gamma^{t'-t}\left(r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})-\alpha\log\pi_\theta\left(\boldsymbol{a}_{t'}|\boldsymbol{s}_{t'}\right)\right)\right)
\end{aligned}$$

B. Example: Reinforcement Learning for LLM Fine-tuning

Reinforcement Learning for LLM fine-tuning represents a canonical instance of policy customization, where the exact reward of the original LLM is unknown. However, the objective remains to fine-tune the LLM to align with preferences while preserving its prior conversational capabilities. Here we show how the insights in Sec. III-C.1 and Sec. III-C.2 deepen our understanding and informs future research.

Vanilla RLHF [14, Eqn. 7] follows the intuition of [11], adding a KL term to reward when computing advantage, as:

$$r^{\mathrm{RLHF}}(\boldsymbol{s}_t,\boldsymbol{a}_t) = r_R(\boldsymbol{s}_t,\boldsymbol{a}_t)+\beta\log\pi\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)-\beta\log\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right). \tag{19}$$

As analyzed, if we assume that the distribution of human language can be represented as a Boltzmann distribution from an implicit reward r_{human}, and that the original LLM models an approximately optimal distribution, then Vanilla RLHF is actually optimizing on a MDP:

$$\mathcal{M}^{\mathrm{RLHF}} = (\mathcal{X}, \mathcal{U}, r_{\mathrm{human}} + r_R, p). \tag{20}$$

Direct Preference Optimization (DPO) [15], [16] also realizes the importance of reward-level modeling. It defines a reparameterized reward as:

$$r^{\mathrm{DPO}}\left(\boldsymbol{s}_t,\boldsymbol{a}_t\right) = \beta\log\frac{\pi_\theta\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)}{\pi\left(\boldsymbol{a}_t|\boldsymbol{s}_t\right)}, \tag{21}$$

and a reward loss of preference demonstrations $\{x^{(i)}, y_w^{(i)}, y_l^{(i)}\}_{i=1}^N$ under Bradley-Terry model:

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta;\pi) = \log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi(y_l|x)}\right), \tag{22}$$

thus transforming policy learning into reward learning from demonstrations. From the view of RPG, the reparameterization of DPO could be easily derived by modeling the log full policy $\beta\log\pi_\theta(\boldsymbol{s}_t|\boldsymbol{a}_t)$ as the total reward $R\left(\boldsymbol{s}_t,\boldsymbol{a}_t\right)$:

$$r_R(\boldsymbol{s}_t, \boldsymbol{a}_t) = R(\boldsymbol{s}_t, \boldsymbol{a}_t) - r_{\text{human}}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \beta \log \frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)}$$
$$= r^{\text{DPO}}(\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{23}$$

Moreover, with the insights of RPG, we can identify several potential impovervements starting from DPO. First, modeling $\log \pi$ as reward requires it to be a maximum-entropy policy, which means that a proper reward loss that aligns with this assumption, e.g., cross-entropy loss, can lead to potentially better performance:

$$\begin{aligned} &l \cdot \log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi(y_w|x)} \right) + \\ &(1-l) \cdot \log \left[1 - \sigma \left(\beta \log \frac{\pi_\theta(y_l|x)}{\pi(y_l|x)} \right) \right]. \end{aligned} \tag{24}$$

The experimental results from a recent work, Implicit PRM [17], provide supporting evidence for our insights: when modeling π_θ, using cross-entropy as the loss function tends to yield higher data efficiency compared to other losses.

Second, the reparameterized reward does not have to be a KL-regularized objective in practice, which means that we can develop a more flexible form of DPO loss:

$$\log \sigma \left(\hat{\alpha} \log \frac{\pi_\theta(y_w|x)}{\pi_\theta(y_l|x)} - \omega' \log \frac{\pi(y_w|x)}{\pi(y_l|x)} \right). \tag{25}$$

Recently, Q-Adapter [18], an RLHF framework based on RQL, has provided further validation of this approach by demonstrating that adjusting the value of ω' can effectively balance the trade-off and even enhance performance on both the basic and add-on tasks in RLHF. Furthermore, Q-Adapter follows the RQL formulation, integrating the $\log \pi$ term into the value function update rather than incorporating it in an augmented manner. This design improves data efficiency and enhances stability, a key aspect also discussed in RQL [1].

C. Related Work

KL-regularized Objective. The KL-regularized objective is related to KL Control [8], [19], which aims to prevent the target policy from deviating too far from a reference, which is widely adopted in RL. Ψ-learning [10] and G-learning [9] both maximize the reward while incurring penalties for the KL-divergence from a reference policies. The KL-regularized objective is also a common approach in transfer learning [20] and RL-based finetuning [21], especially for fine-tuning LLM with human preference.

