SUPPLEMENTARY MATERIALS OF BEYOND SIMPLE SUM OF DELAYED REWARDS: NON-MARKOVIAN REWARD MODELING FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

A IMPLEMENTATION DETAILS

A.1 BENCHMARKS WITH COMPOSITE DELAYED REWARD

In this work, we introduced a new problem setting, called composite delayed rewards, within the suite of locomotion benchmark tasks in both the MuJoCo environment and the DeepMind Control Suite. Our experiments were conducted using the OpenAI Gym platform (Brockman et al., 2016) and the DeepMind Control Suite (Tassa et al., 2018), focusing on tasks with extended horizons and a fixed maximum trajectory length of $T = 1000$. We utilized MuJoCo version 2.0 for our simulations, which can be accessed at `http://www.mujoco.org/`. MuJoCo operates under a commercial license, and we ensured full compliance with its licensing terms. Additionally, the DeepMind Control Suite, distributed under the Apache License 2.0, was used in accordance with its licensing requirements.

Experiments involving composite delayed rewards with varying delay steps (5, 25, 50, 100, 200, and 500) and different composite types (SumSquare, SquareSum, Max, and the traditional Sum) were conducted to validate the effectiveness of the proposed method. In the Max experiment, the scaling parameter β is set to 3. To evaluate its performance, commonly used delayed reward algorithms were adapted to fit within the composite delayed reward framework, acting as baselines. In these experiments, each segment with composite delayed rewards was treated as an independent trajectory, and the modified algorithms were applied accordingly.

A.2 IMPLEMENTATION DETAILS AND HYPER-PARAMETER CONFIGURATION

In our experiments, the policy optimization module was implemented based on soft actor-critic (SAC) (Haarnoja et al., 2018). We evaluated the performance of our proposed methods with the same configuration of hyper-parameters in all environments. The back-end SAC followed the JaxRL implementation (Kostrikov, 2021), which is available under the MIT License.

The proposed CoDeTr was built upon the GPT implementation in JAX (Frostig et al., 2018), available under the Apache License 2.0. Our experiments employed a Causal Transformer with three layers and four self-attention heads, followed by an in-sequence bidirectional attention layer with one self-attention head. For a comprehensive overview of the CoDeTr's hyper-parameter settings, please refer to Table 1.

For the baseline methods, the IRCR (Gangwani et al., 2020) method was implemented following the descriptions provided in the original paper. Both the RRD (Ren et al., 2021) and LIRPG (Zheng et al., 2018) methods are distributed under the MIT License. The code for HC (Han et al., 2022) is available in the supplementary material at `https://openreview.net/forum?id=nsjkNB2oKsQ`, while the code for RBT (Tang et al., 2024) can be found in the original paper.

To maintain consistency in the policy optimization process across all methods, each was subjected to 1,000,000 training iterations. For the proposed method, a dataset of 10,000 time steps was first gathered to pre-train the reward model. This model underwent 100 pre-training iterations, which was deemed necessary to properly initialize the reward model before commencing the main policy learning phase. After this warm-up period, the reward model was updated for 10 iterations following

Table 1: Hyper-parameters of CoDeTr.

Hyper-parameter	Value
Number of Causal Transformer layers	3
Number of in-sequence attention layers	1
Number of attention heads	4
Embedding dimension	256
Batch size	64
Dropout rate	0.1
Learning rate	0.00005
Optimizer	AdamW (Loshchilov & Hutter, 2018)
Weight decay	0.0001
Warmup steps	100
Total gradient steps	10000

the addition of each new trajectory. Furthermore, to monitor performance systematically, evaluations were conducted every 5,000 time steps. During prediction, the sequence length used for prediction is set to $H = 100$. All computations were performed on NVIDIA GeForce A100 GPUs with 40GB of memory, which were dedicated to both training and evaluation tasks.

