Reward Redistribution via Gaussian Process Likelihood Estimation

Minheng Xiao[1], **Xian Yu***[1]

[1] The Ohio State University
xiao.1120@osu.edu, yu.3610@osu.edu

Abstract

In many practical reinforcement learning tasks, feedback is only provided at the end of a long horizon, leading to sparse and delayed rewards. Existing reward redistribution methods typically assume that per-step rewards are independent, thus overlooking interdependencies among state–action pairs. In this paper, we propose a Gaussian Process-based Likelihood Reward Redistribution (GP-LRR) framework that addresses this issue by modeling the reward function as a sample from a Gaussian Process (GP), which explicitly captures dependencies between state–action pairs through the kernel function. By maximizing the likelihood of the observed episodic return via a *leave-one-out* strategy that leverages the entire trajectory, our framework inherently introduces uncertainty regularization. Moreover, we show that the conventional mean squared error (MSE)-based reward redistribution arises as a special case of our GP-LRR framework when using a degenerate kernel without observation noise. When integrated with an off-policy algorithm such as Soft Actor-Critic, GP-LRR yields dense and informative reward signals, resulting in superior sample efficiency and policy performance on several MuJoCo benchmarks.

Code — https://github.com/xiao-1120/AAAI-LRR

Introduction

Reinforcement learning (RL) has been successfully applied in a wide range of fields, including finance (Hambly, Xu, and Yang 2023), healthcare (Yu et al. 2021), robotics (Kober, Bagnell, and Peters 2013), and autonomous driving (Kiran et al. 2021). However, a critical challenge remains: in many practical systems, feedback is provided only at the end of an episode after a long sequence of actions, rather than immediately. For example, in aerospace design, the quality of an aircraft component is evaluated only after the complete manufacturing process is finished (Razzaghi et al. 2024). This delayed feedback creates an extremely sparse reward landscape, making it difficult to determine which actions most significantly influenced the final outcome. Consequently, the learning process may converge slowly or become trapped in suboptimal policies. Overcoming this bottleneck is essential for developing more robust and efficient RL solutions across diverse application domains.

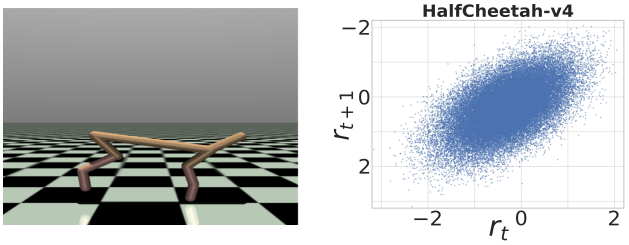


Figure 1: (Left) HalfCheetah-v4 environment visualization. (Right) Lag-1 autocorrelation in rewards collected under random policy, where each point represents a (r_t, r_{t+1}) pair from trajectory rollouts.

Recent studies have started to address these issues from different perspectives (Arjona-Medina et al. 2019; Gangwani, Zhou, and Peng 2020; Ren et al. 2021). A widely adopted strategy to tackle sparse rewards is to decompose the episodic return into per-step contributions (credits), thereby generating dense reward signals. For example, RUDDER uses an LSTM to predict per-step returns and redistributes credit through backpropagation. The model is trained by minimizing the mean squared error (MSE) between predicted and actual returns. RRD (Ren et al. 2021) samples random trajectory subsequences to approximate optimal decomposition, and IRCR (Gangwani, Zhou, and Peng 2020) simply distributes returns uniformly across all timesteps. However, conventional return decomposition approaches typically assume that each step's reward is independent from each other. This assumption overlooks the interdependencies between state-action pairs on a trajectory in many tasks. A simple empirical evidence shows significant lag-1 autocorrelation in several environments (see Figure 1), indicating that rewards are interdependent over time due to the correlation among state-action pairs in consecutive steps. Ignoring such dependencies can lead to ineffective reward redistribution, where important interactions between actions are missed, eventually impairing learning efficiency.

To model such interdependence in the per-step rewards between different state-action pairs, Gaussian Process (GP) is a useful tool. Unlike the aforementioned regression ap-

*Corresponding author

proaches that assume the reward function to lie in a parametric family, GP is a nonparametric approach that treats the unknown reward function as a black box and defines a distribution over it. Through the kernel function (covariance), GP explicitly models the correlations of reward functions between different state-action pairs. In prior works, GP has most commonly been employed either for value-function approximation (Engel, Mannor, and Meir 2005; Chung, Lawrance, and Sukkarieh 2013) or to model the transition dynamics (Deisenroth and Rasmussen 2011). Recently, (Wei, Liu, and Ying 2024) uses GP to approximate the cost function and provides regret bounds on safe RL with instantaneous constraints. However, to the best of our knowledge, this is the first work that leverages GP for reward redistribution in RL.

In this paper, we propose a novel GP-based reward redistribution framework. Instead of treating per-step rewards as isolated signals, our method assumes that rewards arise from a latent function with structured correlations captured by a kernel function. By leveraging a *leave-one-out* (LOO) strategy, our framework computes the likelihood of the observed total return while naturally incorporating dependencies between state-action pairs across the entire trajectory. Our main contributions can be summarized as follows.

- We introduce a GP-based Likelihood Reward Redistribution framework (GP-LRR) that models per-step reward as a sample from a GP and maximizes the likelihood of a trajectory based on LOO targets. This framework explicitly models the interdependent structure of per-step rewards and does not assume any parametric family for the reward function.
- We provide theoretical analyses showing that (i) GP-LRR generalizes existing MSE-based approaches, (ii) through the kernel function and precision matrix, GP-LRR explicitly captures the correlations between different state-action pairs and pools their prediction errors together to update the parameters, and (iii) GP-LRR implicitly regularizes reward estimates via its predictive uncertainty, avoiding overconfident assignments in poorly explored regions.
- We develop a practical algorithm that integrates GP-LRR with Soft Actor-Critic (SAC), demonstrating superior sample efficiency and asymptotic performance compared to state-of-the-art baselines on several MuJoCo benchmarks.

