Supervised Off-Policy Ranking

Yue Jin[1] **Yue Zhang**[2] **Tao Qin**[3] **Xudong Zhang**[1] **Jian Yuan**[1] **Houqiang Li**[2] **Tie-Yan Liu**[3]

Abstract

Off-policy evaluation (OPE) is to evaluate a target policy with data generated by other policies. Most previous OPE methods focus on precisely estimating the true performance of a policy. We observe that in many applications, (1) the end goal of OPE is to compare two or multiple candidate policies and choose a good one, which is a much simpler task than precisely evaluating their true performance; and (2) there are usually multiple policies that have been deployed to serve users in real-world systems and thus the true performance of these policies can be known. Inspired by the two observations, in this work, we study a new problem, supervised off-policy ranking (SOPR), which aims to rank a set of target policies based on supervised learning by leveraging off-policy data and policies with known performance. We propose a method to solve SOPR, which learns a policy scoring model by minimizing a ranking loss of the training policies rather than estimating the precise policy performance. The scoring model in our method, a hierarchical Transformer based model, maps a set of state-action pairs to a score, where the state of each pair comes from the off-policy data and the action is taken by a target policy on the state in an offline manner. Extensive experiments on public datasets show that our method outperforms baseline methods in terms of rank correlation, regret value, and stability. Our code is publicly available at GitHub [1].

1. Introduction

Off-policy evaluation (OPE), which aims to estimate the online/true performance [2] of a policy using only pre-collected historical data generated by other policies, is critical to many real-world applications, in which evaluating or deploying a poorly performed policy online might be prohibitively expensive (e.g., in trading, advertising, traffic control) or even dangerous (e.g., in robotics, autonomous vehicles, drug trials).

Existing OPE methods can be roughly categorized into three classes: distribution correction based methods (Thomas et al., 2015; Liu et al., 2018; Hanna et al., 2019; Xie et al., 2019; Nachum et al., 2019; Zhang et al., 2020; Nachum & Dai, 2020; Yang et al., 2020; Kostrikov & Nachum, 2020), model estimation based methods (Mannor et al., 2004; Thomas & Brunskill, 2016; Hanna et al., 2017; Zhang et al., 2021), and Q-estimation based methods (Le et al., 2019; Munos et al., 2016; Harutyunyan et al., 2016; Precup, 2000; Farajtabar et al., 2018). While those methods are based on different assumptions and with different formulations, most of them (1) focus on precisely estimating the expected return of a target policy using pre-collected historical data, and (2) perform unsupervised estimation without directly leveraging the online performance of previously deployed policies.

We notice that there are some mismatches between the settings of those OPE methods and the OPE problem in real-world applications. First, in many applications, we do not need to estimate the exact true performance of a target policy. Instead, what we need is to compare the true performance of a set of candidate policies and identify the best one which will then be deployed into real-world systems. That is, correct ranking of policies rather than precise return estimation is the end goal of off-policy evaluation. Second, in real-world applications, we usually know the true performance of some polices that have been deployed into real-world systems and interacted with real-world users. Such information is not well exploited in today's OPE methods.

Based on these observations, in this work, we define a new problem, supervised off-policy ranking (SOPR), which is

[1] Department of Electronic Engineering, Tsinghua University, Beijing, China [2] Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, China [3] Microsoft Research Asia, Beijing, China. Correspondence to: Tao Qin <taoqin@microsoft.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

[1] https://github.com/SOPR-T/SOPR-T

[2] We use online performance and true performance interchangeably in this paper.

different from previous OPE in two aspects. First, SOPR aims to correctly rank new policies, rather than accurately estimate their expected returns. Second, SOPR leverages the true performance of a set of pre-collected policies in addition to off-policy data.

A straightforward way to rank policies is to first use a scoring model to score each policy and then rank them based on their scores. Following this idea, we propose a supervised learning-based method to train a scoring model. The scoring model takes logged states in historical data and the actions taken by a policy offline as raw features. Considering that there might be plenty of data, we design a hierarchical Transformer encoder (TE) based model to learn representations at different levels. Specifically, we first adopt k-means clustering to cluster similar states. Within each cluster, a low-level TE is employed to encode each state-action pair. At high level, another TE is employed to encode each cluster. Following each TE, an average pooling operator is adopted to aggregate information in the corresponding level. Finally, a multi-layer perceptron (MLP) maps the overall representation to a score. The hierarchical TE and MLP are jointly trained to correctly rank the pre-collected policies with known performance. Our method is named as SOPR-T, where "T" stands for our TE-based model.

We evaluate SOPR-T on public datasets for offline reinforcement learning (RL) (Fu et al., 2020). Experiments on multiple tasks and different datasets demonstrate that SOPR-T outperforms representative baseline methods in terms of both rank correlation, regret value, and stability.

The main contributions of this work are as follows:

- We point out that OPE should focus on policy ranking rather than exactly estimating the returns of policies, which simplifies the OPE problem.
- According to our knowledge, we are the first to introduce supervised learning into OPE. We take a preliminary step towards supervised OPE in this work and define the problem of supervised off-policy ranking. We hope that our work can inspire more solid works along this direction.
- We propose a hierarchical Transformer encoder based model for policy ranking. Experiments demonstrate the effectiveness and stability of our method. Our code and data have been released at GitHub.

2. Supervised Off-Policy Evaluation/Ranking

In this section, we first give some notations and then formally describe the problems of supervised off-policy evaluation and supervised off-policy ranking.

We consider OPE in a Markov Decision Process (MDP) defined by a tuple (S, A, T, R, γ). S and A denote state and action space. $T(s'|s, a)$ and $R(s, a)$ are state transition distribution and reward function. $\gamma \in (0, 1]$ is a discount factor. The expected return of a policy π is defined as $V(\pi) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t]$, where $a_t \sim \pi(\cdot|s_t), r_t \sim R(s_t, a_t)$, and T is the time horizon of the MDP.

The goal of OPE is to evaluate a policy without running it in the environment. Traditional OPE methods estimate the expected return of a policy leveraging a pre-collected dataset $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$ composed of N trajectories generated by some other policies (usually called behavioral policy), where $\tau_i = s_0^i, a_0^i, r_0^i, \cdots, s_T^i, a_T^i, r_T^i$.

Apart from the pre-collected dataset, we can also collect some previously deployed policies whose true performance is available. For instance, in many real-world applications, such as advertising and recommendation systems, we can get the true performance of previously deployed polices by observing and counting user clicks. Thus, the true performance of these policies is available. Clearly, those information is helpful for OPE, but was ignored in previous OPE methods. In this work, we define a new kind of OPE problem, the supervised OPE problem, as below.

Definition 2.1. Supervised off-policy evaluation: given a pre-collected dataset $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$ with N trajectories and M pre-collected policies $\{\pi_i\}_{i=1}^{M}$ with known performance, estimate the performance of a target policy or a set of target policies without interacting with the environment.

Comparing with previously studied OPE problems, our new problem has more available information, the set of pre-collected policies $\{\pi_i\}_{i=1}^{M}$ with known performance, which can serve as supervised signal while we learn an OPE model. We name this problem "supervised" OPE as we have label information (i.e., the true performance) for the policies available in training. In contrast, previous OPE problems can be deemed unsupervised learning problem, since they do not directly learn from the online performance of pre-collected policies.

In addition, in real-world applications we often need to compare a set of candidate policies and choose a good one from them. Thus, what we really need is the correct ordering of a set of policies, rather than the exact performance value of each policy. Formally, we define a variant of the supervised OPE problem, which is called supervised off-policy ranking.

Definition 2.2. Supervised off-policy ranking: given a pre-collected dataset $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$ with N trajectories and M pre-collected policies $\{\pi_i\}_{i=1}^{M}$ together with their performance ranking, rank a set of target policies without interacting with the environment.

It is not difficult to see that policy ranking is relatively easier than policy evaluation, since accurate evaluation inherently implies correct ranking, but correct ranking does not need accurate evaluation. Considering the practical value and simplicity of policy ranking, we focus on supervised off-policy ranking (SOPR) in the following parts of this paper.

3. Our Method

We propose a method for SOPR in this section, which involves training a scoring model to score and rank a set of policies. We start with introducing policy representation (Section 3.1), and then loss function design and training pipeline (Section 3.2).

3.1. Policy Representation

Policies can be of very different forms: a policy can be a set of designed rules, a linear function of states, or deep neural networks, e.g., convolutional neural networks, recurrent neural networks, and attention-based networks. To map a policy to a score, the first question to answer is how to represent different forms of policies.

To handle all kinds of policies, we should leverage their shared points rather than differences. Obviously, we cannot use the internal parameters of policies, because different policies may have different numbers of parameters and some policies do not have parameters. Fortunately, all the policies for the same task or game share the same input space, state space S, and the same output space, action space A. Therefore, we propose to learn policy representations upon state-action pairs.

