Towards User-Interactive Offline Reinforcement Learning

Phillip Swazinna
Siemens Technology & TU Munich
swazinna@in.tum.de

Steffen Udluft
Siemens Technology
steffen.udluft@siemens.com

Thomas Runkler
Siemens Technology & TU Munich
thomas.runkler@siemens.com

Abstract

Offline reinforcement learning algorithms are still not fully trusted by practitioners due to the risk that the learned policy performs worse than the original policy that generated the dataset or behaves in an unexpected way that is unfamiliar to the user. At the same time, offline RL algorithms are not able to tune their arguably most important hyperparameter - the proximity of the learned policy to the original policy. We propose an algorithm that allows the user to tune this hyperparameter at runtime, thereby addressing both of the above mentioned issues simultaneously.

1 Introduction

Recently, offline reinforcement learning (RL) methods have shown that it is possible to learn effective policies from a static pre-collected dataset instead of directly interacting with the environment (Laroche et al., 2019; Fujimoto et al., 2019; Kumar et al., 2019; Swazinna et al., 2021b). Since direct interaction is in practice usually very costly, these techniques have alleviated a large obstacle on the path of deploying reinforcement learning policies in real world problems.
A major issue that these algorithms still face is tuning their arguably most important hyperparameter: The proximity to the original policy. Virtually all algorithms tackling the offline setting have such a hyperparameter, and it is obviously hard to tune, since no interaction with the real environment is permitted until final deployment. Practitioners thus risk being overly conservative (resulting in no improvement) or overly progressive (risking worse performing policies due to extrapolation error) in their choice.

Additionally, one of the largest obstacles on the path to deployment of RL trained policies in physical industrial control problems is that (offline) RL algorithms ignore the presence of domain experts. We argue that it is important to provide these users with a utility - something that makes them want to use RL solutions in order to facilitate deployments. Other research fields, such as machine learning for medical diagnoses, have already established the idea that domain experts are crucially important to solve the task and complement human users in various ways Babbar et al. (2022); Cai et al. (2019); De-Arteaga et al. (2021). We see our work in line with these and others (Shneiderman, 2020; Schmidt et al., 2021), who suggest that the next generation of AI systems needs to adopt a user-centered approach and develop systems that behave more like an intelligent tool, combining both high levels of human control and high levels of automation. We seek to develop an offline RL method that does just that.

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2022

We thus propose a simple algorithm to address both issues: We train policies conditioned on the proximity to the original policy, so that users can during deployment still adapt the trade-off between proximity and estimated performance. Close proximity to a known solution naturally facilitates trust, enabling conservative users to choose behavior they are more inclined to confidently deploy. Since we observe that the performance as a function of the algorithm's hyperparameter is mostly smooth, we find that users can use it to tune the hyperparameter to a degree.

We argue that the approach not only facilitates trust and enables hyperparameter tuning, but also has the potential to greatly increase deployment efficiency since many behaviors can be explored by the user in a single deployment.
We compare our method's performance with a variety of offline RL baselines and show that a user can achieve state of the art performance with it. Furthermore, we show that our method has advantages over simpler approaches like training many policies with diverse hyperparameters and give an example how a user could potentially use it for hyperparameter tuning.

2 LION: Learning in Interactive Offline eNvironments

In this work, we address two issues of the offline RL setting: First, we would like to provide the user with a high level control option in order to influence the behavior of the policy after deployment, since we argue that the user is crucially important for solving the task. Further we address the issue that in offline RL, the correct hyperparameter controlling the trade-off between pessimism and performance is unknown and can hardly be tuned. By training a policy conditioned in the proximity hyperparameter, we aim to enable the user to find a good trade-off hyperparameter. Crucially, we include the choice of 100% proximity, i.e. behavior cloning, so that the user is guaranteed to have an option that is known and not worse than before. He or she may then slowly move towards better solutions. We name our approach LION (Learning in Interactive Offline eNvironments).
While offline policy evaluation (Hans et al., 2011; Paine et al., 2020; Konyushova et al., 2021; Zhang et al., 2021; Fu et al., 2021) could help at least with the second issue, we do not use it here since it doesn't help us achieving control options for the user.