Related Works

We review the most relevant approaches for handling sparse and delayed rewards in RL through reward redistribution.

LIRPG (Zheng, Oh, and Singh 2018) addresses sparse rewards by jointly learning a policy and an intrinsic reward. Meta-gradients ensure the intrinsic signal improves the true objective, but the on-policy requirement limits sample efficiency.

GASIL (Guo et al. 2018) uses a GAN-style discriminator trained on high-return trajectories to provide dense rewards. It yields continuous learning signals but is on-policy and prone to mode collapse, hindering broader exploration.

RUDDER (Arjona-Medina et al. 2019) converts episodic returns into dense per-step signals via LSTM-based return decomposition, minimizing MSE to the true return. Long-horizon BPTT is unstable and costly, and complexity grows with trajectory length.

IRCR (Gangwani, Zhou, and Peng 2020) uniformly distributes the episodic return across timesteps. It is computation-light but ignores temporal structure, failing to isolate the truly responsible actions.

RRD (Ren et al. 2021) samples short trajectory subsequences and minimizes a variance-regularized objective, balancing cost and accuracy. It integrates well with off-policy RL, but still ignores state–action dependencies and lacks a global view.

GRD (Zhang et al. 2023) learns a causal generative model over states, actions, and latent Markovian rewards, proving identifiability. It produces interpretable, policy-invariant proxy rewards and compact state representations, outperforming IRCR/RRD on delayed-reward MuJoCo with SAC.

DIASter (Lin et al. 2024) decomposes return by cutting a trajectory into two sub-trajectories and defining step-wise rewards as expected differences of assigned sub-trajectory returns. It offers return-equivalence and near-optimal guidance guarantees, uses a GRU implementation (Cho et al. 2014), and improves sample efficiency on MuJoCo.

Overall, the above methods either rely on on-policy learning with limited sample efficiency, ignore interdependencies among state–action pairs during return redistribution, or employ heavy sequential neural networks over (s, a) that incur substantial computational burden.

Preliminaries

MSE-based Reward Redistribution

Traditional RL approaches typically assume that the environment can be modeled as a Markov decision process (MDP) characterized by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \mu \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(s'|s, a)$ captures the unknown transition dynamics, and $R(s, a)$ is the reward function. The initial state distribution is given by μ. The goal is to learn a policy that maximizes the expected discounted sum of rewards:

$$J(\pi) = \mathbb{E}^{\pi}_{s\sim\mu}\left[\sum_{t=0}^{\infty} \gamma^t \, R(s_t, a_t) \mid s_0 = s\right], \qquad (1)$$

where $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$, $a_t \sim \pi(\cdot|s_t)$ and $\gamma \in (0, 1)$ is the discount factor. However, in many real-world tasks, the per-step reward signal $R(s, a)$ remains unknown and the feedback is only provided at the end of an episode. Let $\tau = \{s_0, a_0, s_1, \ldots, s_T\}$ be a trajectory with a cumulative (or episodic) reward $R_{\text{ep}}(\tau)$. In this episodic setting, the objective becomes

$$J_{\text{ep}}(\pi) = \mathbb{E}\left[R_{\text{ep}}(\tau) \,\middle|\, \tau \sim \pi, P\right].$$

Because intermediate rewards are not provided, the decision process loses its strict Markovian property. Nonetheless, it is often assumed that the overall return can be approximately

decomposed additively as

$$R_{\text{ep}}(\tau) \approx \sum_{t=0}^{T-1} R_b(s_t, a_t),$$

where $R_b(s, a)$ is the underlying (unknown) per-step reward function.

To overcome the challenges associated with sparse episodic feedback, reward redistribution techniques aim to derive a surrogate reward function for $R_b(s, a)$ that transforms the delayed signal into a dense, step-by-step reward. A popular method to achieve this is through *return decomposition* (Arjona-Medina et al. 2019; Efroni et al. 2021). In this approach, one trains a parameterized reward model $R_\theta(s, a)$ by minimizing the mean squared loss

$$L_{\text{RD}}(\theta) = \mathbb{E}_{\tau\sim D}\left[\left(R_{\text{ep}}(\tau) - \sum_{t=0}^{T-1} R_\theta(s_t, a_t)\right)^2\right], \quad (2)$$

where D is the dataset of collected trajectories. By minimizing this loss, the model is encouraged to assign local rewards that, when summed over the trajectory, closely match the true episodic return. Once a reliable proxy $R_\theta(s, a)$ is obtained, it can be used in the downstream policy optimization task to replace the sparse terminal reward with a more informative, dense signal, thereby accelerating policy optimization in environments with delayed rewards.

However, such approaches treat the per-step rewards as independent signals and impose strong assumptions on the parametric family. In this paper, we propose a nonparametric approach, GP-LRR, that explicitly models the correlations between different rewards and updates the parameters by maximizing the likelihood. We will demonstrate that GP-LRR encompasses the MSE approaches as special cases.

