Appendix

A. κ-PI-DQN and κ-VI-DQN Algorithms

A.1. Detailed Pseudo-codes

In this section, we report the detailed pseudo-codes of κ-PI-DQN and κ-VI-DQN algorithms, described in Section 4.3, side-by-side.

Algorithm 5 κ-PI-DQN

Initialize replay buffer $\mathcal{D}$; Q-networks Q_θ and Q_ϕ with random weights θ and ϕ;
Initialize target networks Q'_θ and Q'_ϕ with weights $\theta' \leftarrow \theta$ and $\phi' \leftarrow \phi$;
for $i = 0, \dots, N_\kappa - 1$ **do**
    # Policy Improvement
    for $t = 1, \dots, T_\kappa$ **do**
        Select a_t as an ϵ-greedy action w.r.t. $Q_\theta(s_t, a)$;
        Execute a_t, observe r_t and s_{t+1}, and store the tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
        Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
        Update θ by minimizing the following loss function:
            $\mathcal{L}_{\mathcal{Q}_\theta} = \frac{1}{N}\sum_{j=1}^N \big[Q_\theta(s_j, a_j) - \big(r_j(\kappa, V_\phi) + \gamma\kappa \max_a Q'_\theta(s_{j+1}, a)\big)\big]^2$, where
            $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi_{i-1}(s_{j+1}))$ and $\pi_{i-1}(s_{j+1}) \in \arg\max_a Q'_\theta(s_{j+1}, a)$;
        Copy θ to θ' occasionally $(\theta' \leftarrow \theta)$;
    end for
    # Policy Evaluation
    Set $\pi_i(s) \in \arg\max_a Q'_\theta(s, a)$;
    for $t' = 1, \dots, T(\kappa)$ **do**
        Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
        Update ϕ by minimizing the following loss function:
            $\mathcal{L}_{\mathcal{Q}_\phi} = \frac{1}{N}\sum_{j=1}^N \big[Q_\phi(s_j, a_j) - (r_j + \gamma Q'_\phi(s_{j+1}, \pi_i(s_{j+1})))\big]^2$;
        Copy ϕ to ϕ' occasionally $(\phi' \leftarrow \phi)$;
    end for
end for

Algorithm 6 κ-VI-DQN

Initialize replay buffer $\mathcal{D}$; Q-networks Q_θ and Q_ϕ with random weights θ and ϕ;
Initialize target network Q'_θ with weights $\theta' \leftarrow \theta$;
for $i = 0, \dots, N_\kappa - 1$ **do**
    # Evaluate $T_\kappa V_\phi$ and the κ-greedy policy w.r.t. V_ϕ
    for $t = 1, \dots, T_\kappa$ **do**
        Select a_t as an ϵ-greedy action w.r.t. $Q_\theta(s_t, a)$;
        Execute a_t, observe r_t and s_{t+1}, and store the tuple (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
        Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
        Update θ by minimizing the following loss function:
            $\mathcal{L}_{\mathcal{Q}_\theta} = \frac{1}{N}\sum_{j=1}^N \big[Q_\theta(s_j, a_j) - (r_j(\kappa, V_\phi) + \kappa\gamma \max_a Q'_\theta(s_{j+1}, a))\big]^2$, where
            $V_\phi(s_{j+1}) = Q_\phi(s_{j+1}, \pi(s_{j+1}))$ and $\pi(s_{j+1}) \in \arg\max_a Q_\phi(s_{j+1}, a)$;
        Copy θ to θ' occasionally $(\theta' \leftarrow \theta)$;
    end for
    Copy θ to ϕ $(\phi \leftarrow \theta)$
end for

Hyperparameter	Value
Horizon (T)	1000
Adam stepsize	1×10^{-4}
Target network update frequency	1000
Replay memory size	100000
Discount factor	0.99
Total training time steps	10000000
Minibatch size	32
Initial exploration	1
Final exploration	0.1
Final exploration frame	1000000
#Runs used for plot averages	10
Confidence interval for plot runs	$\sim 95\%$

Table 3: Hyperparameters for κ-PI-DQN and κ-VI-DQN.