2.1 Training

During training time, we optimize three components: A model of the original policy $\beta_\phi(s)$, an ensemble of transition dynamics models $\{f^i_{\psi_i}(s,a)|i \in 0,\dots,N-1\}$, as well as the user adaptive policy $\pi_\theta(s,\lambda)$. The dynamics models $\{f^i\}$ as well as the original policy β are trained in isolation before the actual policy training starts. Both π and β are always simple feedforward neural networks which map states directly to actions in a deterministic fashion (practitioners likely favor deterministic policies over stochastic ones due to trust issues). β is trained to simply imitate the behavior present in the dataset by minimizing the mean squared distance to the observed actions:

$$L(\phi) = \frac{1}{N} \sum_{s_t,a_t \sim \mathcal{D}} \left[a_t - \beta_\phi(s_t)\right]^2 \tag{1}$$

Depending on the environment, the transition models are either also feedforward networks or simple recurrent networks with a single recurrent layer. The recurrent networks build their hidden state over G steps and are then trained to predict a window of size F into the future (similarly to (Hein et al., 2017b)), while the feedforward dynamics simply predict single step transitions. Both use mean squared error as loss:

$$L(\psi_i) = \frac{1}{N} \sum_{s_t,a_t,s_{t+1} \sim \mathcal{D}} \left[s_{t+1} - f^i_{\psi_i}(s_t,a_t)\right]^2 \tag{2}$$

$$L(\psi_i) = \frac{1}{N} \sum_{t \sim \mathcal{D}} \sum_{f=1}^{F} [s_{t+G+f+1} - f(s_t, a_t, \dots s_{t+G}, a_{t+G}, \dots \hat{s}_{t+G+f}, a_{t+G+f})]^2 \tag{3}$$

where $\hat{s}_{t+H+f}$ are the model predictions that are fed back to be used as input again. For simplicity, in this notation we assume the reward to be part of the state. Also we do not explicitly show the

recurrence and carrying over of the hidden states.
After having trained the two components $\beta_\phi(s)$ and $\{f^i_{\psi_i}(s,a)\}$, we can then move on to policy training. Similarly to MOOSE (Swazinna et al., 2021b), we optimize the policy π_θ by sampling start states from $\mathcal{D}$ and performing virtual rollouts throughout the dynamics ensemble using the current policy candidate. In every step, the ensemble predicts the reward as the minimum among its members and the next state that goes with it. At the same time we collect the mean squared differences between the actions that π_θ took in the rollout and the one that β_ϕ would have taken. The loss is then computed as a weighted sum of the two components. Crucially, we sample the weighting factor λ randomly and pass it to the policy as an additional input - the policy thus needs to learn all behaviors ranging from pure behavior cloning to entirely free optimization:

$$L(\theta) = -\sum_{s_0 \sim \mathcal{D}} \sum_{t}^{T} \gamma^t [\lambda e(s_t, a_t) - (1-\lambda) p(a_t)] \tag{4}$$

where we sample λ between 0 & 1, $a_t = \pi_\theta(s_t, \lambda)$ are the actions proposed by the policy, $p(a_t) = [\beta_\psi(s_t) - a_t]^2$ denotes the penalty based on the squared distance between the original policy and the actions proposed by π_θ and $e(s_t, a_t) = \min\{r(f^i_{\psi^i}(s_t, a_t)) | i \in 0, ..., N-1\}$ denotes the output of the ensemble prediction for reward (we omit explicit notation of recurrence for simplicity). Furthermore, during training, we do not sample λ uniformly between zero and one, but instead choose a beta distribution with parameters $(0.1, 0.1)$. Similarly to Seo et al. (2021), we find that it is important to put emphasis on the edge cases, so that the extreme behavior is properly learned. The interpolation between them seems to be easier and thus require less samples.

2.2 Deployment

At deployment time, the trained policy can at any point be influenced by the user that would otherwise be in control of the system, by choosing the λ that is passed to the policy together with the current system state to obtain an action:

$$a_t = \pi_\theta(s_t, \lambda) \quad \lambda \in \mathrm{User}(\mathrm{s_t}). \tag{5}$$

He or she may choose to be conservative or adventurous, observe the feedback and always adjust the proximity parameter of the policy accordingly. At this point, any disliked behavior can immediately be corrected without any time loss due to re-training and deploying a new policy.
We propose to initially start with $\lambda = 0$ during deployment, in order to check whether the policy is actually able to reproduce the original behavior and to gain the user's trust in the found solution. Then, depending on how critical failures are and how much time is at hand, λ may be increased in small steps for as long as the user is still comfortable with the observed behavior. Figure 2 shows an example of how the policy behavior changes over the course of λ. Once the performance stops to increase (or the user is otherwise not satisfied), we can immediately return to the last satisfying λ value.