Soft Actor-Critic

Soft Actor–Critic (SAC) (Haarnoja et al. 2018) is an off-policy Actor–Critic algorithm that augments the standard RL objective with an entropy term. Instead of optimizing the objective (1), SAC seeks a policy π that maximizes

$$J_{\text{SAC}}(\pi) = \mathbb{E}^{\pi}_{s\sim\mu}\left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))) \mid s_0 = s\right],$$

where $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a\sim\pi(\cdot|s_t)}[\log \pi(a|s_t)]$ is the Shannon entropy of the policy at state s_t and the temperature $\alpha > 0$ controls the reward-versus-exploration trade-off. In the policy-evaluation stage, it updates a soft Q-function Q_ω by minimizing the mean-squared soft Bellman residual

$$\mathcal{L}_Q(\omega) = \mathbb{E}_{\substack{(s,a,r,s')\sim\mathcal{D} \\ a'\sim\pi_\zeta}}\left[\frac{1}{2}\Big(Q_\omega(s, a) - \hat{Q}_\omega(s, a)\Big)^2\right], \quad (3)$$

where $\hat{Q}_\omega(s, a)$ is the soft TD-target defined as

$$\hat{Q}_\omega(s, a) = r + \gamma(Q_{\bar{\omega}}(s', a') - \alpha \log \pi_\zeta(a'|s')), \quad (4)$$

where $Q_{\bar{\omega}}$ is a slowly polyak-averaged target network and $\mathcal{D}$ is the replay buffer. In the policy-improvement stage, SAC performs a gradient step which minimizes the objective

$$J_{\text{actor}}(\zeta) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\zeta(\cdot|s)}\big[\alpha \log \pi_\zeta(a|s) - Q_\omega(s, a)\big]. \quad (5)$$

Since the optimal temperature depends on how stochastic one wishes the final policy to be, SAC learns the temperature α online by minimizing

$$J_{\text{temp}}(\alpha) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\zeta(\cdot|s)}\big[-\alpha \log \pi_\zeta(a|s) - \alpha\bar{\mathcal{H}}\big], \quad (6)$$

so that the realized entropy matches a user-specified target $\bar{\mathcal{H}}$. By embedding the entropy bonus directly in both the critic target and the actor loss, SAC achieves high sample-efficiency, remains robust to function-approximation error, and attains state-of-the-art performance on continuous-control benchmarks.

We will integrate GP-LRR with SAC by using the current estimated mean function as the reward signal for SAC updates. While collecting trajectories with SAC, we periodically train the GP model on complete episodes to learn the reward redistribution and then use these redistributed rewards instead of the sparse episodic signal in SAC's Q-function and policy updates.

GP-based Likelihood Reward Redistribution

In this section, we present a GP-based reward redistribution framework that leverages probabilistic modeling and nonparametric regression. Unlike conventional methods that rely on parametric regression approaches (Arjona-Medina et al. 2019) or subsample segments of trajectories (Ren et al. 2021)—both of which treat rewards as isolated, independent signals, our method models the per-step proxy reward as a random variable from a GP and computes the likelihood over the entire trajectory using the LOO strategy. We then update the parameters in the GP by maximizing the likelihood. As we will show in the Theoretical Analyses section, our framework includes the conventional MSE-based reward redistribution (2) as a special case and inherently incorporates an uncertainty regularizer that robustly mitigates trivial solutions. Furthermore, GP-LRR enables gradient information pooling through explicit correlation modeling.

Gaussian Process Likelihood

In this section, we present a principled reward redistribution framework based on GP likelihood estimation. Our method explicitly models correlations between different state-action pairs through a kernel function, enabling effective credit assignment across temporally and spatially related states. The key insight is that similar state-action pairs should yield similar rewards—a structure we exploit through the GP's kernel function.

Gaussian Process Reward Modeling Consider the per-step reward $R(s, a)$ at each state-action pair as a random variable following

$$R(s, a) = R_b(s, a) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2_\epsilon), \quad (7)$$

where $R_b(s,a)$ is the unknown ground-truth reward function at (s,a) and ϵ represents the Gaussian noise. Instead of assuming the ground truth reward model $R_b(s,a)$ to lie in a parametric family, we model this as a black-box function and a possible sample from a GP:

$$R_b(s,a) \sim \mathcal{GP}\big(\mu_{\boldsymbol{\theta}}(s,a), k_{\boldsymbol{\phi}}((s,a),(s',a'))\big),$$

where $\mu_{\boldsymbol{\theta}}(s,a)$ is the mean function of the reward at state-action pair (s,a) parameterized by $\boldsymbol{\theta}$, and $k_{\boldsymbol{\phi}}((s,a),(s',a'))$ is the kernel function (covariance) between any two state-action pairs (s,a) and (s',a') with hyperparameters $\boldsymbol{\phi}$. Note that in the case of discrete state and action spaces, the mean function can be exactly represented by a lookup table $\mu(s,a)$ without any function approximation. To accommodate large-scale state/action space and continuous control environments, we will continue with the parameterized function $\mu_{\boldsymbol{\theta}}(s,a)$ and utilize neural networks to represent it in our experiments. To measure the correlation between different state-action pairs, one option is to use the Radial Basis Function (RBF) kernel (Duvenaud 2014):

$$k_{\boldsymbol{\phi}}((s,a),(s',a')) = \sigma_f^2 \exp\left(-\frac{\|(s,a)-(s',a')\|^2}{2\ell_{\text{rbf}}^2}\right), \tag{8}$$

where $\boldsymbol{\phi} = [\sigma_f^2, \ell_{\text{rbf}}]$ with σ_f^2 being the signal variance and ℓ_{rbf} being the characteristic length scale. We derive the gradients based on this RBF kernel in the following and present other options (e.g., Matérn kernel and Rational Quadratic) in the Appendix and numerical experiments. For any trajectory $\tau = \{(s_0,a_0),(s_1,a_1),\ldots,(s_{T-1},a_{T-1})\}$, the joint ground-truth reward vector follows a multivariate Gaussian:

$$\mathbf{r}_b = [R_b(s_0,a_0),\ldots,R_b(s_{T-1},a_{T-1})]^\top \sim \mathcal{N}\big(\boldsymbol{\mu}_{\boldsymbol{\theta}}, \mathbf{K}_{\boldsymbol{\phi}}\big),$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}} = [\mu_{\boldsymbol{\theta}}(s_0,a_0), \mu_{\boldsymbol{\theta}}(s_1,a_1),\ldots,\mu_{\boldsymbol{\theta}}(s_{T-1},a_{T-1})]^\top$ and $[\mathbf{K}_{\boldsymbol{\phi}}]_{ij} = k_{\boldsymbol{\phi}}((s_i,a_i),(s_j,a_j))$.