A.2. Ablation Test for C_{FA}

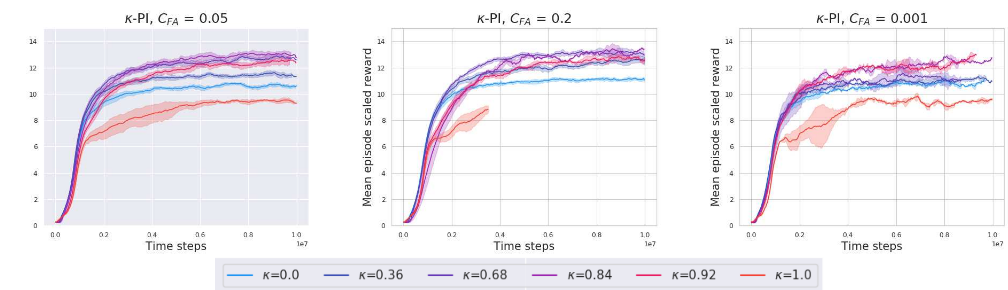

Figure 4: Performance of κ-PI-DQN and κ-VI-DQN on Breakout for different values of C_{FA}.

A.3. κ-PI-DQN and κ-VI-DQN Plots

In this section, we report additional results of the application of κ-PI-DQN and κ-VI-DQN on the Atari domains. A summary of these results has been reported in Table 1 in the main paper.

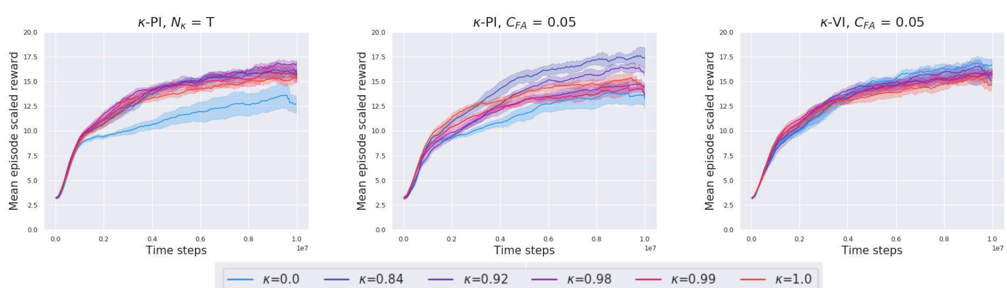

Figure 5: Training performance of the 'naive' baseline $N_\kappa = T$ and κ-PI-DQN, κ-VI-DQN for $C_{\text{FA}} = 0.05$ on SpaceInvaders

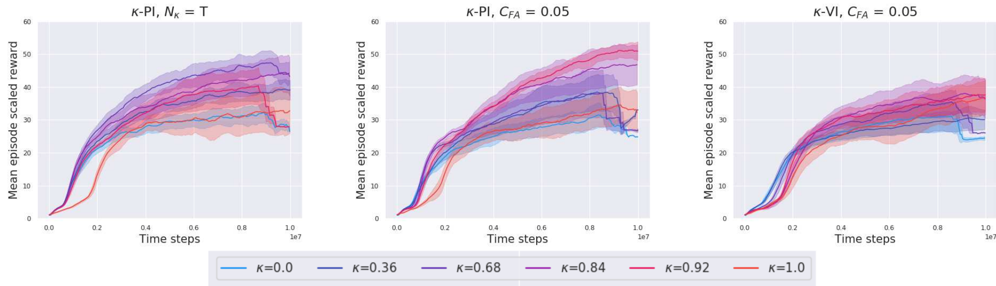

Figure 6: Training performance of the ‘naive’ baseline $N_\kappa = T$ and κ-PI-DQN, κ-VI-DQN for $C_{\text{FA}} = 0.05$ on Seaquest

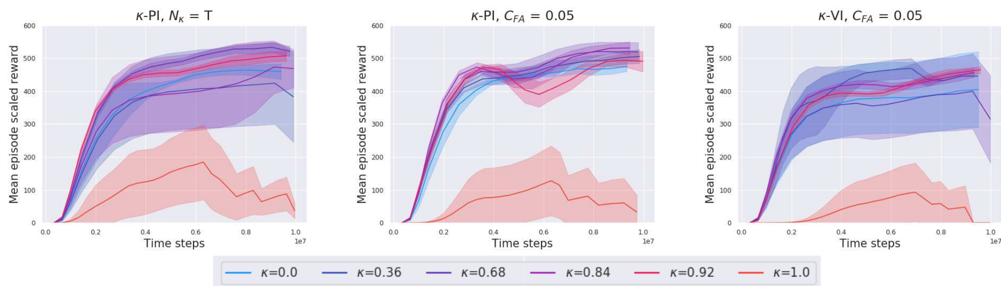

Figure 7: Training performance of the ‘naive’ baseline $N_\kappa = T$ and κ-PI-DQN, κ-VI-DQN for $C_{\text{FA}} = 0.05$ on Enduro