The flexibility of KL decoupling discussed in Sec. III-C.2 can be achieved through alternative approaches. Distral [22] simultaneously employs KL divergence and entropy with different coefficients to regularize the joint training of the task-specific and meta policy. TD-M(PC)^2 [23] uses the log distribution to bring the nominal policy and planner closer during learning, leading to a more performant policy. However, these results are derived from practical usage, whereas RPG provides a reward-level perspective to support such implementations.

Many works originating from preference-based RL have ultimately derived results similar to RQL, and converge toward maximum-entropy RL and specifically the $\log \pi$ form. IQ-Learn [24] and IPL [25] all recognized that learning the maximum-entropy policy or Q-value provides an implicit yet efficient way to capture preferences from data, leading to an updating approach similar to RQL. However, these methods focus on Inverse RL, whereas RQL formulates a training pipeline from the perspective of optimizing a new policy.

Residual Q-Learning. RQL [1] proposes a principled framework to solve policy customization without the reward knowledge of the prior policy. However, this framework is not limited to RL-based approaches or solely addressing the policy customization: inspired by RQL, Residual-MPPI [12] integrates model-based planning to enable online policy customization, while MEReQ [26] leverages RQL as an efficient approach for inverse RL.

D. MuJoCo Experiments

1) Environment Configuration: In this section, we introduce the detailed configurations of the selected environments, including the basic tasks, add-on tasks, and the corresponding rewards. The action and observation space of all the environments follow the default settings in `gym[mujoco]-v3`.

Half Cheetah. In HalfCheetah, the basic goal is to apply torque on the joints to make the cheetah run forward as fast as possible. The state and action space has 17 and 6 dimensions, and the action represents the torques applied between links. The basic reward consists of two parts: forward reward $r_{\text{forward}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{\Delta x}{\Delta t}$ and control cost $r_{\text{control}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = -0.1 \times ||\boldsymbol{u}_t||_2^2$. During policy customization, we demand an additional task that requires the cheetah to limit the angle of its hind leg. This customization goal is formulated as an add-on rewards defined as $r_R(\boldsymbol{x}_t, \boldsymbol{u}_t) = -|\theta_{\text{hind leg}}|$

Hopper. In Hopper, the basic goal is to make the hopper move in the forward direction by applying torques on the three hinges. The state and action space has 11 and 3 dimensions, and the action represents the torques applied between links. The basic reward consists of three parts: alive reward $r_{\text{alive}} = 1$, forward reward $r_{\text{forward}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{\Delta x}{\Delta t}$, control cost $r_{\text{control}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = -0.001 \times ||\boldsymbol{u}_t||_2^2$. During policy customization, we demand an additional task that requires the hopper to jump higher. This customization goal is formulated as an add-on reward defined as $r_R(\boldsymbol{x}_t, \boldsymbol{u}_t) = z$.

Ant. In Ant, the basic goal is to coordinate the four legs to move in the forward direction by applying torques on the eight hinges. The state and action space has 27 and 8 dimensions, and the action represents the torques applied at the hinge joints. The basic reward consists of three parts: alive reward $r_{\text{alive}} = 1$, forward reward $r_{\text{forward}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{\Delta x}{\Delta t}$, control cost $r_{\text{control}}(\boldsymbol{x}_t, \boldsymbol{u}_t) = -0.5 \times ||\boldsymbol{u}_t||_2^2$. During policy customization, we demand an additional task that requires the ant to move along the y-axis. This customization goal is formulated as an add-on reward defined as $r_R(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{\Delta y}{\Delta t}$.

2) Implementation Setting: The training pipeline is developed upon rsl-rl [27] implementation. All variants are prior policies and customized policies share the task-specific

TABLE III: **Ablation of policy distribution modeling in MuJoCo Experiments.** The evaluation results are computed over 500 episodes across 5 seeds. The results are in the form of $\mathrm{mean} \pm \mathrm{std}$

Env.	Average Performance		Best Performance	
	Global Std.	State Std.	Global Std.	State Std.
Ant	$\mathbf{3845.6 \pm 911.7}$	3055.3 ± 1231.5	$\mathbf{4581.1 \pm 540.3}$	3915.1 ± 599.9
Hopper	$\mathbf{3105.3 \pm 829.5}$	1504.0 ± 948.3	$\mathbf{3689.4 \pm 29.2}$	2511.6 ± 784.6
Half Cheetah	$\mathbf{4993.7 \pm 1262.3}$	2621.7 ± 1605.3	$\mathbf{5896.5 \pm 371.8}$	5405.0 ± 1288.8

parameters finetuned from the predefined settings in this implementation The policy customization process is executed for 5M steps in the environment, where the actor model is directly loaded from the prior policy. Additionally, the first 5% of steps are used to fit a reinitialized value function to stabilize the training process. The primary differences among variants lie in the choice of entropy coefficient and augmentation coefficient shown in Table IV and Table V.