To compute the likelihood of such a reward vector, we need a proxy reward for each step. In environments with only episodic reward $R_{ep}(\tau)$, we employ the following LOO strategy to construct training targets. Specifically, for each time step i, we define the LOO target as the following:

$$\tilde{r}(s_i,a_i) = R_{\text{ep}}(\tau) - \sum_{t=0,t\neq i}^{T-1} \mu_{\boldsymbol{\theta}}(s_t,a_t). \tag{9}$$

These LOO targets serve as noisy observations of the ground-truth rewards, forming a training dataset $\mathcal{D}_\tau = \{((s_i,a_i),\tilde{r}(s_i,a_i))\}_{i=0}^{T-1}$. Given such a dataset, we optimize a *pseudo-likelihood* that treats the LOO targets as noisy observations:

$$p(\tilde{\mathbf{r}} \mid \boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}, \mathbf{K}_{\boldsymbol{\phi}} + \sigma_\epsilon^2\mathbf{I}). \tag{10}$$

Throughout this section, the LOO targets $\tilde{\mathbf{r}}$ are constructed at the current $\boldsymbol{\theta}$ and then treated as constants during gradient updates (i.e., no backpropagation through $\tilde{\mathbf{r}}$).

To learn the hyperparameters $\boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon$, we maximize the log marginal likelihood of the reward targets on trajectory τ, which corresponds to the following:

$$\log p(\tilde{\mathbf{r}} \mid \boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon) = \underbrace{-\frac{1}{2}(\tilde{\mathbf{r}}-\boldsymbol{\mu}_{\boldsymbol{\theta}})^\top(\mathbf{K}_{\boldsymbol{\phi}}+\sigma_\epsilon^2\mathbf{I})^{-1}(\tilde{\mathbf{r}}-\boldsymbol{\mu}_{\boldsymbol{\theta}})}_{\text{data fitting term}} \underbrace{-\frac{1}{2}\log\det(\mathbf{K}_{\boldsymbol{\phi}}+\sigma_\epsilon^2\mathbf{I})}_{\text{Occam factor}} - \frac{|\tau|}{2}\log(2\pi), \tag{11}$$

where the first term ("data fitting") measures how well the model explains the data, and the second term ("Occam factor") penalizes model overfitting. Then, our problem can be formulated as

$$\max_{\boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon} \log p(\tilde{\mathbf{r}} \mid \boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon),$$

or equivalently,

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon} \mathcal{L}(\tau;\boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon) = -\log p(\tilde{\mathbf{r}} \mid \boldsymbol{\theta},\boldsymbol{\phi},\sigma_\epsilon). \tag{12}$$

To optimize this objective, we compute the gradient with respect to each hyperparameter and update the parameter in the gradient descent direction. The detailed gradient derivations and their theoretical implications are presented in the Theoretical Analyses section.

Algorithms

In this section, we present the algorithm of our GP-LRR. Our approach follows an iterative paradigm that alternates between reward modeling via GP regression and policy optimization via SAC using the learned reward signals.

Algorithm 1 details the core GP reward redistribution procedure. For each trajectory batch, we construct LOO targets and optimize the marginal likelihood by performing gradient descent steps. The matrix inverse $\mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}}_j - \boldsymbol{\mu}_j)$ can be computed via Cholesky decomposition for numerical stability and efficiency.

Algorithm 2 then integrates GP-LRR with SAC by using the learned mean function $\mu_{\boldsymbol{\theta}}(s,a)$ as dense reward signals. We maintain two buffer sets: transition buffer $\mathcal{D}$ for SAC updates and complete trajectory buffer $\mathcal{D}_\tau$ for GP training. While GP training has $\mathcal{O}(M|\tau|^3)$ complexity, we amortize this cost by updating the GP model every n_{update} episodes while performing SAC updates more frequently.

Theoretical Analyses

In this section, we analyze the theoretical properties of GP-LRR. We begin by establishing that our framework generalizes existing MSE-based approaches. Then, we demonstrate how the GP structure enables more effective credit assignment through correlation modeling.

Proposition 1 (MSE as a Special Case)**.** *The traditional MSE-based reward redistribution approach emerges as a special case of our GP framework. Specifically, when the kernel matrix reduces to identity ($\mathbf{K}_{\boldsymbol{\phi}} = \mathbf{I}$) and observation noise vanishes ($\sigma_\epsilon = 0$), the objective function for* (12) *becomes*

$$\mathcal{L}(\tau;\boldsymbol{\theta}) \propto \frac{|\tau|}{2}\left(R_{ep}(\tau) - \sum_{t=0}^{|\tau|-1}\mu_{\boldsymbol{\theta}}(s_t,a_t)\right)^2$$