A.3 Data Normalization Procedures

The normalization process for our data varies depending on the type of composite delayed reward. Specifically:

- For **SumSquare**, the normalization is calculated as:

$$\frac{\sum \hat{R}_{\text{co}}}{T \cdot \sum (r_{\max})^2}.$$

- For **SquareSum**, the normalization is given by:

$$\frac{\sum \hat{R}_{\text{co}}}{\sum \left(\frac{T}{n} \cdot (r_{\max} \cdot n)^2\right)}.$$

- For **Max**, the normalization is computed as:

$$\frac{\sum \hat{R}_{\text{co}}}{\sum r_{\max}}.$$

Here, $\hat{R}_{\text{co}}$ represents the predicted composite delayed reward, T is the total number of time steps in a trajectory, $r_{\max}$ is the maximum possible reward in the environment, and n is the number of delayed steps in each segment.

In essence, the normalization process involves scaling $\sum R_{\text{co}}$ by the maximum achievable reward in the given environment. This approach ensures that results from experiments with different delayed steps are on the same scale, enabling meaningful comparisons across varying delayed steps and their impact on the learning process.

B Algorithm

The training process involves alternating between updating the reward model and optimizing the policy, which creates a continuous loop of mutual improvement. First, the agent collects trajectories by interacting with the environment according to the current policy. These trajectories are then used to train the CoDeTr, which learns to predict instance-level rewards and composite delayed rewards for sequences. The training is done by minimizing the mean squared error (MSE) loss between the predicted composite reward $R_{\text{co}}(\tau)$ and the observed composite delayed reward $\hat{R}_{\text{co}}(\tau)$. This

Algorithm 1 Policy Optimization with CoDeTr

Initialize: replay buffer $\mathcal{D}$, CoDeTr parameters ψ, and policy π.
while training is not complete **do**
  Collect a trajectory $\mathcal{T}$ by interacting with the environment using the current policy π.
  Store trajectory $\mathcal{T}$ with composite delayed reward information based on sequences $\{(\tau, R_{\text{co}}(\tau))\}$ in replay buffer $\mathcal{D}$..
  Sample batches from replay buffer $\mathcal{D}$.
  Compute the mean squared error loss $(R_{\text{co}}(\tau) - \hat{R}_{\text{co}}(\tau))^2$ for CoDeTr using the sampled sequences from the replay buffer.
  Update CoDeTr parameters ψ based on the computed loss.
  Relabel instance-level rewards in replay buffer $\mathcal{D}$ using the updated CoDeTr.
  Optimize policy π using the relabeled data with an off-the-shelf RL algorithm (e.g., SAC (Haarnoja et al., 2018)).
end while

loss function allows CoDeTr to accurately capture the relationships and dependencies within each sequence, ensuring that both individual and sequence-level contributions are effectively represented. Using the updated CoDeTr model, the rewards for state-action pairs in replay buffer are relabeled, providing more accurate feedback for policy optimization. The updated rewards are used to further refine the policy using reinforcement learning algorithms like SAC, enabling the agent to learn effective strategies even in environments with delayed rewards. This iterative procedure enhances both the reward model and the policy through each training cycle.

C Additional Result

Table 2: Performance comparison across different settings, utilizing various delayed steps ranging from 25 to 200 in the Ant-v2 environment, evaluated over 3 independent trials. The scores presented are normalized to ensure comparability across different configurations. The methods that demonstrated the best performance, along with those that were statistically comparable based on a paired t-test at a significance level of 5%, are highlighted in boldface for emphasis.

Delayed Type	SAC	LIRPG	HC	IRCR	RRD	RBT	CoDeTr(ours)
Sum	0.0004 (0.0002)	−0.1759 (0.0631)	0.0025 (0.0058)	0.03364 (0.0281)	0.3327 (0.2095)	**0.5699** (**0.0162**)	**0.5493** (**0.0187**)
SumSquare	−0.0067 (0.0022)	−0.0159 (0.0027)	0.0198 (0.0005)	−0.0617 (0.0273)	0.0308 (0.0207)	0.2821 (0.0703)	**0.3910** (**0.0431**)
SquareSum	−0.0902 (0.1031)	−0.0012 (0.0001)	−0.0280 (0.0275)	−0.1060 (0.0303)	0.0575 (0.0173)	0.0890 (0.0240)	**0.1992** (**0.0110**)
Max	0.04093 (0.0065)	−0.1982 (0.0373)	0.0108 (0.0103)	−0.0093 (0.0524)	0.2193 (0.0416)	0.4669 (0.1078)	**0.5318** (**0.0821**)