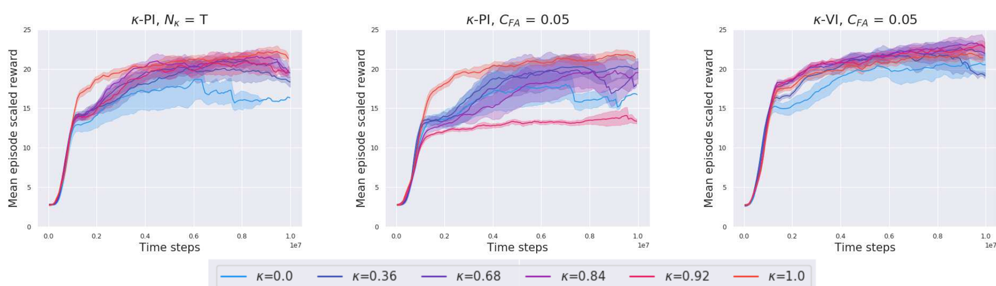

Figure 8: Training performance of the ‘naive’ baseline $N_\kappa = T$ and κ-PI-DQN, κ-VI-DQN for $C_{\text{FA}} = 0.05$ on BeamRider

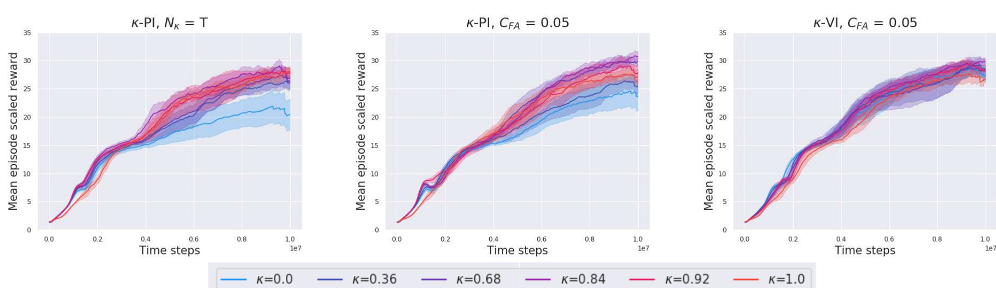

Figure 9: Training performance of the ‘naive’ baseline $N_\kappa = T$ and κ-PI-DQN, κ-VI-DQN for $C_{\text{FA}} = 0.05$ on Qbert

B. κ-PI-TRPO and κ-VI-TRPO Algorithms

B.1. Detailed Pseudo-codes

In this section, we report the detailed pseudo-codes of the κ-PI-TRPO and κ-VI-TRPO algorithms, described in Section 4.4, side-by-side.

Algorithm 7 κ-PI-TRPO

Initialize V-networks V_θ and V_ϕ with random weights θ and ϕ; policy network π_ψ with random weights ψ;
for $i = 0, \ldots, N_\kappa - 1$ **do**
  for $t = 1, \ldots, T_\kappa$ **do**
    Simulate the current policy π_ψ for M time-steps;
    for $j = 1, \ldots, M$ **do**
      Calculate $R_j(\kappa, V_\phi) = \sum_{t=j}^{M} (\gamma\kappa)^{t-j} r_t(\kappa, V_\phi)$ and $\rho_j = \sum_{t=j}^{M} \gamma^{t-j} r_t$;
    end for
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update θ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V_\phi))^2$;
    # Policy Improvement
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update ψ using TRPO with advantage function computed by $\{(R_j(\kappa, V_\phi), V_\theta(s_j))\}_{j=1}^{N}$;
  end for
  # Policy Evaluation
  Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
  Update ϕ by minimizing the loss function: $\mathcal{L}_{V_\phi} = \frac{1}{N} \sum_{j=1}^{N} (V_\phi(s_j) - \rho_j)^2$;
end for

Algorithm 8 κ-VI-TRPO

Initialize V-networks V_θ and V_ϕ with random weights θ and ϕ; policy network π_ψ with random weights ψ;
for $i = 0, \ldots, N_\kappa - 1$ **do**
  # Evaluate $T_\kappa V_\phi$ and the κ-greedy policy w.r.t. V_ϕ
  for $t = 1, \ldots, T_\kappa$ **do**
    Simulate the current policy π_ψ for M time-steps;
    for $j = 1, \ldots, M$ **do**
      Calculate $R_j(\kappa, V_\phi) = \sum_{t=j}^{M} (\gamma\kappa)^{t-j} r_t(\kappa, V_\phi)$;
    end for
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update θ by minimizing the loss function: $\mathcal{L}_{V_\theta} = \frac{1}{N} \sum_{j=1}^{N} (V_\theta(s_j) - R_j(\kappa, V_\phi))^2$;
    Sample a random mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{j=1}^{N}$ from the simulated M time-steps;
    Update ψ using TRPO with advantage function computed by $\{(R_j(\kappa, V_\phi), V_\theta(s_j))\}_{j=1}^{N}$;
  end for
  Copy θ to ϕ $(\phi \leftarrow \theta)$;
end for