Let $\mathcal{D}_s = \{s_i\}_{i=1}^{N_s}$ denote a set of states included in the pre-collected dataset $\mathcal{D}$, where N_s is the number of these states. For a policy π to be ranked, let it take actions for all the states in $\mathcal{D}_s$ and obtain a dataset $\mathcal{D}_\pi = \{(s_i, a_i^\pi)\}_{i=1}^{N_s}$, where $s_i \in \mathcal{D}_s, a_i^\pi \sim \pi(\cdot|s_i)$. Now, any policy π can be represented by a dataset $\mathcal{D}_\pi$ in the same format. [3]

Now the question becomes how to map a set of state-action pairs/points[4], $\mathcal{D}_\pi = \{(s_i, a_i^\pi)\}_{i=1}^{N_s}$, to a score that indicates the performance of policy π. Following (Kool et al., 2018; Nazari et al., 2018; Bello et al., 2016; Xin et al., 2020; Vinyals et al., 2015), we design a Transformer encoder (TE) (Vaswani et al., 2017) based model to encode all those points. We (1) first project each point into an embedding vector, (2) use a few self-attention layers to get high-level representations of those points, (3) aggregate them by average pooling to get a representation of the dataset (and thus the policy), and (4) finally adopt a linear projection to map the vector to a score.

A computational challenge is that there could be millions of states in the pre-collected historical data. It is impossible for Transformer to handle such a large scale of data. Our solution is to down sample the states and encode the state-action pairs in a hierarchical way as follows:

1. Randomly sample a subset D_s of states from $\mathcal{D}_s$ and cluster them into K clusters by k-means clustering.
2. Let a policy π take actions over the states in D_s and obtain a set of state-action pairs.
3. Use a low-level TE to encode all the state-action pairs in a cluster and get a vector representation for each cluster by average pooling.
4. Use a high-level TE to encode all the clusters and get a vector representation for the policy by average pooling.
5. Use a linear projection to map the vector in the above step to a score.

Since our scoring function is based on Transformer, we name our method SOPR-T. Figure 1 illustrates the pipeline of SOPR-T. Figure 2 shows the architecture of our proposed hierarchical TE based policy scoring model.

3.2. Pairwise Loss and Learning Algorithm

As aforementioned, the goal of SOPR is to rank a set of policies correctly. We adopt a pairwise ranking loss following (Burges et al., 2005):

$$l(\pi_i, \pi_j; \theta) = -\Bigg[y_{ij}\log(\frac{1}{1+e^{-(f(\pi_i, D_s;\theta)-f(\pi_j, D_s;\theta))}}) \\ +(1-y_{ij})\log(1-\frac{1}{1+e^{-(f(\pi_i, D_s;\theta)-f(\pi_j, D_s;\theta))}})\Bigg], \quad (1)$$

where $f(\pi, D_s; \theta)$ denotes the scoring function with parameter θ, π_i and π_j are two policies with a performance ranking y_{ij}. $y_{ij} = 0$ means π_i is worse than π_j, $y_{ij} = 1$ means π_i is better than π_j, and $y_{ij} = 0.5$ means the two policies perform similarly. To minimize the loss, the scoring function needs to correctly rank two policies, i.e., consistent with the ranking of their true performance.

The complete learning algorithm of SOPR-T is shown in Algorithm 1. Note that, in training, we sample multiple subsets of states in a manner similar to mini-batch training. In inference, we sample multiple subsets and compute a score using the well-trained scoring function over each subset. The final score of a test policy is the average score over those subsets.

3.3. Discussions

One may notice that we use only the states in the pre-collected historical data $\mathcal{D}$, while most previous OPE methods use both the states and immediate rewards $R(s, a)$ in

[3] For the simplicity of description, we consider deterministic policies here. For stochastic policies, we can use the distribution or the statistics over actions for a state s to create the dataset $\mathcal{D}_\pi$.

[4] A state-action pair can be deemed a point in a high-dim space.

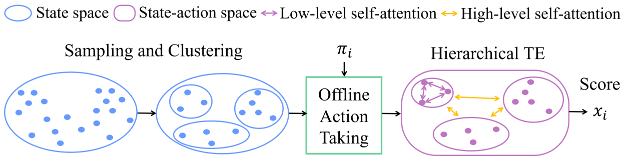


Figure 1. Illustration of SOPR-T method.

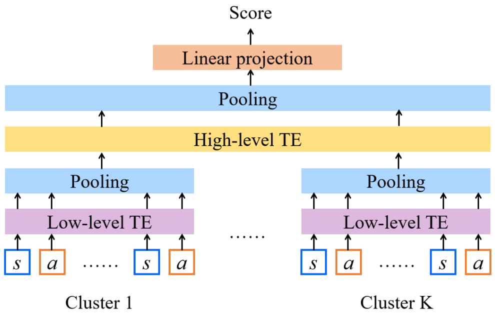


Figure 2. Architecture of hierarchical Transformer encoder based scoring model.

Algorithm 1 Training procedure of SOPR-T

```
Input: State dataset $\mathcal{D}_s$, training policy set $\{\pi_i\}_{i=1}^M$, and pairwise ranking label $\{y_{ij}\}_{i,j=1,i\neq j}^M$.
Initialize: Scoring model $f(\pi, \mathcal{D}_s; \theta)$.
for iteration $t$ in $\{1, \cdots, T\}$ do
    Sample a subset $D_s^t$ of states from $\mathcal{D}_s$.
    Compute the score $f(\pi_i, D_s^t; \theta)$ as described in Section 3.1 for all the $M$ policies.
    Perform gradient update to minimize the ranking loss $l(\pi_i, \pi_j; \theta)$ for each pair of policy $\pi_i$ and $\pi_j$.
end for
```

$\mathcal{D}$. There are several considerations for only using states. First, a new policy to be evaluated usually takes actions different from the historical data. For a state with a new action a', $R(s, a')$ is unknown if such a state-action pair is not contained in the historical data. Thus, we do not directly use immediate rewards in this work. Second, in the setting of SOPR, we have a set of training policies with known performance. The performance information of those policies is more reliable and can be used as a direct signal for supervised learning, as compared with immediate rewards. This is because the performance information comes from online interactions with real-world systems and directly indicates the performance of a policy. Third, consider the following formulation of the expected return of a policy,

$$V(\pi) = \mathbb{E}[d^\pi(s, a)R(s, a)], \tag{2}$$

where $d^\pi(s, a)$ denotes the stationary distribution of state-action pairs under π. Note that, only $d^\pi(s, a)$ depends on policy π, while $R(s, a)$ is the same across all policies for a task. Therefore, to rank different policies for a task, the more important part is $d^\pi(s, a)$, rather than the immediate rewards. Of course, if we can work out a good solution to accurately predict and well leverage immediate rewards, this will further improve the accuracy of supervised off-policy evaluation/ranking. Such a problem is beyond the scope of this paper. We leave it to future work.

Although we focus on supervised off-policy ranking in this work, our proposed scoring model can be easily applied to supervised OPE. For this purpose, we need the true performance of training polices, and only performance ranking is not enough. Given the true performance of training policies, we can train the scoring model by minimizing the gap between the true performance and the predicted performance of a policy.

4. Experiments

We compare SOPR-T with different kinds of baseline OPE algorithms on various tasks, including evaluating policies learned by different algorithms, in different games, and with different settings of datasets. Below we first introduce experimental settings, including tasks, training/validation/test sets, baselines, and evaluation metrics. Then, we present experimental results regarding performance comparison among different algorithms as well as further studies of our algorithm.

4.1. Experimental Settings

Tasks We evaluate SOPR-T and baseline OPE algorithms on D4RL datasets [5] (Fu et al., 2020) which are widely used in offline RL studies. Overall, we use 12 tasks covering three Mujoco games, i.e., Hopper-v2, HalfCheetah-v2, and Walker2d-v2, and four types of off-policy datasets for each game, i.e., expert, medium, medium-replay (m-replay for short), and full-replay (f-replay for short). The expert and medium datasets are collected by a single expert and medium policy, respectively. The two policies are trained with Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018) online, and the medium policy only achieves 1/3 performance of the expert policy. The full-replay and medium-replay datasets contain all the data collected during the training of the expert and medium policy, respectively.

Training Set and Validation Set Training set and validation set consist of policies and their rank labels. The policies are collected during online SAC training. For each game, we collect 50 policy snapshots during the training process and get their performance by online evaluation using 100 Monte-Carlo rollouts in the real environment. After the

[5] https://github.com/rail-berkeley/d4rl

training process is finished, we randomly select 30 policies to form training policy set and another 10 policies to form validation policy set. The remaining 10 policies are used to form a test policy set. We will provide a detailed description of the test policy set later. Note that, in the training phase of SOPR-T, only the rank of the policies are used as labels.

Test Set In each task, we use two kinds of test policy sets to simulate two kinds of OPE cases.