3 Experiments

At first, we intuitively showcase LION in a simple 2D-world in order to get an understanding of how the policy changes its behavior based on λ. Afterwards, we move to a more serious test, evaluating our algorithm on the 16 industrial benchmark (IB) datasets (Hein et al., 2017a; Swazinna et al., 2021b).

3.1 2D-World

We evaluate the LION approach on a simplistic 2D benchmark. The states are x & y coordinates in the environment and rewards are given based on the position of the agent, following a Gaussian distribution around a fixed point in the state space, i.e.

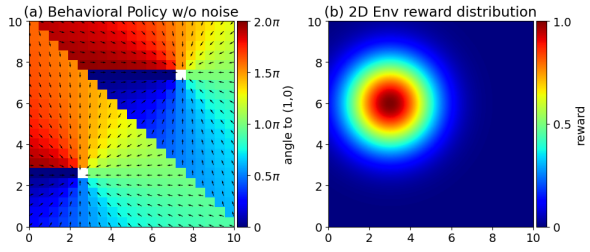

Figure 1: (a) Original baseline policy for data collection and (b) reward distribution in the 2D test environment

$r(s_t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5((s_t-\mu)/\sigma)^2}$. In this example we set $\mu = (3, 6)^{\mathrm{T}}$ and $\sigma = (1.5, 1.5)^{\mathrm{T}}$. A visualization of the reward distribution can be seen in Fig. 1 (b). We collect data from the environment using a simple policy that moves either to position $(2.5, 2.5)^{\mathrm{T}}$ or to $(7.5, 7.5)^{\mathrm{T}}$, depending on which is closer from the randomly drawn start state (shown in Fig. 1(a)), adding $\varepsilon = 10\%$ random actions as exploration. Then we follow the outlined training procedure to train a proximity conditioned policy. Fig. 2 shows policy maps for $\lambda \in \{0.0, 0.4, 0.6, 0.65, 0.7, 0.85, 1.0\}$, moving from simply imitating the original policy, over different mixtures, to purely optimizing for return. Since the task is easy and accurately modeled by the dynamics ensemble, it is no problem to give absolute freedom to the policy and optimize for return only. As it can be seen, the policy moves quickly to the center of the reward distribution for $\lambda = 1$.

3.2 Industrial Benchmark

We evaluate LION on the industrial benchmark (IB) datasets initially proposed in Swazinna et al. (2021b). The 16 datasets are created with three different baseline original policies (*optimized*, *mediocre*, *bad*) mixed with varying degrees of exploration (0-100%). Together, they constitute a diverse set of offline RL settings.

We compare performances of the LION policy with various state of the art offline RL baselines, such as BEAR, BRAC, BCQ, CQL, TD3+BC, MOOSE, WSBC, MOPO & MOReL Kumar et al. (2019); Wu et al. (2019); Fujimoto et al. (2019); Kumar et al. (2020); Fujimoto & Gu (2021); Swazinna et al. (2021b,a); Yu et al. (2020); Kidambi et al. (2020), in order to test whether the policies are able to provide state of the art performance anywhere in the λ range. Fig. 3 shows a subset of the results (one dataset for each baseline as well as the 100% exploration dataset), see Appendix A for all of them. We find that the performance curves are rather well behaved, meaning they do not switch between going up an down often. Rather, there is usually a single maximum before which the performance is rising and after which the performance is strictly dropping. This is a very desirable characteristic for usage in the user interactive setting, as it enables users to much more easily navigate the performance landscape and find a well performing λ value for the policy. In 13 out of 16 datasets, users can thus match or outperform the current state of the art method on that dataset, and achieve close to on-par performance on the remaining three. The distance-to-original policy curves are even monotonously increasing from start to finish, enabling practitioners to find a solution he or she is still comfortable with in terms of distance to the familiar original policy.