Hyperparameter	Value
Horizon (T)	1000
Adam stepsize	1×10^{-3}
Number of samples per Iteration	1024
Entropy coefficient	0.01
Discount factor	0.99
Number of Iterations	2000
Minibatch size	128
#Runs used for plot averages	10
Confidence interval for plot runs	$\sim 95\%$

Table 4: Hyper-parameters of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains.

B.2. Ablation Test for C_{FA}

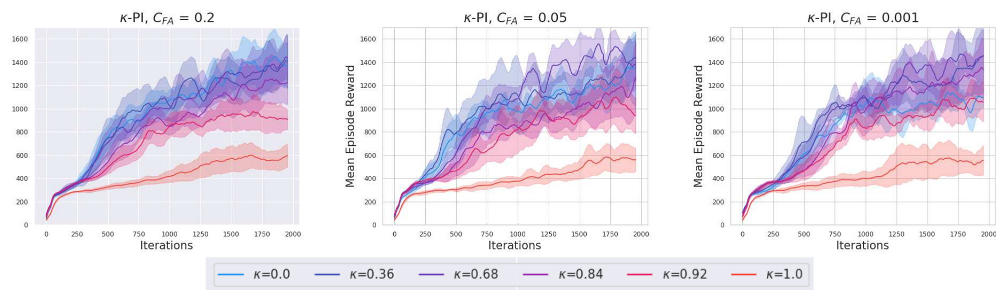

Figure 10: Performance of κ-PI-TRPO and κ-VI-TRPO on Walker2d-v2 for different values of C_{FA}.

B.3. κ-PI-TRPO and κ-VI-TRPO Plots

In this section, we report additional results of the application of κ-PI-TRPO and κ-VI-TRPO on the MuJoCo domains. A summary of these results has been reported in Table 2 in the main paper.

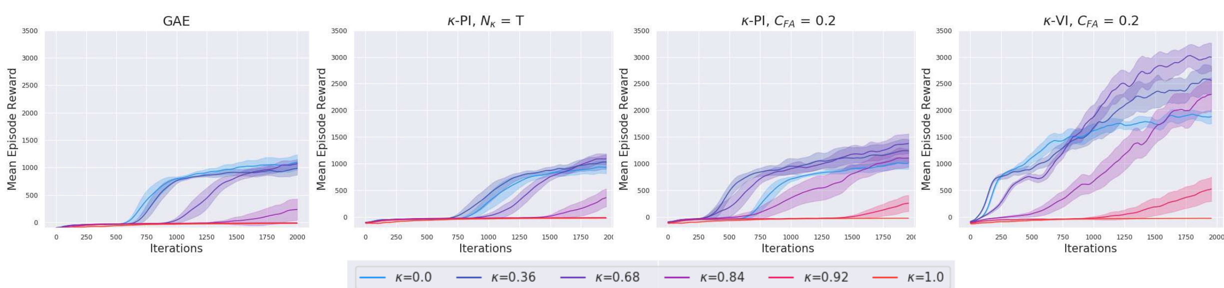

Figure 11: Performance of GAE, 'Naive' baseline and κ-PI-TRPO, κ-VI-TRPO on Ant-v2.

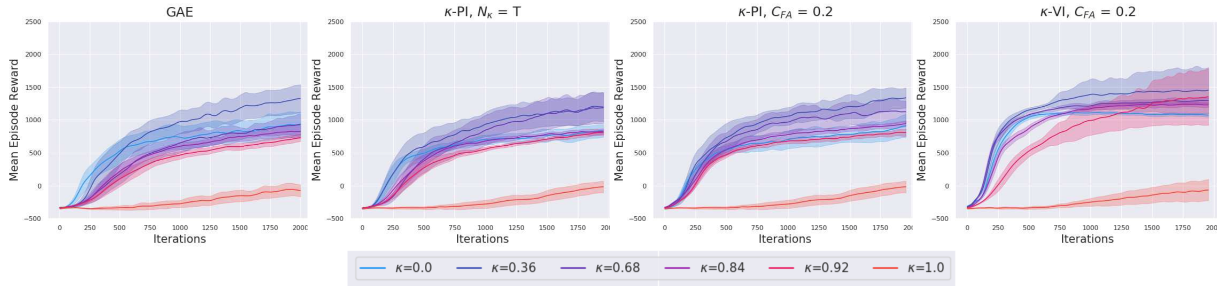

Figure 12: Performance of GAE, ‘Naive’ baseline and κ-PI-TRPO, κ-VI-TRPO on HalfCheetah-v2.