Algorithm 1: Gaussian Process Reward Redistribution

Input: Initialize parameters $\boldsymbol{\theta}, \sigma_f^2, \ell_{\text{rbf}}, \sigma_\epsilon^2$, replay buffer $\mathcal{D}_\tau$, batch size M, learning rate β, gradient steps G
Sample trajectory batch $\{(\tau_m, R_{\text{ep}}(\tau_m))\}_{m=1}^M$ from $\mathcal{D}_\tau$
Initialize $\mathcal{L} = 0$
for $m \in \{1, \dots, M\}$ **do**
  Compute mean vector

$$\boldsymbol{\mu}_m = [\mu_{\boldsymbol{\theta}}(s_0^m, a_0^m), \dots, \mu_{\boldsymbol{\theta}}(s_{|\tau_m|-1}^m, a_{|\tau_m|-1}^m)]^\top$$

  Compute kernel using trajectory τ_m

$$[\mathbf{K}_m]_{ij} = \sigma_f^2 \exp\left(-\frac{\|(s_i^m, a_i^m) - (s_j^m, a_j^m)\|^2}{2\ell_{\text{rbf}}^2}\right)$$

  for $i \in \{0, \dots, |\tau_m| - 1\}$ **do**
    $\tilde{r}(s_i^m, a_i^m) = R_{\text{ep}}(\tau_m) - \sum_{t=0, t\neq i}^{|\tau_m|-1} \mu_{\boldsymbol{\theta}}(s_t^m, a_t^m)$
  end for
  Construct LOO targets $\tilde{\mathbf{r}}_m = [\tilde{r}(s_0^m, a_0^m), \dots, \tilde{r}(s_{|\tau_m|-1}^m, a_{|\tau_m|-1}^m)]^\top$.
  Compute $\mathbf{K}_\sigma = \mathbf{K}_m + \sigma_\epsilon^2 \mathbf{I}$
  Compute $\boldsymbol{\alpha}_m = \mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}}_m - \boldsymbol{\mu}_m)$
  Compute trajectory negative log marginal likelihood

$$\mathcal{L}_m \propto \frac{1}{2}(\tilde{\mathbf{r}}_m - \boldsymbol{\mu}_m)^\top \boldsymbol{\alpha}_m + \frac{1}{2}\log\det(\mathbf{K}_\sigma)$$

  $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_m / M$
end for
for gradient step $= 1, \dots, G$ **do**
  Update the hyperparameters

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}, \quad \sigma_f^2 \leftarrow \sigma_f^2 - \beta \nabla_{\sigma_f^2} \mathcal{L}$$
$$\ell_{\text{rbf}} \leftarrow \ell_{\text{rbf}} - \beta \nabla_{\ell_{\text{rbf}}} \mathcal{L}, \quad \sigma_\epsilon^2 \leftarrow \sigma_\epsilon^2 - \beta \nabla_{\sigma_\epsilon^2} \mathcal{L}$$

end for

This proposition shows that MSE objective (2) corresponds to a degenerate GP with no correlation structure between state-action pairs.

Remark 1. *MSE-based methods assume independent per-step rewards, causing high variance under sparse feedback. By kernelizing correlations, our GP framework pools information across related state–action pairs, yielding smoother and more sample-efficient credit assignment.*

In GP–LRR, per-step errors are not treated in isolation but are aggregated through the precision matrix to form a *temporal credit-assignment network*:

Proposition 2 (Gradient Flow with Correlations)**.** *The gradient of the negative log marginal likelihood $\mathcal{L}(\tau; \boldsymbol{\theta}, \boldsymbol{\phi}, \sigma_\epsilon)$ with respect to the mean function parameters $\boldsymbol{\theta}$ is*

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = - \underbrace{\frac{\partial \boldsymbol{\mu}_{\boldsymbol{\theta}}^\top}{\partial \boldsymbol{\theta}}}_{\text{Neural Network Jacobian}} \cdot \underbrace{\mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})}_{\text{precision-weighted residual}} \tag{13}$$

For a specific state-action pair (s_i, a_i), the contribution to its gradient is formed by the prediction errors from all state-action pairs:

$$[\mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})]_i = w_{ii}(\tilde{r}(s_i, a_i) - \mu_{\boldsymbol{\theta}}(s_i, a_i)) + \sum_{j \neq i} w_{ij}(\tilde{r}(s_j, a_j) - \mu_{\boldsymbol{\theta}}(s_j, a_j)),$$

where $w_{ij} = [\mathbf{K}_\sigma^{-1}]_{ij}$ is the element of the precision matrix $\mathbf{K}_\sigma^{-1} = (\mathbf{K}_{\boldsymbol{\phi}} + \sigma_\epsilon^2 \mathbf{I})^{-1}$.

Remark 2 (Credit Assignment through Correlations)**.** *This decomposition reveals the fundamental difference between GP and MSE approaches. In MSE where $\mathbf{K}_\sigma = \mathbf{I}$, we have $w_{ij} = 0$ for $i \neq j$, so each gradient depends only on its own prediction error. In contrast, the GP formulation creates a* credit assignment network *through the precision matrix: when (s_i, a_i) and (s_j, a_j) are highly correlated, their errors are pooled together during learning. This is particularly powerful for temporal credit assignment—errors at future time steps can directly influence the reward estimates at earlier time steps, addressing the fundamental challenge of learning from sparse episodic feedback.*

Proposition 3 (Length Scale and Smoothness Trade-off)**.** *The gradient of the negative log marginal likelihood $\mathcal{L}(\tau; \boldsymbol{\theta}, \boldsymbol{\phi}, \sigma_\epsilon)$ with respect to the RBF length scale ℓ_{rbf} is*

$$\frac{\partial \mathcal{L}}{\partial \ell_{rbf}} = \frac{1}{2} tr\left(\mathbf{K}_\sigma^{-1} \frac{\partial \mathbf{K}_{\boldsymbol{\phi}}}{\partial \ell_{rbf}}\right) - \frac{1}{2} \boldsymbol{\alpha}^T \frac{\partial \mathbf{K}_{\boldsymbol{\phi}}}{\partial \ell_{rbf}} \boldsymbol{\alpha}, \tag{14}$$

where $tr(A)$ denotes the trace of matrix A, $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})$ and $\left[\frac{\partial \mathbf{K}_{\boldsymbol{\phi}}}{\partial \ell_{rbf}}\right]_{ij} = k_{ij} \frac{d_{ij}^2}{\ell_{rbf}^3}$ with $d_{ij} = \|(s_i, a_i) - (s_j, a_j)\|$, and $k_{ij} = k_\phi((s_i, a_i), (s_j, a_j))$.