In Test Set I, we investigate the capability of SOPR-T and baseline OPE algorithms to rank and select good policies in offline RL settings. Specifically, Test Set I is composed of policies collected by running 3 popular offline RL algorithms, BEAR (Kumar et al., 2019), CQL (Kumar et al., 2020), and CRR (Wang et al., 2020). The implementation of the three algorithms is based on a public codebase [6]. Note that, these algorithms adopt different network architectures. Thus, the types of policy models are not all the same. For each task, we run each algorithm until convergence and collect policy snapshots during training. To get the ground truth rank labels of the performance of these policies, we also evaluate the performance of each policy using 100 Monte-Carlo rollouts in the real environment, and use the rank of the performance as rank labels. Then, we mix these policies and select 10 policies whose performance are evenly spaced in the performance range of all policies. In this way, the selected polices have diverse performance. In addition, mixing policies generated by different algorithms to form a test policy set is aligned with the practical cases where the sources of policies are various and unknown.

In Test Set II, we investigate the capability of SOPR-T to rank policies that are approximately within the distribution of training policies. Because in practice, such as production development and update, the updated policies and the previously evaluated policies usually have many common properties. Thus, it can be assumed that the policies to be evaluated and the policies that have been evaluated are in two similar distributions. To simulate this case in experiments, we leverage the 10 policies mentioned in the description regarding the training policy set to form Test Set II. Because the 10 policies and the training policies are uniformly sampled from the same policy set, they are approximately within the same distribution. As these policies are learned online, we name Test Set II online learned policies.

Baselines We compare SOPR-T with four representative baseline OPE algorithms.

1. Fitted Q-Evaluation (FQE) (Le et al., 2019), which is a Q-estimation based OPE method. It learns a neural network to approximate the Q-function of the target policy by leveraging Bellman expectation backup operator (Sutton & Barto, 2018) iteratively based on the off-policy data.

2. Model-based estimation (MB). It estimates the environment model including a state transition distribution and a reward function. The expected return of the target policy is estimated using the returns of Monte-Carlo rollouts in the modeled environment.

3. Weighted importance sampling (IW), which is a distribution correction-based method. It leverages importance sampling to correct the weight of the reward regarding the data from the behavior data distribution to the target data distribution, and adopts weight normalization.

4. DualDICE (Nachum et al., 2019), which is also based on distribution correction, but without directly using importance weights. It learns to estimate the state-action stationary distribution correction.

We leverage a popular implementation [7] of these algorithms.

Evaluation Metrics We evaluate SOPR-T and baseline OPE algorithms with two metrics, i.e., Spearman's rank correlation coefficient and normalized Regret@k, to reflect their performance of ranking candidate policies, which is aligned with a related work (Paine et al., 2020). Specifically, Spearman's rank correlation is the Pearson correlation between the ground truth rank sequence and the evaluated rank sequence of the candidate policies. Normalized Regret@k is the normalized performance gap between the truly best policy of all candidate policies and the truly best policy of the ranked top k policies, i.e., Regret@k $= (V_{\text{max}} - V_{\text{max\_topk}})/(V_{\text{max}} - V_{\text{min}})$, where V_{max} and V_{min} are the ground truth performance of the truly best and the truly worst policy of all candidate policies, respectively. $V_{\text{max\_topk}}$ is the ground truth performance of the truly best policy of the ranked top k policies. We use $k = 3$ in our experiments.

4.2. Performance on Offline Learned Policies

We first evaluate SOPR-T and baseline OPE algorithms on Test Set I, i.e., the offline learned policies. In the evaluation process, we use 3 random seeds for each experiment.

Due to space limitation, we only present the results on the Hopper game (top row of Figure 3), and an overall performance ranking of 5 algorithms on 12 tasks (second row of Figure 3) here. Results on other games can be found in Appendix A.3. As can be seen from the results shown in Figure 3(a) and Figure 3(b), SOPR-T achieves higher rank correlation coefficient and smaller regret value than baseline algorithms, which means SOPR-T can rank different poli-

[6] https://github.com/takuseno/d3rlpy

[7] https://github.com/google-research/google-research/tree/master/policy_eval

cies with higher accuracy and also figure out good policies from the candidate policies. In addition, SOPR-T performs the most stably, which does not have negative rank correlation results in all the tasks, whereas each of the baseline OPE algorithms has one or more negative rank correlation results. Though in Walker2d and Halfcheetah (shown in Appendix A.3), SOPR-T does not hold consistent superiority, it still performs the most stably.

Figure 3(c) and Figure 3(d) present the overall performance ranking statistics of five algorithms (SOPR-T and four OPE baselines) on 12 tasks. The results in Figure 3(c) indicate that SOPR-T has the top performance of rank correlation on 5 tasks. In Figure 3(d), SOPR-T has the top performance of regret value on 6 tasks. Among all the algorithms, the number of tasks on which SOPR-T achieves the top performance in terms of both rank correlation and regret value is the highest. The results demonstrate the overall advantage of SOPR-T.

4.3. Performance on Online Learned Policies

Then, we evaluate SOPR-T and OPE baselines on Test Set II. Because these policies are collected during online training rather than offline trained with the datasets, they are irrelevant to the datasets, which is aligned with many practical cases where the policies to be evaluated are not designed or learned based on the dataset.

We also run SOPR-T and each baseline OPE algorithm with 3 different seeds. Results on the Hopper game are shown in the third row of Figure 3 and other games in Appendix A.4. The results demonstrate that SOPR-T outperforms baseline OPE algorithms dramatically on almost all the tasks. Performance ranking statistics of all algorithms on 12 tasks are shown in the bottom row of Figure 3. As can be seen from the results, SOPR-T achieves the top performance of rank correlation on 10 tasks and the top performance of regret value on all the 12 tasks.

Compared with the performance of SOPR-T on ranking the offline learned policies, the performance on ranking the online learned policies improves. We consider that the performance of SOPR-T is affected by the distribution difference between the training policy set and test policy set. In this respect, we measure the distribution distance between the two policy sets by:

$$D\left(\{\pi_i^{\text{train}}\}_{i=1}^{M_{\text{train}}}, \{\pi_j^{\text{test}}\}_{j=1}^{M_{\text{test}}} \Big| \mathcal{D}_s\right) = \frac{1}{M_{\text{test}}}\sum_{j=1}^{M_{\text{test}}} \min_i\left(\frac{1}{N_s}\sum_{n=1}^{N_s}||a_n^{\pi_j^{\text{test}}} - a_n^{\pi_i^{\text{train}}}||_2^2\right), \qquad (3)$$

where N_s is the number of states in the dataset, $a_n^\pi \sim \pi(\cdot|s_n), s_n \in \mathcal{D}_s$, M_{train} and M_{test} are the number of policies in the training set and test set, respectively. Distance

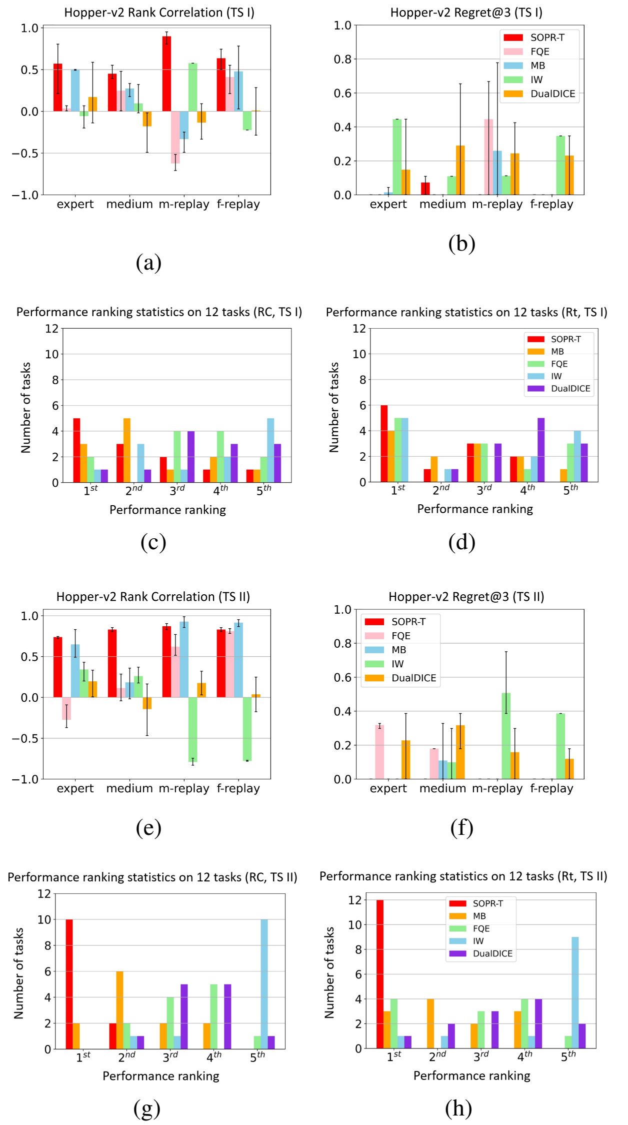


Figure 3. Performance comparison. Left: rank correlation (abbreviate to RC). Right: regret@3 (abbreviate to Rt). Top 2 rows: Test Set I (offline learned policies). Bottom 2 rows: Test Set II (online learned policies). Row 1 and Row 3: performance comparison on 4 types of datasets of the Hopper game. Row 2 and Row 4: performance ranking statistics of 5 algorithms on 12 tasks.