We briefly show that the smoothness of the performance curves is not something that can simply be expected from other offline RL algorithms: In Fig. 4, we train MOPO for a variety of different pessimism parameters and record its deployment performance. Somewhat surprisingly, the resulting return as a function of the hyperparameter is extremely wiggly, making parameter selection much harder for a practitioner. It appears that the representation of many policies in a single network performed by LION forces the behaviors to smoothly interpolate among each other, see Appendix B for details.

We find that effectively, LION takes much of the risk that users normally assume in offline RL away due the possibility to start with what has been previously known and the smooth performance

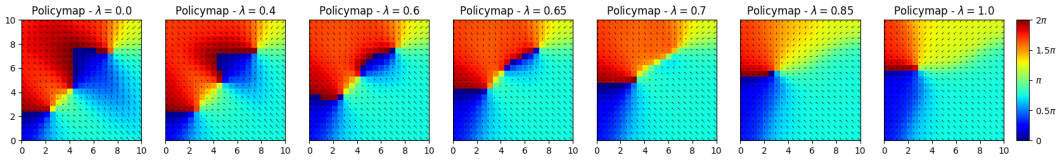

Figure 2: Policy maps for increasing values of λ in the 2D environment. Initially, the policy simply imitates the original policy (see Fig. 1 (a)). With increased freedom, the policy moves less to the upper right and more to the bottom left goal state of the original policy, since that one is closer to the high rewards. Then, the policy moves its goal slowly upwards on the y-axis until it is approximately at the center of the reward distribution. Since enough data was available (1,000 interactions) and the environment so simple, the models capture the true dynamics well and the optimal solution is found at $\lambda = 1$. This is however not necessarily the case if not enough or not the right data was collected (e.g. due to a suboptimal original policy - see Fig. 3, e.g. bad-0.4 or mediocre-0.0).

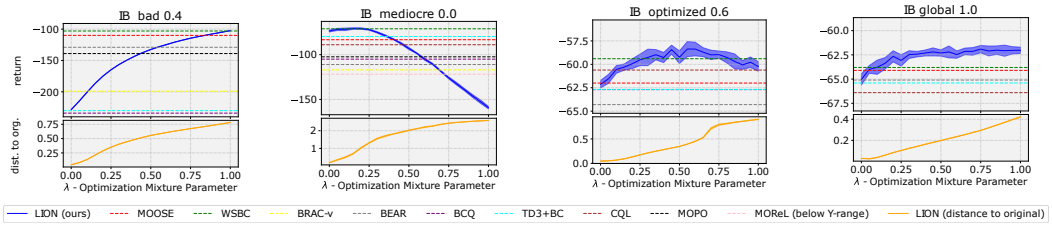

Figure 3: Evaluation performance and distance to the original policy be of the LION approach over the chosen λ hyperparameter. Various state of the art baselines are added as dashed lines with their standard set of hyperparameters (results from Swazinna et al. (2022)). Even though the baselines all exhibit some hyperparameter that controls the distance to the original policy, all are implemented differently and we can neither map them to a corresponding lambda value of our algorithm, nor change the behavior at runtime, which is why we display them as dashed lines over the entire lambda spectrum.

curves. With prior algorithms, users always faced the risk that the chosen hyperparameter turned out not to be favorable for their dataset - a few examples: WSBC produces state of the art results for many of the IB datasets, however for mediocre-0.6 it produces a catastrophic -243 (original performance is -75), likely destroying any user's trust in the method. Similarly, CQL is the prior best method on optimized-0.8, however the same method produces a performance of -292 on bad-0.2 (MOPO, MOOSE, and WSBC get between -110 & -130). The same hyperparameters obviously don't work well on all considered datasets, which is no wonder since they range from 0% (needs a lot of pessimism) up to 100% exploration (needs next to no pessimism). With LION, the user is able to find a fitting hyperparameter after deployment on all of them.

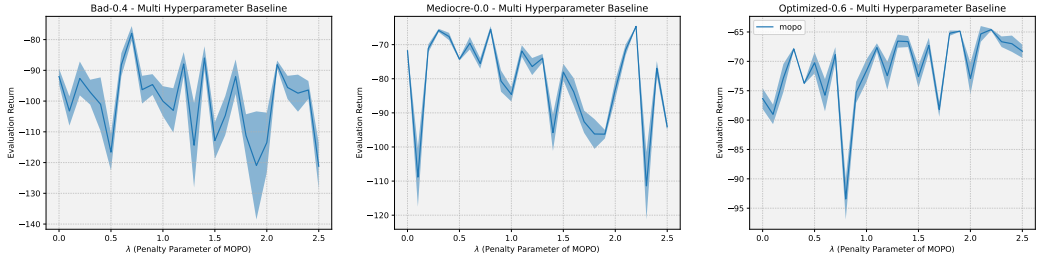

Figure 4: Previous offline RL algorithms such as MOPO may behave unsteadily when trained over a range of penalizing hyperparameters.