Remark 3 (Automatic Smoothness Adaptation)**.** *The gradient in Proposition 3 reveals how the GP framework automatically balances model complexity and data fitting through*

Algorithm 2: SAC with GP-LRR

Initialize: Critic networks $Q_{\omega_1}, Q_{\omega_2}$, actor π_ζ, GP reward model parameters $\boldsymbol{\theta}, \sigma_f^2, \ell_{\text{rbf}}, \sigma_\epsilon^2$, Transition buffer $\mathcal{D} \leftarrow \emptyset$, trajectory buffer $\mathcal{D}_\tau \leftarrow \emptyset$
for episode $= 1, 2, \dots$ **do**
  Collect trajectory $\tau = \{(s_t, a_t)\}_{t=0}^{T-1}$ by following π_ζ
  Store trajectory: $\mathcal{D}_\tau \leftarrow \mathcal{D}_\tau \cup \{(\tau, R_{\text{ep}}(\tau))\}$
  Store transitions: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, s_{t+1}) : t \in [0, T-1]\}$
  if episode $\bmod n_{\text{update}} = 0$ **then**
    Update GP model using Algorithm 1 with batch from $\mathcal{D}_\tau$
  end if
  for gradient step $= 1, \dots, G$ **do**
    Sample minibatch $\mathcal{B} = \{(s_i, a_i, s_i')\}_{i=1}^B \sim \mathcal{D}$
    Compute dense rewards $r_i = \mu_{\boldsymbol{\theta}}(s_i, a_i)$, $i \in [1, B]$
    Compute TD targets following (4) using r_i
    Update critics ω_1, ω_2 by minimizing (3)
    Update actor π_ζ by minimizing (5)
    Update temperature α by minimizing (6)
  end for
end for

the length scale parameter ℓ_{rbf}. Here, a smaller ℓ_{rbf} leads to a more complex model that can fit the data better, while a larger ℓ_{rbf} results in a smoother model that generalizes more. By maximizing marginal likelihood, GP-LRR automatically adjusts ℓ_{rbf}, thus balancing data fitting with model complexity.

Proposition 4 (Observation Noise Adaptation)**.** *The gradient of the negative log marginal likelihood $\mathcal{L}(\tau; \boldsymbol{\theta}, \boldsymbol{\phi}, \sigma_\epsilon)$ with respect to the observation noise variance σ_ϵ^2 is:*

$$\frac{\partial \mathcal{L}}{\partial \sigma_\epsilon^2} = \frac{1}{2} tr(\mathbf{K}_\sigma^{-1}) - \frac{1}{2} \|\boldsymbol{\alpha}\|^2, \tag{15}$$

where $\boldsymbol{\alpha} = \mathbf{K}_\sigma^{-1}(\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})$.

Remark 4 (Automatic Noise Level Selection)**.** *The trace term $tr(\mathbf{K}_\sigma^{-1})$ increases as σ_ϵ^2 decreases, acting as a regularizer that prevents the noise from vanishing. Note that as $\sigma_\epsilon^2 \to 0$, we have $\mathbf{K}_\sigma^{-1} \approx \mathbf{K}_\phi^{-1}$, and the trace can become arbitrarily large for ill-conditioned kernels. The second term $\|\boldsymbol{\alpha}\|^2 = (\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})^T \mathbf{K}_\sigma^{-2} (\tilde{\mathbf{r}} - \boldsymbol{\mu}_{\boldsymbol{\theta}})$ measures the weighted prediction error—when the model cannot explain the LOO targets well, this term becomes large and drives σ_ϵ^2 upward to accommodate the misfit.*

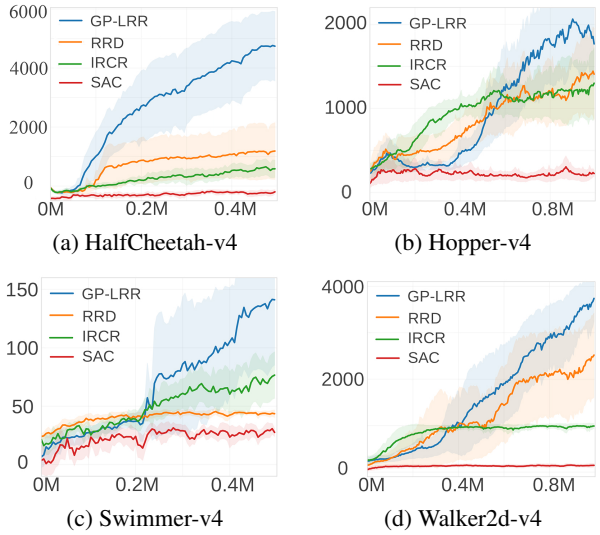
Experiments

In this section, we evaluate our proposed GP-LRR framework on continuous control tasks with delayed episodic rewards. We compare GP-LRR against several baselines across multiple MuJoCo environments to demonstrate its effectiveness in credit assignment and sample efficiency. All experiments were conducted on a single NVIDIA RTX 4080 Super GPU with 16GB memory.