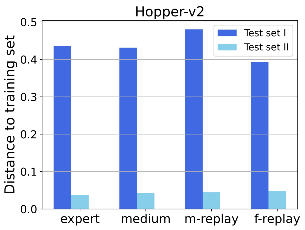


Figure 4. Distance between training and test policy sets.

results are shown in Figure 4 and Figure 11 in Appendix A.4. As can be seen from the results, the distance between Test Set I and the training policy set is much larger than the distance between Test Set II and the training policy set.

4.4. Further Studies

In this part, we further conduct a set of experiments to better understand our algorithm.

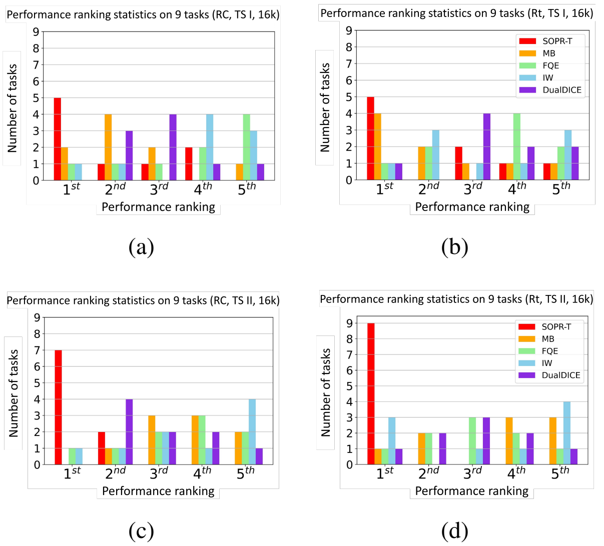
Effect of Data Size We investigate the sensitivity of SOPR-T and baseline OPE algorithms to the size of dataset. We set the number of data (states used in SOPR-T, tuples used in baseline OPE algorithms) as 4k, 8k, 16k, and 32k. Note that, in the original datasets, the size of data is not identical among different tasks, but all of them contain more than 200k data.

We sample different amounts of data from the original dataset in trajectory form. For the expert and medium datasets (composed of data collected by a single policy), trajectories are sampled randomly. For the medium-replay dataset (composed of all data during training SAC in sequence), we fetch data sequentially. Note that, in this experiment, we use three kinds of datasets, i.e., expert, medium, and medium-replay, as we suppose the first part of data in the full-replay dataset is similar to the medium-replay dataset. We still test three games as before, and thus there are 9 tasks (different games or different datasets) in total in this part of experiments.

Due to space limitation, here we only show performance ranking statistics of all the algorithms on 9 tasks corresponding to 16k data in Figure 5. Detailed performance of each algorithm on each individual task can be found in Figure 12 and Figure 13 in Appendix A.5. The results indicate that when using much less data, SOPR-T still outperforms baseline OPE algorithms.

Figure 6 shows the average results over all tasks of each algorithm with different data size. The results demonstrate that SOPR-T is relatively robust to data size, although with a slight drop as dataset decreases in size. Remarkably, SOPR-T outperforms baseline OPE algorithms consistently.

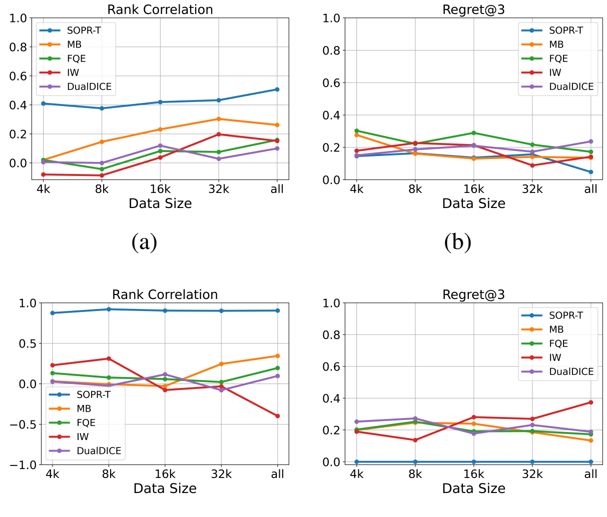
Effect of Transformer Encoder We investigate the effect of using Transformer encoder to encode state-action pairs. To this end, we replace Transformer encoder with MLP encoder. We name the MLP-based SOPR as SOPR-MLP. Both the number of hidden layers and the number of units of each hidden layer in the MLP encoder are aligned with the Transformer encoder used in SOPR-T. Details of training and inference methods are also the same as SOPR-T.



(a) (b)

(c) (d)

Figure 5. Performance ranking statistics of 5 algorithms when the size of dataset is 16k. Left: rank correlation. Right: regret@3. Top row: Test Set I. Bottom row: Test Set II.

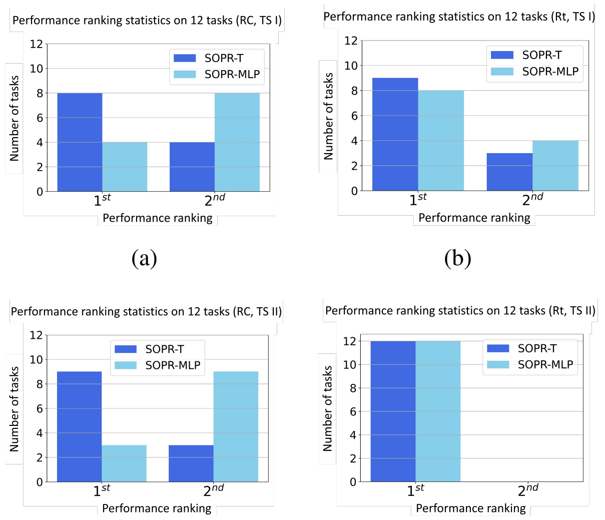


(a) (b)

(c) (d)

Figure 6. Performance comparison with different data size. Top row: average results on ranking the policies in Test Set I. Bottom row: average results on ranking the policies in Test Set II. Left: rank correlation. Right: regret@3.

We compare the performance of SOPR-T and SOPR-MLP also with two test policy sets. Figure 7 presents the performance ranking statistics between SOPR-MLP and SOPR-T on 12 tasks. As can be seen from the results, SOPR-T outperforms SOPR-MLP in most of the tasks. Specifically, for the results on Test Set I, as shown in the top row of Figure 7, SOPR-T achieves higher rank correlation in 8 out of 12 tasks and lower or the same (zero) regret value in 9 out of 12 tasks. For Test Set II, as shown in the bottom row of Figure 7, the results demonstrate that SOPR-T achieves higher rank correlation in 9 out of 12 tasks, and the same (zero) regret value in all the tasks. The results indicate that the Transformer encoder is superior to MLP encoder in terms of encoding state-action pairs and representing policies in our SOPR method. Detailed performance of each algorithm on each individual task can be found in Figure 14 and Figure 15 in Appendix A.5.



(a)

(b)

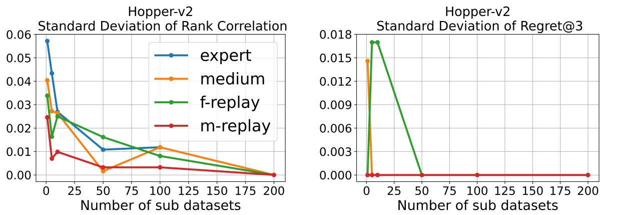
(c)

(d)

Figure 7. Performance ranking statistics between SOPR-T and SOPR-MLP on 12 tasks. Top row: Test Set I. Bottom row: Test Set II. Left: rank correlation. Right: regret@3.

Performance Variance In this part, we investigate the performance variance of SOPR-T in inference, especially when using small data size in inference. In many real-world applications, the inference speed of policy ranking matters a lot. Although our SOPR-T is efficient due to its end-to-end scoring model, we would also like to investigate whether SOPR-T can use less data in inference than that used in the training phase in order to reduce inference time. To this end, in inference, we sample different amounts (1, 5, 10, 50, 100, and 200) of subsets from the whole dataset to calculate an average score for each policy, and then rank different policies with their scores. The size of each subset is the same as the batch size of training, which is provided in Appendix A.1. Note that, in the above experiments regarding the Effect of Data Size, both training and testing are conducted on the same dataset. However, results in this part correspond to using less data (subsets) in inference, and using the original dataset in training.