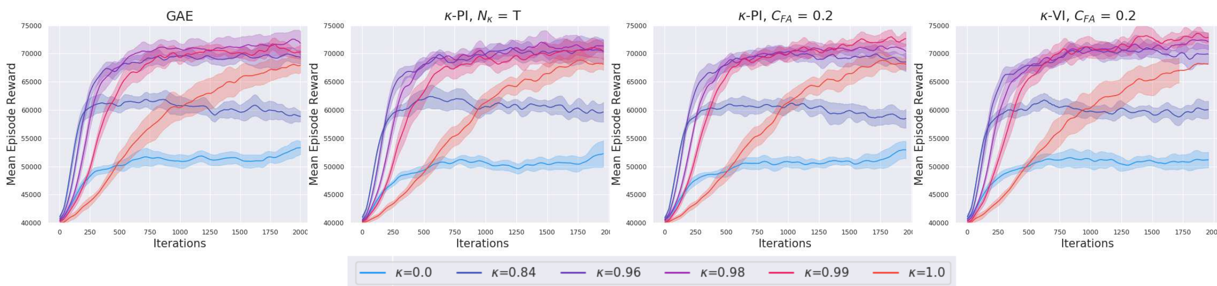

Figure 13: Performance of GAE, ‘Naive’ baseline and κ-PI-TRPO, κ-VI-TRPO on HumanoidStandup-v2.

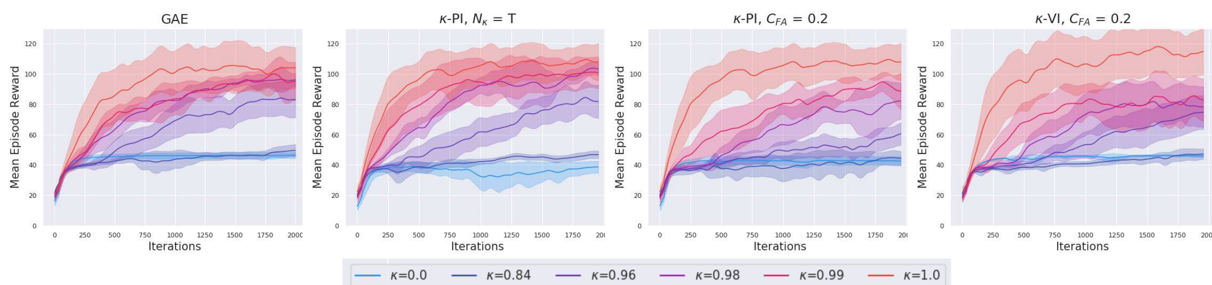

Figure 14: Performance of GAE, ‘Naive’ baseline and κ-PI-TRPO, κ-VI-TRPO on Swimmer-v2.

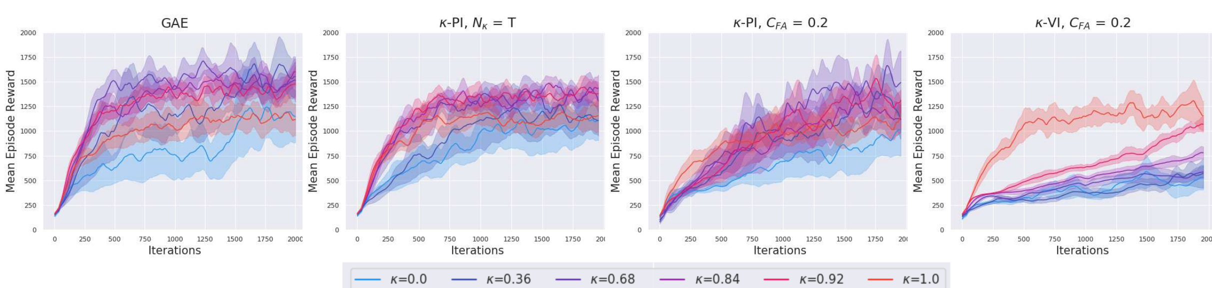

Figure 15: Performance of GAE, ‘Naive’ baseline and κ-PI-TRPO, κ-VI-TRPO on Hopper-v2.