Experimental Setup

We evaluate our algorithm on four continuous control tasks: HalfCheetah-v4, Hopper-v4, Swimmer-v4, and Walker2d-v4. Following prior work (Ren et al. 2021), we modify these environments to provide only episodic rewards, where the agent receives zero reward at all intermediate steps and only observes the cumulative return $R_{\text{ep}}(\tau) = \sum_{t=0}^{T-1} r_t$ at episode termination, which creates a challenging sparse reward setting that tests each method's ability to perform temporal credit assignment.

We compare GP-LRR against three baselines. First, we evaluate vanilla Soft Actor-Critic (SAC) (Haarnoja et al. 2018) using only the sparse episodic reward signal, which serves as a lower bound representing performance without any reward redistribution. Second, we test IRCR (Gangwani, Zhou, and Peng 2020), which uniformly redistributes the episodic return across all timesteps by assigning $r_t = R_{\text{ep}}(\tau)/T$ to each state-action pair. Third, we compare against RRD (Ren et al. 2021), the state-of-the-art randomized return decomposition method that approximates full trajectory decomposition through random subsequence sampling. We use RRD's recommended configuration with sampling parameter $K = 64$, which achieved the best performance in their experiments. Note that we do not compare GP-LRR with RUDDER, as RRD has been shown to consistently outperform RUDDER in (Ren et al. 2021).



(a) HalfCheetah-v4 (b) Hopper-v4

(c) Swimmer-v4 (d) Walker2d-v4

Figure 2: Learning curves on multiple MuJoCo environments, with the x-axis showing timesteps and the y-axis denoting total reward. Solid curves show mean returns over 5 runs; shaded regions denote standard deviation. Performance is evaluated every 5,000 steps. GP-LRR uses the RBF kernel in these experiments.

All methods use SAC as the base RL algorithm with identical hyperparameters. For GP-LRR, the kernel hyperparameters including length scale ℓ_{rbf}, signal variance σ_f^2, and noise variance σ_ϵ^2 are learned jointly via gradient descent on the marginal likelihood. We update the GP reward model every 100 environment steps using trajectory batches of size 4.

Comparison with Baselines

Figure 2 presents the learning curves comparing GP-LRR against the three baselines across four representative environments.[1] We report the mean and standard deviation with random initialization over 5 independent simulation runs. For GP-LRR method, we adopted the RBF kernel.

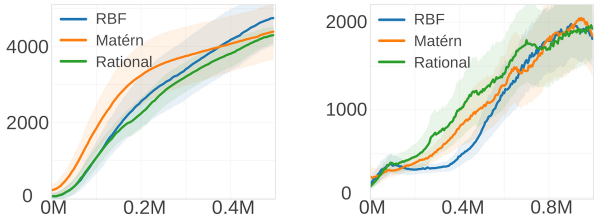
GP-LRR consistently outperforms all baselines across the evaluated environments, with particularly striking improvements in HalfCheetah-v4 where the smooth dynamics and spatially correlated rewards align well with the GP's principled correlation modeling. Unlike methods requiring on-policy samples (Guo et al. 2018; Zheng, Oh, and Singh 2018), GP-LRR seamlessly integrates with off-policy algorithms like SAC, further improving sample efficiency.

Ablation Study

To understand the key components of GP-LRR, we conduct ablation studies examining the effect of kernel choice and length scale initialization on performance.

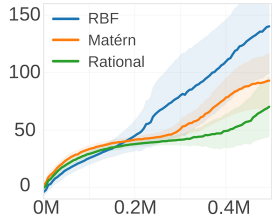
[1]Note that we only evaluate these four environments since not all MuJoCo environments are suitable for modeling state-action pair correlations.

Effect of Kernel Choice In GP-LRR, the kernel choice shapes reward interpolation: (i) the RBF kernel yields infinitely smooth, Gaussian-weighted correlations ideal for gently varying reward landscapes, (ii) the Matérn–3/2 kernel produces once-differentiable, rougher paths that handle abrupt changes, and (iii) the Rational Quadratic kernel—an infinite mixture of RBFs with heavy-tailed scales—adaptively captures both fine details and broad trends.

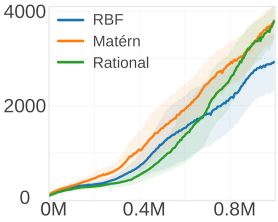


(a) HalfCheetah-v4

(b) Hopper-v4



(c) Swimmer-v4



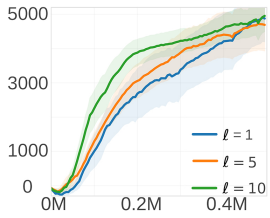
(d) Walker2d-v4

Figure 3: Learning curves on different MuJoCo environments with sparse episodic rewards using the GP-LRR method with various kernels, with the x-axis showing timesteps and the y-axis indicating total return. Solid curves show mean returns over 5 runs; shaded regions denote standard deviation. Performance is evaluated every 5,000 steps.

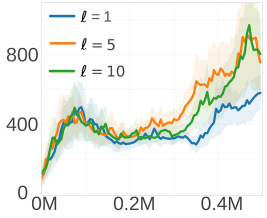
The results in Figure 3 reveal that kernel performance is task-dependent. While RBF excels in HalfCheetah, Hopper, and Swimmer—environments with smooth reward landscapes—it performs worst in Walker2d, where Matérn and Rational Quadratic kernels achieve superior results. This suggests that Walker2d's reward structure may contain discontinuities or sharp transitions that the infinitely differentiable RBF kernel cannot capture effectively. The Matérn kernel's ability to model less smooth functions proves advantageous in this more complex balancing task.