Given the number of sampled subsets, we repeat the inference process five times, and compute the variance of the five results regarding rank correlation or regret value. The relationship between the standard deviation of the results and the number of sampled subsets is shown in Figure 8 (Hopper game) and Appendix A.5 (Figure 16 and Figure 17). As can be seen from the figures, the standard deviations of rank correlation and regret value of SOPR-T are both small consistently for different amounts of subsets. As the number of subsets increases, the standard deviation decreases gradually, which indicates SOPR-T achieves more stable performance.



(a)

(b)

Figure 8. Standard deviation of SOPR-T with different number of subsets in inference. Left: rank correlation. Right: regret@3.

5. Conclusions and Future Work

In this work, we defined a new problem, supervised off-policy ranking (SOPR), which aims to score target policies for the purpose of correct ranking them instead of precise return estimation. We leveraged a set of training policies with known performance ranking in addition to the off-policy data. We proposed a hierarchical Transformer encoder-based scoring model, which represents a policy by a set of state-action pairs and then maps such a set to a score indicating the relative performance of the policy. The scoring model is trained by minimizing a pairwise ranking loss. Experiments demonstrate that our SOPR method outperforms four representative baseline OPE algorithms in terms of rank correlation, regret value, and stability.

As SOPR is a newly defined problem, our work is just a first and preliminary step and far away from solving it. Many possible future directions are left to explore. First, beyond the state-action pair-based representations used in this work, how to better represent a policy? For example, considering the existence of many OPE methods, the values of a policy estimated by those methods can be used to represent the policy, and one can conduct supervised learning to combine the estimations of those methods. Furthermore, the policy values estimated by those OPE methods can serve as features and enhance the representations of a policy adopted in this work. Second, we employed a Transformer based scoring model in this work. It is interesting to explore other possibilities for the scoring model. Third, we adopted the pairwise ranking loss in our algorithm. There are other ranking loss functions (e.g., listwise loss (Cao et al., 2007)) which are worth trying. Fourth, since we conducted supervised learning, a limitation of our algorithm is that its effectiveness will be heavily impacted by the quality of training policies. We will study and quantify the impact of the quality of training policies in our future work. Last but not least, as a supervised learning problem, there are many theoretical questions to answer (Vapnik, 1999; 2013). For example, is a learning algorithm consistent for supervised OPE or supervised OPR? What is the rate of convergence of a learning algorithm? What is the generalization bound of a learning algorithm? We hope that our work can inspire more research along the direction of supervised OPE/OPR.

References

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96, 2005.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, 2007.

Farajtabar, M., Chow, Y., and Ghavamzadeh, M. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning*, pp. 1447–1456. PMLR, 2018.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

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

Hanna, J., Stone, P., and Niekum, S. Bootstrapping with models: Confidence intervals for off-policy evaluation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Hanna, J., Niekum, S., and Stone, P. Importance sampling policy evaluation with an estimated behavior policy. In *International Conference on Machine Learning*, pp. 2605–2613. PMLR, 2019.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. Q (λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pp. 305–320. Springer, 2016.

Kool, W., Van Hoof, H., and Welling, M. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Kostrikov, I. and Nachum, O. Statistical bootstrapping for uncertainty estimation in off-policy evaluation. *arXiv preprint arXiv:2007.13609*, 2020.

Kumar, A., Fu, J., Tucker, G., and Levine, S. Stabilizing off-policy Q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32 (NeurIPS), 2019. ISSN 10495258.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

Le, H. M., Voloshin, C., and Yue, Y. Batch policy learning under constraints. *36th International Conference on Machine Learning, ICML 2019*, 2019-June(i):6589–6600, 2019.

Liu, Q., Li, L., Tang, Z., and Zhou, D. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, pp. 5356–5366, 2018.

Mannor, S., Simester, D., Sun, P., and Tsitsiklis, J. N. Bias and variance in value function estimation. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 72, 2004.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062, 2016.

Nachum, O. and Dai, B. Reinforcement learning via fenchel-rockafellar duality. *arXiv preprint arXiv:2001.01866*, 2020.

Nachum, O., Chow, Y., Dai, B., and Li, L. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*, 2019.

Nazari, M., Oroojlooy, A., Snyder, L. V., and Takáč, M. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

Paine, T. L., Paduraru, C., Michi, A., Gulcehre, C., Zolna, K., Novikov, A., Wang, Z., and de Freitas, N. Hyperparameter Selection for Offline Reinforcement Learning. 2020. URL http://arxiv.org/abs/2007.09055.

Precup, D. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pp. 80, 2000.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT press, 2018.

Thomas, P. and Brunskill, E. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pp. 2139–2148, 2016.

Thomas, P. S., Theocharous, G., and Ghavamzadeh, M. High-confidence off-policy evaluation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Vapnik, V. *The Nature of Statistical Learning Theory*. Springer science & business media, 2013.

Vapnik, V. N. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

Wang, Z., Novikov, A., Żołna, K., Springenberg, J. T., Reed, S., Shahriari, B., Siegel, N., Merel, J., Gulcehre, C., Heess, N., et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

Xie, T., Ma, Y., and Wang, Y.-X. Towards optimal off-policy evaluation for reinforcement learning with marginalized importance sampling. In *Advances in Neural Information Processing Systems*, pp. 9668–9678, 2019.

Xin, L., Song, W., Cao, Z., and Zhang, J. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. *arXiv preprint arXiv:2012.10638*, 2020.

Yang, M., Nachum, O., Dai, B., Li, L., and Schuurmans, D. Off-policy evaluation via the regularized lagrangian. *arXiv preprint arXiv:2007.03438*, 2020.

Zhang, M. R., Paine, T., Nachum, O., Paduraru, C., Tucker, G., ziyu wang, and Norouzi, M. Autoregressive dynamics models for offline policy evaluation and optimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=kmqjgSNXby.

Zhang, R., Dai, B., Li, L., and Schuurmans, D. Gendice: Generalized offline estimation of stationary values. *arXiv preprint arXiv:2002.09072*, 2020.

A. Appendix

A.1. Model and Training Configurations

Table 1 lists the configurations of our model and training process.

Table 1. Model and training configurations for SOPR-T.

Hyperparameter	Value
Input linear projection layer	((dim_s+dim_a), 64)
Low-level encoder	n_layers=2, n_head=2, dim_feedforward=128, dropout=0.1
High-level encoder	n_layers=6, n_head=8, dim_feedforward=512, dropout=0.1
Output linear projection layer	(256, 1)
Optimizer	Adam
Learning rate	0.001
Batch size	$\lvert D_s \rvert = 16$k
Number of clusters	$K = 256$

A.2. Computational Resource and Time Cost

Our experiments are run with a Nvidia Tesla P100 GPU. Table 2 and Table 4 present training and inference time cost of SOPR-T regarding different random seeds, respectively. Table 3, Table 5, and Table 6 present comparison of average time cost among different algorithms. Note that, the time cost of SOPR-T in inference depends on the number of subsets used to calculate an average score for a policy. Unless otherwise specified, in inference, we use all (200) subsets that are used in training. We observe that, as shown in the last part of Section 4.4 and the last part of Appendix (Figure 16), SOPR-T with 5 subsets achieves nearly the same performance as using 200 subsets. Approximately, to score and rank 10 policies in each task shown in Table 4, SOPR-T takes less than **25.24** (1009.6/40) seconds to get a good ranking, which is comparable to IW and much faster than FQE and DualDICE.