4 Conclusion

In this work we presented a novel offline RL approach that, to the best of our knowledge, is the first to let the user adapt the proximity/performance trade-off after training is finished. We let the user tune the behavior by allowing him to choose the desired proximity to the original policy, in an attempt to solve two issues: (1) Giving users a high level control option to provide them with an incentive to use (offline) RL. (2) The problem that practitioners cannot tune the hyperparameter in prior offline RL algorithms.
With LION, users should get at least the original performance when they start at $\lambda = 0$, minimizing deployment risk. Practitioners can then find better solutions (if existing) on all the considered datasets due to the smooth behavior interpolation, alleviating problems related to trust and risk of deployment for offline RL policies in practice.

References

Babbar, V., Bhatt, U., and Weller, A. On the utility of prediction sets in human-AI teams. *arXiv preprint arXiv:2205.01411*, 2022.

Cai, C. J., Reif, E., Hegde, N., Hipp, J., Kim, B., Smilkov, D., Wattenberg, M., Viegas, F., Corrado, G. S., Stumpe, M. C., et al. Human-centered tools for coping with imperfect algorithms during medical decision-making. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pp. 1–14, 2019.

De-Arteaga, M., Dubrawski, A., and Chouldechova, A. Leveraging expert consistency to improve algorithmic decision support. *arXiv preprint arXiv:2101.09648*, 2021.

Fu, J., Norouzi, M., Nachum, O., Tucker, G., Wang, Z., Novikov, A., Yang, M., Zhang, M. R., Chen, Y., Kumar, A., et al. Benchmarks for deep off-policy evaluation. *arXiv preprint arXiv:2103.16596*, 2021.

Fujimoto, S. and Gu, S. S. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pp. 2052–2062. PMLR, 2019.

Hans, A., Duell, S., and Udluft, S. Agent self-assessment: Determining policy quality without execution. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 84–90. IEEE, 2011.

Hein, D., Depeweg, S., Tokic, M., Udluft, S., Hentschel, A., Runkler, T. A., and Sterzing, V. A benchmark environment motivated by industrial control problems. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8. IEEE, 2017a.

Hein, D., Udluft, S., Tokic, M., Hentschel, A., Runkler, T. A., and Sterzing, V. Batch reinforcement learning on the industrial benchmark: First experiences. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 4214–4221. IEEE, 2017b.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

Konyushova, K., Chen, Y., Paine, T., Gulcehre, C., Paduraru, C., Mankowitz, D. J., Denil, M., and de Freitas, N. Active offline policy selection. *Advances in Neural Information Processing Systems*, 34, 2021.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. Stabilizing off-policy Q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pp. 11761–11771, 2019.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Laroche, R., Trichelair, P., and Des Combes, R. T. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pp. 3652–3661. PMLR, 2019.

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. *arXiv preprint arXiv:2007.09055*, 2020.

Schmidt, A., Giannotti, F., Mackay, W., Shneiderman, B., and Väänänen, K. Artificial intelligence for humankind. In *IFIP Conference on Human-Computer Interaction*, pp. 335–339. Springer, 2021.

Seo, S., Arik, S., Yoon, J., Zhang, X., Sohn, K., and Pfister, T. Controlling neural networks with rule representations. *Adv. in Neural Information Processing Systems*, 34, 2021.

Shneiderman, B. Human-centered artificial intelligence: three fresh ideas. *AIS Transactions on Human-Computer Interaction*, 12(3):109–124, 2020.

Swazinna, P., Udluft, S., Hein, D., and Runkler, T. Behavior constraining in weight space for offline reinforcement learning. *arXiv preprint arXiv:2107.05479*, 2021a.

Swazinna, P., Udluft, S., and Runkler, T. Overcoming model bias for robust offline deep reinforcement learning. *Engineering Applications of Artificial Intelligence*, 104:104366, 2021b.