In Table 2, the performance of different methods is compared across various composite delayed reward types: Sum, SumSquare, SquareSum, and Max, on the Ant-v2 environment. Our proposed method, CoDeTr, consistently performed well across all composite delayed reward configurations, either achieving the best results or performing comparably to the top baseline methods. In particular, CoDeTr showed strong performance under different composite delayed reward settings, demonstrating its ability to handle complex reward structures effectively. These results indicate that our approach is robust and adaptable, providing high-quality performance across a range of composite delayed reward scenarios.

D DISCUSSION

Limitation. Our experimental results demonstrate that the proposed approach performs effectively across various types of composite delayed rewards, showing notable improvements over baseline methods. However, we also observed increasing difficulty in efficiently learning the policy as the delay length grew longer. This challenge arises because longer delays weaken the temporal connection between specific actions and their resulting outcomes, increasing uncertainty when attempting to determine which actions contributed to the observed reward. Consequently, the diminished ability to effectively assign credit to individual actions complicates the policy training process, leading to slower convergence and reduced overall performance in scenarios with extended delay lengths, as evidenced in our experiments.

Future Direction. A key area for future work lies in addressing the challenges posed by longer reward delays. Our experiments have shown that increasing the delay length significantly complicates credit assignment to individual actions. To better capture long-range dependencies in such settings, future research could focus on developing advanced temporal credit assignment methods, such as improved attention mechanisms or memory-augmented neural networks. These techniques may enhance the model's ability to trace rewards back to responsible actions, even in situations with extended delays.

Expanding the use of composite delayed rewards to broader application scenarios represents another promising direction. Domains such as healthcare, autonomous vehicles, and industrial robotics often involve delayed and complex feedback that makes instance-level rewards impractical. Investigating how our proposed approach can generalize to these real-world applications would demonstrate its practical utility and robustness. Moreover, such exploration could help identify potential modifications required to adapt the framework to specific challenges, such as safety and real-time requirements inherent in these domains.

In addition, integrating human-in-the-loop feedback with composite delayed rewards could significantly enhance the learning process. Human evaluators often assign feedback based on pivotal events and use non-linear reasoning, which traditional reward models may fail to capture. Incorporating human feedback more directly, possibly through preference learning models aligned with composite delayed rewards, could improve the agent's ability to learn behaviors that align with human expectations. This approach would be particularly valuable in interactive environments where understanding human intent is crucial for the agent's success.

REFERENCES

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. *Systems for Machine Learning*, 4(9), 2018.

Tanmay Gangwani, Yuan Zhou, and Jian Peng. Learning guidance rewards with trajectory-space smoothing. In *The Thirty-third Annual Conference on Advances in Neural Information Processing Systems*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *The Thirty-fifth International Conference on Machine Learning*. PMLR, 2018.

Beining Han, Zhizhou Ren, Zuofan Wu, Yuan Zhou, and Jian Peng. Off-policy reinforcement learning with delayed rewards. In *The Thirty-ninth International Conference on Machine Learning*. PMLR, 2022.

Ilya Kostrikov. JAXRL: Implementations of Reinforcement Learning algorithms in JAX, 10 2021. URL `https://github.com/ikostrikov/jaxrl`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *The Sixth International Conference on Learning Representations*, 2018.

Zhizhou Ren, Ruihan Guo, Yuan Zhou, and Jian Peng. Learning long-term reward redistribution via randomized return decomposition. In *The Ninth International Conference on Learning Representations*, 2021.

Yuting Tang, Xin-Qiang Cai, Yao-Xiang Ding, Qiyu Wu, Guoqing Liu, and Masashi Sugiyama. Reinforcement learning from bagged reward. In *ICML 2024 Workshop: Aligning Reinforcement Learning Experimentalists and Theorists*, 2024.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite, 2018.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *The Thirty-first Annual Conference on Advances in Neural Information Processing Systems*, 2018.