Table 2. Training time cost of SOPR-T. (seconds)

Task Name	Seed 0	Seed 1	Seed 2	Average
halfcheetah-expert	3937.8	3987.3	3993.0	**3972.7**
halfcheetah-medium	3816.1	3973.8	3923.8	**3904.6**
halfcheetah-medium-replay	4653.5	4987.4	4969.4	**4870.1**
halfcheetah-full-replay	5283.5	5151.5	6300.0	**5578.3**
hopper-expert	3204.7	3214.5	3228.9	**3216.0**
hopper-medium	3216.3	3177.5	3298.5	**3230.8**
hopper-medium-replay	3799.9	3811.7	4123.5	**3911.7**
hopper-full-replay	3837.4	3795.7	3909.6	**3847.6**
walker2d-expert	3702.3	4030.1	3678.7	**3803.7**
walker2d-medium	3601.0	3740.9	3647.4	**3663.1**
walker2d-medium-replay	4687.4	4719.2	4606.0	**4670.9**
walker2d-full-replay	4683.7	4566.4	4648.7	**4632.9**

Table 3. Comparison of training time cost. (seconds)

Task Name	**SOPR-T**	FQE	DualDICE	MB	IW
halfcheetah-expert	**3972.7**	/	/	6610.0	3841.2
halfcheetah-medium	**3904.6**	/	/	4386.1	4000.8
halfcheetah-medium-replay	**4870.1**	/	/	4682.8	2762.3
halfcheetah-full-replay	**5578.3**	/	/	5037.2	3370.9
hopper-expert	**3216.0**	/	/	7614.4	3695.9
hopper-medium	**3230.8**	/	/	4764.0	3684.9
hopper-medium-replay	**3911.7**	/	/	5104.8	3034.7
hopper-full-replay	**3847.6**	/	/	4776.6	3523.0
walker2d-expert	**3803.7**	/	/	4582.4	3371.0
walker2d-medium	**3663.1**	/	/	4629.5	3542.5
walker2d-medium-replay	**4670.9**	/	/	4780.0	2933.2
walker2d-full-replay	**4632.9**	/	/	4732.3	3532.7

Table 4. Inference time cost of SOPR-T (ranking 10 policies). (seconds)

Task Name	Test Set I				Test Set II			
	Seed 0	Seed 1	Seed 2	Average	Seed 0	Seed 1	Seed 2	Average
halfcheetah-expert	799.9	770.9	775.5	**782.1**	600.4	515.0	523.6	**546.3**
halfcheetah-medium	741.7	775.9	746.5	**754.7**	532.5	543.0	539.8	**538.4**
halfcheetah-medium-replay	854.3	864.5	849.6	**856.1**	658.6	660.1	668.9	**662.5**
halfcheetah-full-replay	1038.1	1050.7	940.1	**1009.6**	657.7	664.7	658.8	**660.4**
hopper-expert	843.4	841.4	833.6	**839.4**	490.3	502.2	494.7	**495.7**
hopper-medium	696.6	701.1	761.8	**719.8**	481.9	488.1	500.6	**490.2**
hopper-medium-replay	772.9	790.1	803.7	**788.9**	586.5	583.7	591.2	**587.1**
hopper-full-replay	737.2	735.0	745.6	**739.3**	586.0	643.9	650.3	**626.7**
walker2d-expert	814.1	809.8	785.0	**803.0**	544.5	518.8	531.3	**531.5**
walker2d-medium	906.5	874.7	893.9	**891.7**	544.7	547.8	520.3	**537.6**
walker2d-medium-replay	781.7	781.8	779.9	**781.2**	652.7	663.5	659.0	**658.4**
walker2d-full-replay	946.9	939.6	946.7	**944.4**	659.3	671.7	656.3	**662.4**

Table 5. Comparison of inference time cost (ranking 10 policies in Test Set I). (seconds)

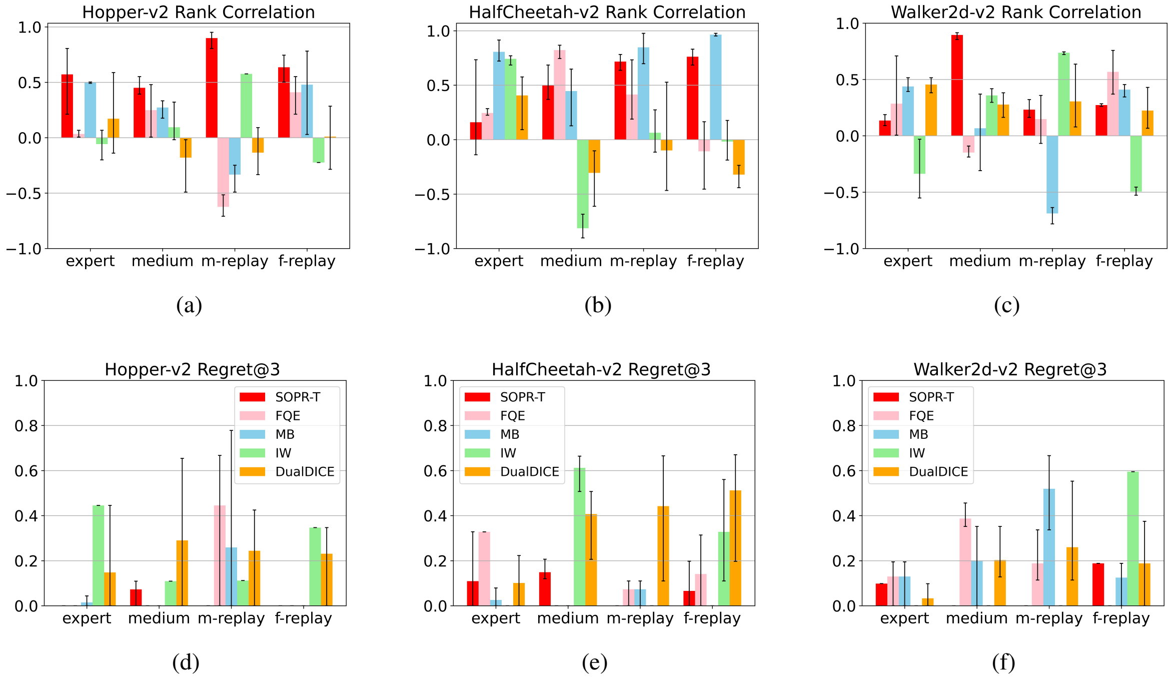
Task Name	**SOPR-T**	FQE	DualDICE	MB	IW
halfcheetah-expert	**782.1**	63262.7	70975.3	142.4	27.8
halfcheetah-medium	**754.7**	39271.1	48123.2	121.3	19.0
halfcheetah-medium-replay	**856.1**	38574.8	44506.9	96.7	14.3
halfcheetah-full-replay	**1009.6**	39221.6	48345.0	141.0	21.1
hopper-expert	**839.4**	39055.9	50544.5	135.0	19.2
hopper-medium	**719.8**	72422.3	79971.6	163.9	26.0
hopper-medium-replay	**788.9**	39408.0	53415.2	137.6	27.1
hopper-full-replay	**739.3**	299882.7	299237.4	446.2	48.8
walker2d-expert	**803.0**	38092.9	58832.3	125.0	19.8
walker2d-medium	**891.7**	39518.1	47013.5	134.2	20.6
walker2d-medium-replay	**781.2**	39394.2	62608.0	142.2	20.3
walker2d-full-replay	**944.4**	47131.7	51518.7	214.9	26.6

Table 6. Comparison of inference time cost (ranking 10 policies in Test Set II). (seconds)

Task Name	**SOPR-T**	FQE	DualDICE	MB	IW
halfcheetah-expert	**546.3**	48336.0	47690.6	125.8	19.6
halfcheetah-medium	**538.4**	45660.3	52757.4	105.4	19.7
halfcheetah-medium-replay	**662.5**	42467.1	56625.4	82.9	14.6
halfcheetah-full-replay	**660.4**	43887.4	47438.6	110.1	21.1
hopper-expert	**495.7**	44551.3	48764.1	103.4	19.3
hopper-medium	**490.2**	51706.9	59532.8	152.1	25.2
hopper-medium-replay	**587.1**	44676.7	58953.9	122.5	25.9
hopper-full-replay	**626.7**	60112.8	90679.3	180.8	37.2
walker2d-expert	**531.5**	52014.5	49146.8	112.1	19.6
walker2d-medium	**537.6**	49727.6	53583.4	115.2	20.7
walker2d-medium-replay	**658.4**	44017.6	49982.8	108.2	20.2
walker2d-full-replay	**662.4**	44859.8	56993.5	129.3	27.3

A.3. Performance on Offline Learned Policies (Corresponding to Section 4.2)

In Section 4.2, we presented the performance of each algorithm on offline learned policies (Test Set I) in the Hopper game. Here, Figure 9 shows all the results in three games. As can be seen from the results, SOPR-T outperforms baseline OPE algorithms consistently on four tasks of the Hopper game. Though in Walker2d and Halfcheetah, SOPR-T does not hold consistent superiority, it performs the most stably. That is, SOPR-T does not have negative rank correlation results in all the tasks, whereas all the baseline OPE algorithms have one or more negative correlation results.