Swazinna, P., Udluft, S., Hein, D., and Runkler, T. Comparing model-free and model-based algorithms for offline reinforcement learning. *ICONS*, 2022.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. MOPO: Model-based offline policy optimization. In *Advances in Neural Information Processing Systems*, volume 33, pp. 14129–14142, 2020.

Zhang, M. R., Paine, T. L., Nachum, O., Paduraru, C., Tucker, G., Wang, Z., and Norouzi, M. Autoregressive dynamics models for offline policy evaluation and optimization. *arXiv preprint arXiv:2104.13877*, 2021.

A Complete Industrial Benchmark Experiments

Datasets We evaluate LION on the industrial benchmark datasets initially proposed in (Swazinna et al., 2021b). The 16 datasets are created with three different baseline original policies (*optimized*, *mediocre*, *bad*) mixed with varying degrees of exploration. The *optimized* baseline is an RL trained policy and simulates an expert practitioner. The *mediocre* baseline moves the system back and forth around a fixed point that is rather well behaved, while the *bad* baseline steers to a point on the edge of the state space in which rewards are deliberately bad. Each baseline is combined with $\varepsilon \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$-greedy exploration to collect a dataset (making the $\varepsilon = 0.0$ datasets extreme cases of the narrow distribution problem). Together, they constitute a diverse set of offline RL settings. The exact baseline policies are given by:

$$\pi_{\text{bad}} = \begin{cases} 100 - v_t \\ 100 - g_t \\ 100 - h_t \end{cases} \qquad \pi_{\text{med}} = \begin{cases} 25 - v_t \\ 25 - g_t \\ 25 - h_t \end{cases} \qquad \pi_{\text{opt}} = \begin{cases} -\tilde{v}_{t-5} - 0.91 \\ 2\tilde{f}_{t-3} - \tilde{p} + 1.43 \\ -3.48\tilde{h}_{t-3} - \tilde{h}_{t-4} + 2\tilde{p} + 0.81 \end{cases}$$

The datasets contain 100,000 interactions collected by the respective baseline policy combined with the ε-greedy exploration. The IB is a high dimensional and partially observable environment - if access to the full Markov state were provided, it would contain 20 state variables. Since only six of those are observable, and the relationship to the other variables and their subdynamics are complex and feature heavily delayed components, prior work Hein et al. (2017b) has stated that up to 30 past time steps are needed to form a state that can hope to recover the true dynamics, so the state can be considered 180 dimensional. In our case we thus set the number of history steps $G = 30$. The action space is 3 dimensional. The benchmark is not supposed to mimic a single industrial application, but rather exhibit common issues observable in many different applications (partial observability, delayed rewards, multimodal and heteroskedastic noise, ...). The reward is a weighted combination of the observable variables fatigue and consumption, which are conflicting (usually move in opposite directions and need trade-off) and are influenced by various unobservable variables. As in prior work Swazinna et al. (2021b) we optimize for a horizon of 100. The datasets are available at `https://github.com/siemens/industrialbenchmark/tree/offline_datasets/datasets` under the Apache License 2.0.