Impact of Length Scale Initialization The length scale parameter ℓ_{rbf} controls how far correlations extend in the state-action space. Small values create localized credit assignment where only nearby states influence each other, while large values enable global information sharing across distant states. This parameter essentially determines whether the GP assumes rewards change rapidly (small ℓ_{rbf}) or smoothly (large ℓ_{rbf}) across the state space.

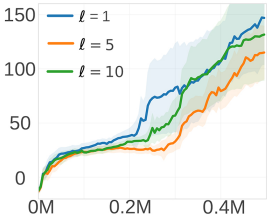
We test four *initialization* strategies: small ($\ell = 0.1$) for environments with sharp reward transitions, medium ($\ell = 1.0$) as a balanced default, large ($\ell = 10.0$) for smooth reward landscapes, and adaptive initialization based on median pairwise distances in early trajectories.



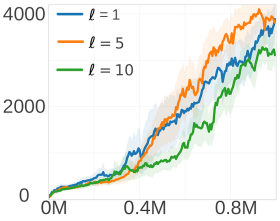
(a) HalfCheetah-v4



(b) Hopper-v4



(c) Swimmer-v4



(d) Walker2d-v4

Figure 4: Learning curves on different MuJoCo environments with sparse episodic rewards using the GP-LRR method under varying ℓ_{rbf} initialization, with the x-axis showing timesteps and the y-axis indicating total return. Solid curves show mean returns over 5 runs; shaded regions denote standard deviation. Performance is evaluated every 5,000 steps.

The results in Figure 4 show no universal optimal length scale initialization—each environment responds differently. HalfCheetah prefers large values ($\ell = 10$), Walker2d favors medium scales, while Hopper and Swimmer show high variance or insensitivity to initialization. This environment-specific behavior confirms that correlation structure depends heavily on task dynamics, with no one-size-fits-all solution despite GP-LRR's adaptive optimization.

Conclusion

We introduced a novel likelihood-based framework (GP-LRR) for reward redistribution in RL with episodic returns, where we modeled the per-step reward as a sample from a GP and maximized the likelihood of a trajectory via an LOO strategy. Unlike other conventional reward decomposition methods, our GP-LRR explicitly incorporates interdependencies among state-action pairs using the kernel function. Our theoretical analyses illustrated that GP-LRR includes the conventional MSE-based approach as a special case. We further showed that it pools gradient information from different state-action pairs together and implicitly incorporates an uncertainty regularizer to ensure more efficient parameter updating. Empirical results on MuJoCo benchmarks demonstrated that, when combined with SAC, our method yields dense, robust reward signals that improve sample efficiency and final policy performance. In future work, we plan to extend this framework to non-Gaussian noise models and investigate integrations with other off-policy algorithms.

Acknowledgments

This work was supported in part by the National Science Foundation under Grant #2331782.

References

Arjona-Medina, J. A.; Gillhofer, M.; Widrich, M.; Unterthiner, T.; Brandstetter, J.; and Hochreiter, S. 2019. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chung, J. J.; Lawrance, N. R.; and Sukkarieh, S. 2013. Gaussian processes for informative exploration in reinforcement learning. In *2013 IEEE International Conference on Robotics and Automation*, 2633–2639. IEEE.

Deisenroth, M.; and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, 465–472.

Duvenaud, D. 2014. The kernel cookbook: Advice on covariance functions. *URL https://www. cs. toronto. edu/duvenaud/cookbook*.

Efroni, Y.; et al. 2021. Return Decomposition for Delayed Rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Engel, Y.; Mannor, S.; and Meir, R. 2005. Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, 201–208.

Gangwani, T.; Zhou, Y.; and Peng, J. 2020. Learning guidance rewards with trajectory-space smoothing. *Advances in Neural Information Processing Systems*, 33: 822–832.

Guo, Y.; Oh, J.; Singh, S.; and Lee, H. 2018. Generative adversarial self-imitation learning. *arXiv preprint arXiv:1812.00950*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 1861–1870. PMLR.

Hambly, B.; Xu, R.; and Yang, H. 2023. Recent advances in reinforcement learning in finance. *Mathematical Finance*, 33(3): 437–503.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6): 4909–4926.

Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274.

Lin, H.; Wu, H.; Zhang, J.; Sun, Y.; Ye, J.; and Yu, Y. 2024. Episodic return decomposition by difference of implicitly assigned sub-trajectory reward. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 13808–13816.

Razzaghi, P.; Tabrizian, A.; Guo, W.; Chen, S.; Taye, A.; Thompson, E.; Bregeon, A.; Baheri, A.; and Wei, P. 2024. A survey on reinforcement learning in aviation applications. *Engineering Applications of Artificial Intelligence*, 136: 108911.

Ren, Z.; Guo, R.; Zhou, Y.; and Peng, J. 2021. Learning long-term reward redistribution via randomized return decomposition. *arXiv preprint arXiv:2111.13485*.

Wei, H.; Liu, X.; and Ying, L. 2024. Safe reinforcement learning with instantaneous constraints: The role of aggressive exploration. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 21708–21716.

Yu, C.; Liu, J.; Nemati, S.; and Yin, G. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, 55(1): 1–36.

Zhang, Y.; Du, Y.; Huang, B.; Wang, Z.; Wang, J.; Fang, M.; and Pechenizkiy, M. 2023. Interpretable reward redistribution in reinforcement learning: A causal approach. *Advances in Neural Information Processing Systems*, 36: 20208–20229.

Zheng, Z.; Oh, J.; and Singh, S. 2018. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31.