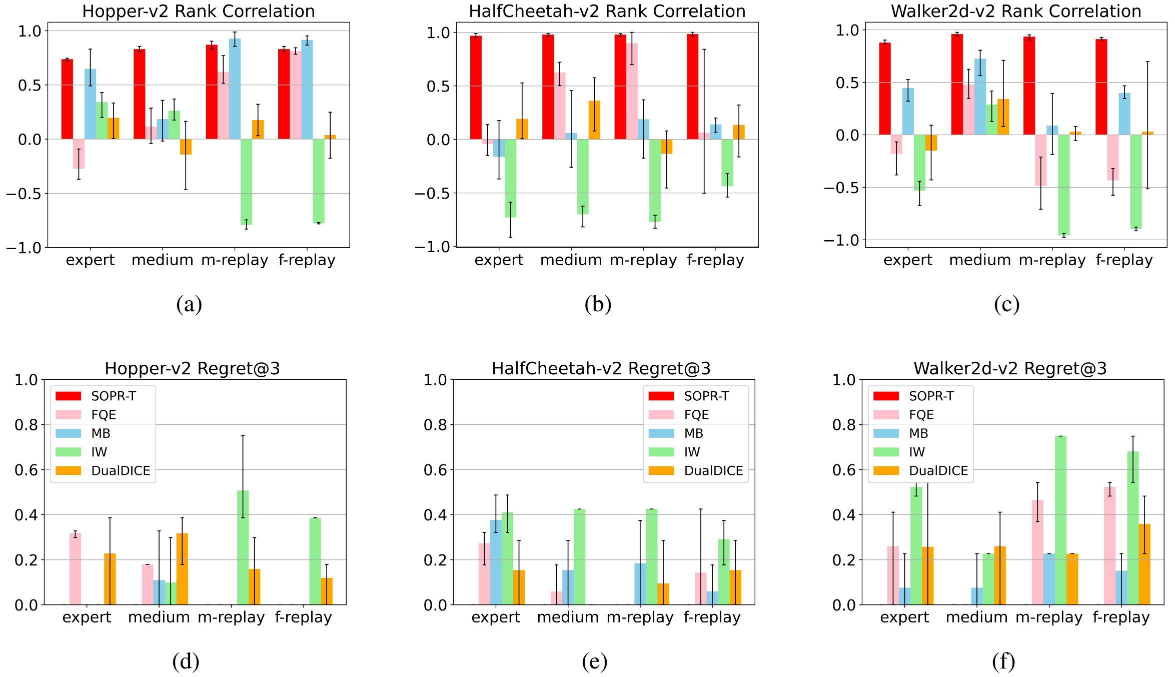
(a) (b) (c)

(d) (e) (f)

Figure 9. Performance comparison (ranking the policies in Test Set I). Top row: rank correlation. Bottom row: regret@3.

A.4. Performance on Online Learned Policies (Corresponding to Section 4.3)

In Section 4.3, we presented the performance of each algorithm on online learned policies (Test Set II) in the Hopper game. Here, Figure 10 shows all the results in three games. As can be seen from the results, SOPR-T achieves very high rank correlation and zero regret values in all the tasks. In addition, the variance caused by random seeds is very small.

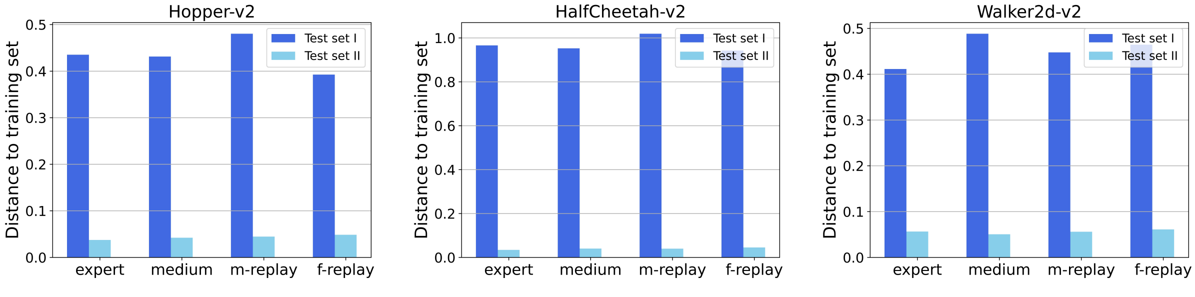


(a) (b) (c)

(d) (e) (f)

Figure 10. Performance comparison (ranking the policies in Test Set II). Top row: rank correlation. Bottom row: regret@3.

Figure 11 presents the distribution distance between the training and test policy sets measured by Eq. (3). As can be seen from the results, in different dataset cases, the distance between Test Set I and the training policy set is larger than the distance between Test Set II and the training policy set. The results reflect the influence of distribution distance between training and test policy sets on the performance of our algorithm.



(a) (b) (c)

Figure 11. Distance between training and test policy sets.

A.5. Additional Results of Section 4.4

Effect of Data Size

In Section 4.4, we presented the performance ranking statistics of 5 algorithms on all tasks when the size of off-policy dataset is 16k. Here, Figure 12 and Figure 13 show detailed performance of each algorithm on each individual task. As shown in Figure 12 and Figure 13, SOPR-T outperforms baseline OPE algorithms in most of the tasks.

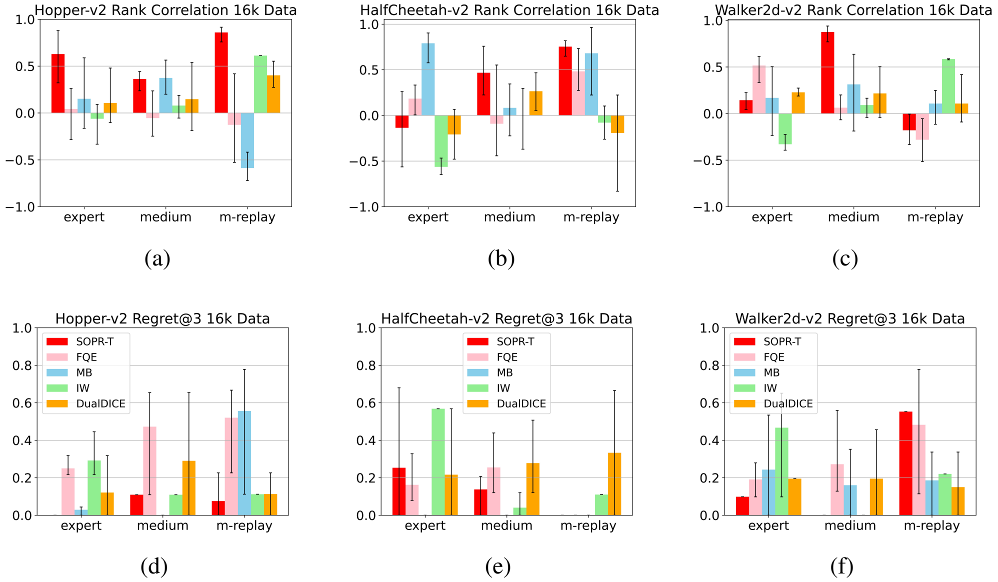


Figure 12. Performance comparison (ranking the policies in Test Set I) when the size of dataset is 16k. Top row: rank correlation. Bottom row: regret@3.

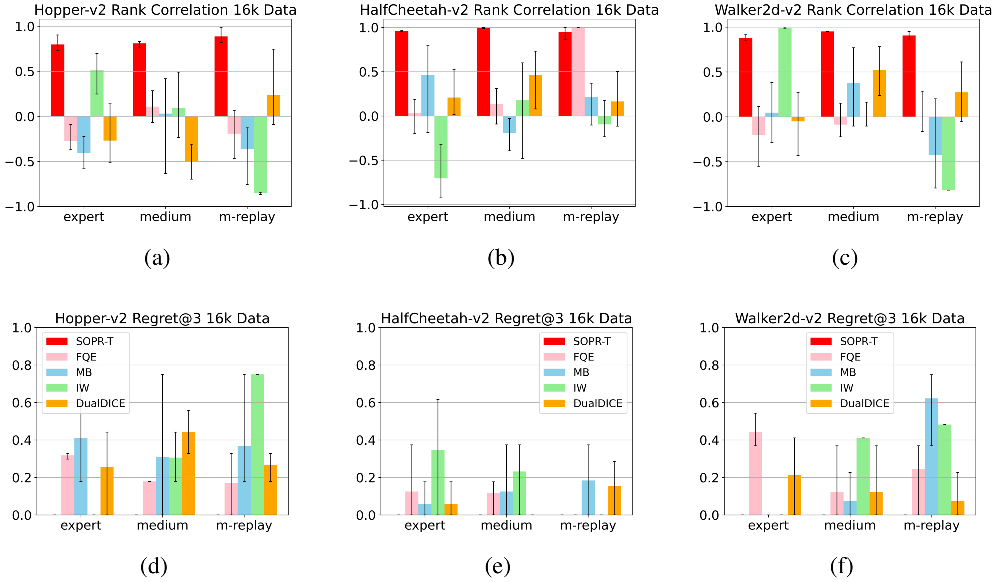
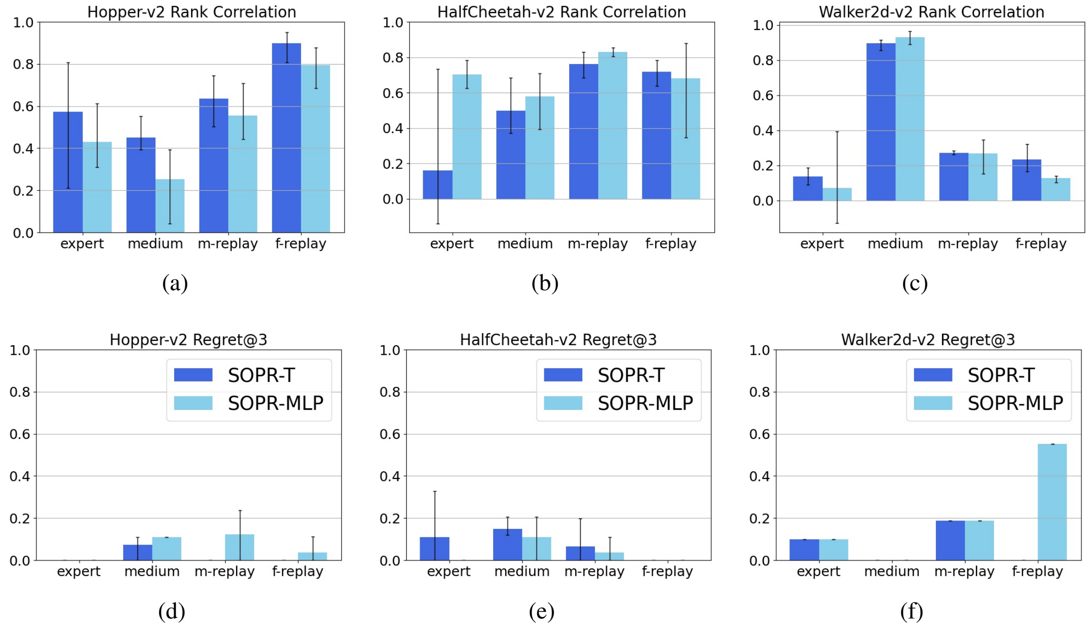