TABLE IV: Entropy-Encourage Coefficient α

α	Half Cheetah	Ant	Hopper
End-Entropy PPO	0.01	0.001	0.0005
Repeat-Entropy PPO	0.13472	0.0001	0.001
Soft PPO	0.13472	0.0001	0.001

TABLE V: Augmentation Coefficient ω'

ω'	Half Cheetah	Ant	Hopper
Greedy PPO	0	0	0
KL PPO	0.13472	0.0001	0.001
Residual PPO	0.17	0.01	0.01

3) Global Std. Ablation: As shown in Table III, despite the sacrifices in interpretability, the widely adopted global std design outperforms the state-dependent std variant in both average performance and best performance in practices.

REFERENCES

[1] C. Li, C. Tang, H. Nishimura, J. Mercat, M. Tomizuka, and W. Zhan, "Residual q-learning: Offline and online policy customization without value," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[2] M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin, "Simplifying deep temporal difference learning," *arXiv preprint arXiv:2407.04811*, 2024.

[3] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[4] V. Mnih, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.

[5] W. Shi, S. Song, and C. Wu, "Soft policy gradient method for maximum entropy deep reinforcement learning," *arXiv preprint arXiv:1909.03198*, 2019.

[6] J. Schulman, X. Chen, and P. Abbeel, "Equivalence between policy gradients and soft q-learning," *arXiv preprint arXiv:1704.06440*, 2017.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] E. Todorov, "Linearly-solvable markov decision problems," *Advances in neural information processing systems*, vol. 19, 2006.

[9] R. Fox, A. Pakman, and N. Tishby, "Taming the noise in reinforcement learning via soft updates," *arXiv preprint arXiv:1512.08562*, 2015.

[10] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," 2013.

[11] N. Jaques, S. Gu, D. Bahdanau, J. M. Hernández-Lobato, R. E. Turner, and D. Eck, "Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1645–1654.

[12] P. Wang, C. Li, C. Weaver, K. Kawamoto, M. Tomizuka, C. Tang, and W. Zhan, "Residual-mppi: Online policy customization for continuous control," *arXiv preprint arXiv:2407.00898*, 2024.

[13] P. Thomas, "Bias in natural actor-critic algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 441–448.

[14] 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.

[15] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] R. Rafailov, J. Hejna, R. Park, and C. Finn, "From r to Q^*: Your language model is secretly a q-function," *arXiv preprint arXiv:2404.12358*, 2024.

[17] L. Yuan, W. Li, H. Chen, G. Cui, N. Ding, K. Zhang, B. Zhou, Z. Liu, and H. Peng, "Free process rewards without process labels," *arXiv preprint arXiv:2412.01981*, 2024.

[18] Y.-C. Li, F. Zhang, W. Qiu, L. Yuan, C. Jia, Z. Zhang, Y. Yu, and B. An, "Q-adapter: Customizing pre-trained llms to new preferences with forgetting mitigation," *arXiv preprint arXiv:2407.03856*, 2024.

[19] H. J. Kappen, V. Gómez, and M. Opper, "Optimal control as a graphical model inference problem," *Machine learning*, vol. 87, pp. 159–182, 2012.

[20] D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, G. Wayne, R. Pascanu, Y. W. Teh, and N. Heess, "Exploiting hierarchy for learning and transfer in kl-regularized rl," *arXiv preprint arXiv:1903.07438*, 2019.

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

[22] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017.

[23] H. Lin, P. Wang, J. Schneider, and G. Shi, "Td-m (pc) ^{2}: Improving temporal difference mpc through policy constraint," *arXiv preprint arXiv:2502.03550*, 2025.

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

[25] J. Hejna and D. Sadigh, "Inverse preference learning: Preference-based rl without a reward function," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[26] Y. Chen, C. Tang, C. Li, R. Tian, W. Zhan, P. Stone, and M. Tomizuka, "Mereq: Max-ent residual-q inverse rl for sample-efficient alignment from intervention," *arXiv preprint arXiv:2406.16258*, 2024.

[27] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proceedings of the 5th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 164. PMLR, 2022, pp. 91–100. [Online]. Available: https://proceedings.mlr.press/v164/rudin22a.html