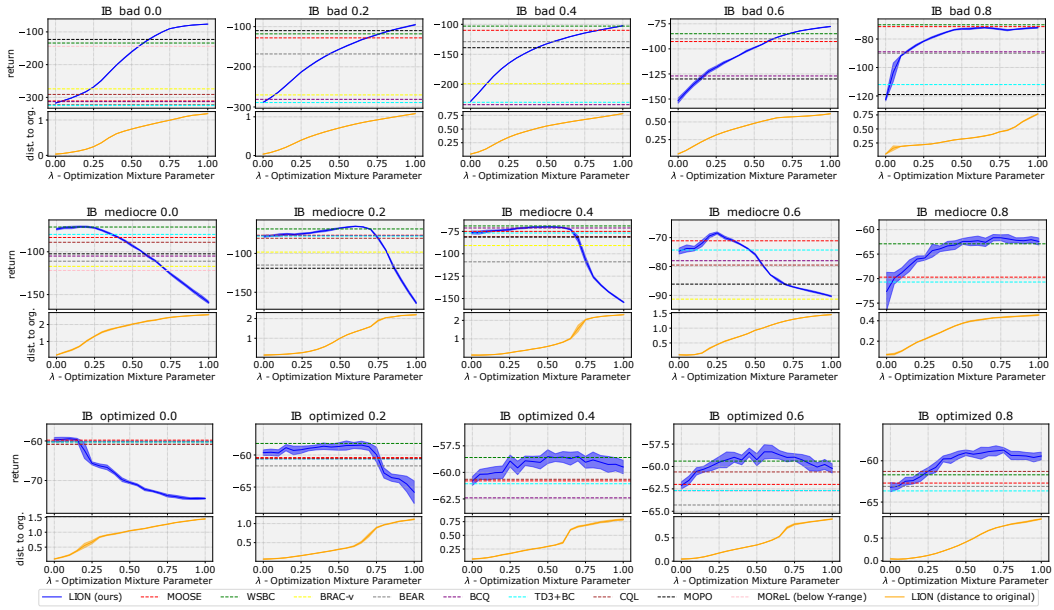

Figure 5: Evaluation performance (top portion of each graph) and distance to the original policy (lower portion of each graph) of the LION approach over the chosen λ hyperparameter. Various state of the art baselines are added as dashed lines with their standard set of hyperparameters (results from (Swazinna et al., 2022)). Even though the baselines all exhibit some hyperparameter that controls the distance to the original policy, all are implemented differently and we can neither map them to a corresponding lambda value of our algorithm, nor change the behavior at runtime, which is why we display them as dashed lines over the entire λ-spectrum. See Fig. 3 for the 100% exploration dataset.

Baselines We compare performances of LION with various state of the art offline RL baselines:

- BEAR, BRAC, BCQ, CQL and TD3+BC (Kumar et al., 2019; Wu et al., 2019; Fujimoto et al., 2019; Kumar et al., 2020; Fujimoto & Gu, 2021) are model-free algorithms. They mostly regularize the policy by minimizing a divergence to the original policy. BCQ samples only likely actions and CQL searches for a Q-function that lower bounds the true one.
- MOOSE and WSBC (Swazinna et al., 2021a,b) are purely model based algorithms that optimize the policy via virtual trajectories through the learned transition model. MOOSE penalizes reconstruction loss of actions under the original policy (learned by an autoencoder), while WSBC constrains the policy directly in weight space. MOOSE is from the policy training perspective the closest to our LION approach.
- MOPO and MOReL (Yu et al., 2020; Kidambi et al., 2020) are hybrid methods that learn a transition model as well as a value function. Both use the models to collect additional data and regularize the policy by means of model uncertainty. MOPO penalizes uncertainty directly, while MOReL simply stops episodes in which future states become too unreliable. MOReL uses model-disagreement and MOPO Gaussian outputs to quantify uncertainty.

Evaluation In order to test whether the trained LION policies are able to provide state of the art performance anywhere in the λ range, we evaluate them for λ from 0 to 1 in many small steps. Fig. 5 show results for the IB datasets. We find that the performance curves do not exhibit many local optima. Rather, there is usually a single maximum before which the performance is rising and after which the performance is strictly dropping. This is a very desirable characteristic for usage in the user interactive setting, as it enables users to easily find the best performing λ value for the policy. In 13 out of 16 datasets, users can thus match or outperform the current state of the art method on that dataset, and achieve close to on-par performance on the remaining three. The distance-to-original-policy curves are even monotonously increasing from start to finish, making it possible for the practitioner to find the best solution he or she is still comfortable with in terms of distance to the familiar behavior.

B Discrete Baseline

A simpler approach than LION might be to train an existing offline RL algorithm for many trade-offs in advance, to provide at least discrete options. Two downsides are obvious: (a) we wouldn't be able to handle the continuous case, i.e. when a user wants a trade-off that lies between two discrete policies, and (b) the computational cost increases linearly with the number of policies trained. We show that a potentially even bigger issue exists in Figure 4: When we train a discrete collection of policies with different hyperparameters, completely independently of each other, they often exhibit wildly different behaviors even when the change in hyperparameter was small. LION instead expresses the collection as a single policy network, training them jointly and thus forcing them to smoothly interpolate among each other. This helps to make the performance a smooth function of the hyperparameter (although this must not always be the case) and results in a performance landscape that is much easier to navigate for a user searching for a good trade-off.