Figure 13. Performance comparison (ranking the policies in Test Set II) when the size of dataset is 16k. Top row: rank correlation. Bottom row: regret@3.

Transformer Encoder vs. MLP Encoder

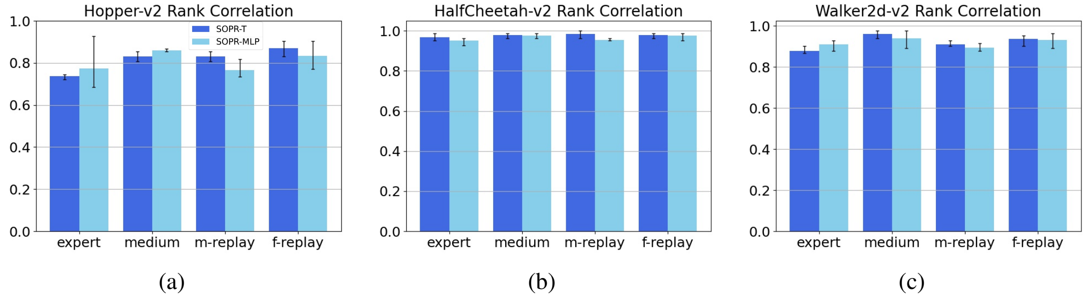
In Section 4.4, we investigated the performance difference between SOPR-T and SOPR-MLP. Here, the detailed performance of SOPR-T and SOPR-MLP on each individual task is shown in Figure 14 (Test Set I) and Figure 15 (Test Set II). Because the regret values of both SOPR-T and SOPR-MLP in Test Set II in all the tasks are zero, the regret value results are not presented and we only show rank correlation results in Figure 15. As can be seen from the results, SOPR-T outperforms SOPR-MLP in most tasks in terms of both rank correlation and regret value.



(a) (b) (c)

(d) (e) (f)

Figure 14. Performance comparison between SOPR-T and SOPR-MLP (Test Set I). Top row: rank correlation. Bottom row: regret@3.



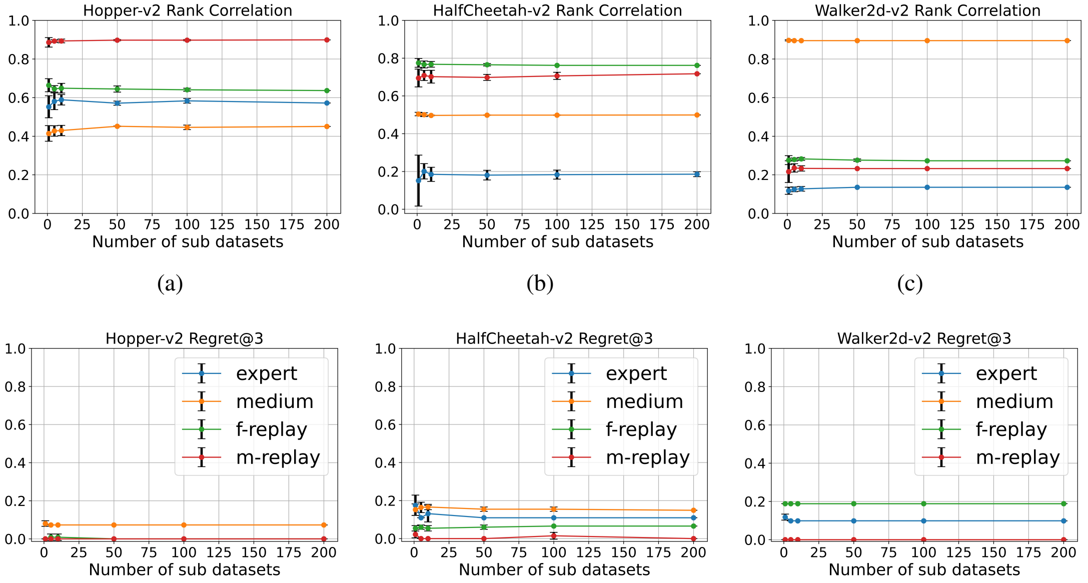
(a) (b) (c)

Figure 15. Performance comparison between SOPR-T and SOPR-MLP (Test Set II, rank correlation).

Effect of Using Different Number of Subsets of Data in Inference

In the last part of Section 4.4, we investigated the performance variance of our SOPR-T algorithm and the effect of using different amounts of subsets in inference. Here, Figure 16 shows the average results with std bar on all the tasks with different amounts of subsets. Figure 17 shows the detailed standard deviation results.

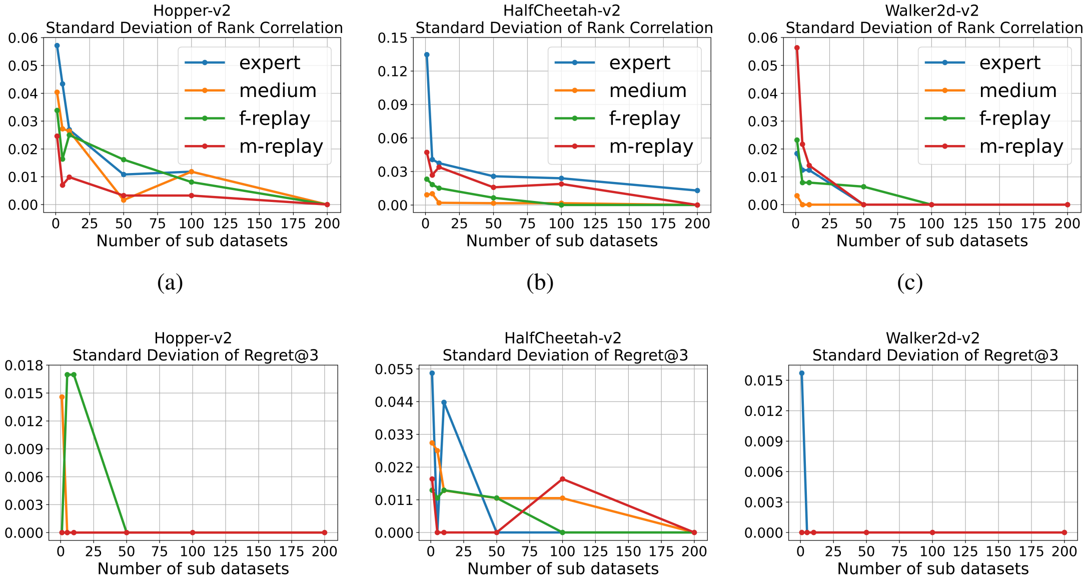
As can be seen from the results in Figure 16, the number of subsets makes little difference on the average performance of SOPR-T in all the tasks. Figure 17 indicates that the standard deviation decreases fast as the number of subsets increases. In addition, in almost all the tasks, the standard deviation is very small. Therefore, we can draw a conclusion that the performance of SOPR-T is stable even though it only uses small amount of data in inference. Further, the inference time cost can be reduced by using a small number of subsets in practice.



(a) (b) (c)

(d) (e) (f)

Figure 16. Performance of SOPR-T with different numbers of subsets in inference. Top row: rank correlation. Bottom row: regret@3.



(a) (b) (c)

(d) (e) (f)

Figure 17. Standard deviation of SOPR-T with different numbers of subsets in inference. Top row: rank correlation. Bottom row: regret@3.