Data-Efficient Pipeline for Offline Reinforcement Learning with Limited Data

Allen Nie* **Yannis Flet-Berliac** **Deon R. Jordan**
William Steenbergen **Emma Brunskill**
Department of Computer Science
Stanford University
*anie@stanford.edu

Abstract

Offline reinforcement learning (RL) can be used to improve future performance by leveraging historical data. There exist many different algorithms for offline RL, and it is well recognized that these algorithms, and their hyperparameter settings, can lead to decision policies with substantially differing performance. This prompts the need for pipelines that allow practitioners to systematically perform algorithm-hyperparameter selection for their setting. Critically, in most real-world settings, this pipeline must only involve the use of historical data. Inspired by statistical model selection methods for supervised learning, we introduce a task- and method-agnostic pipeline for automatically training, comparing, selecting, and deploying the best policy when the provided dataset is limited in size. In particular, our work highlights the importance of performing multiple data splits to produce more reliable algorithm-hyperparameter selection. While this is a common approach in supervised learning, to our knowledge, this has not been discussed in detail in the offline RL setting. We show it can have substantial impacts when the dataset is small. Compared to alternate approaches, our proposed pipeline outputs higher-performing deployed policies from a broad range of offline policy learning algorithms and across various simulation domains in healthcare, education, and robotics. This work contributes toward the development of a general-purpose meta-algorithm for automatic algorithm-hyperparameter selection for offline RL.

1 Introduction

Offline/batch reinforcement learning has the potential to learn better decision policies from existing real-world datasets on sequences of decisions made and their outcomes. In many of these settings, tuning methods online is infeasible and deploying a new policy involves time, effort and potential negative impact. Many of the existing datasets for applications that may benefit from offline RL may be fairly small in comparison to supervised machine learning. For instance, the MIMIC intensive care unit dataset on sepsis that is often studied in offline RL has 14k patients (Komorowski et al., 2018), the number of students frequently interacting with an online course will often range from hundreds to tens of thousands (Bassen et al., 2020), and the number of demonstrations collected from a human operator manipulating a robotic arm is often on the order of a few hundred per task (Mandlekar et al., 2018). In these small data regimes, recent studies (Mandlekar et al., 2021; Levine et al., 2020) highlight that with limited data, the selection of hyperparameters using the training set is often challenging. Yet hyperparameter selection also has a substantial influence on the resulting policy's performance, particularly when when the algorithm leverages deep neural networks.

One popular approach to address this is to learn policies from particular algorithm-hyperparameter pairs on a training set and then use offline policy selection, which selects the best policy given a validation set (Thomas et al., 2015a, 2019; Paine et al., 2020; Kumar et al., 2021). However, when

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Common Practices	Non-Markov Env	Data Efficient (re-train)	Compare Across OPL	Considers Evaluation Variation	Considers Training Variation
Policy selection (1 split)					
Internal Objective / TD-Error (Thomas et al., 2015b, 2019)	(depends)	✗	✗	✗	✗
OPE methods (Komorowski et al. (2018); Paine et al. (2020)	(depends)	✗	✓	✗	✗
OPE + BCa Val. (Thomas et al., 2015b)	(depends)	✗	✓	✓	✗
BVFT (Xie and Jiang, 2021)	✗	✗	✗	✗	✗
BVFT + OPE (Zhang and Jiang, 2021)	✗	✗	✓	✓	✗
Q-Function Workflow (Kumar et al., 2021)	✗	✓	✗	✗	✗
Ours: $\mathcal{A}_i$ selection (multi-split)					
Cross-Validation	✓	✓	✓	✓	✓
Repeated Random Subsampling	✓	✓	✓	✓	✓

Table 1: A summary of commonly used approaches for choosing a deployment policy from a fixed offline RL dataset. We define *Data Efficient* as: the approach assumes the algorithm can be re-trained on all data points; *(depends)* as: depends on whether the underlying OPL or OPE methods make explicit Markov assumption or not.

the dataset is limited in size, this approach can be limited: (a) if the validation set happens to have no or very few good/high-reward trajectories, then trained policies cannot be properly evaluated; (b) if the training set has no or very few such trajectories, then no good policy behavior can be learned through any policy learning algorithm; and (c) using one fixed training dataset is prone to overfitting the hyperparameters on this one dataset and different hyperparameters could be picked if the training set changes. One natural solution to this problem is to train on the entire dataset and compare policy performance on the same dataset, which is often referred to as the internal objective approach. In Appendix A.1 we conduct a short experiment using D4RL where this approach fails due to the common issue of Q-value over-estimation (Fujimoto et al., 2019).

There has been much recent interest in providing more robust methods for offline RL. Many rely on the workflow just discussed, where methods are trained on one dataset and Offline Policy Evaluation (OPE) is used to do policy selection (Su et al., 2020; Paine et al., 2020; Zhang and Jiang, 2021; Kumar et al., 2021; Lee et al., 2021; Tang and Wiens, 2021; Miyaguchi, 2022). Our work highlights the impact of a less studied issue: the challenge caused by data partitioning variance. We first motivate the need to account for train/validation partition randomness by showing the wide distribution of OPE scores the same policy can obtain with different subsets of data or the very different performance results the same algorithm and hyperparameters can have depending on the training set partition. We also prove a single partition can have a notable failure rate in identifying the best algorithm-hyperparameter.

We then introduce a general pipeline for algorithm-hyperparameters (`AH`) selection and policy deployment that: (a) uses repeated random sub-sampling (RRS) with replacement of the dataset to perform `AH` training, (b) uses OPE on the validation set, (c) computes aggregate statistics over the RRS splits to inform `AH` selection, and (d) allows to use the selected `AH` to retrain on the entire dataset to obtain the deployment policy. Though such repeated splitting is common in supervised learning, its impact and effect have been little studied in the offline RL framework. Perhaps surprisingly, we show that our simple pipeline leads to substantial performance improvements in a wide range of popular benchmark tasks, including D4RL (Fu et al., 2020) and Robomimic (Mandlekar et al., 2021).

2 Related work

Offline Policy Learning (OPL). In OPL, the goal is to use historical data from a fixed behavior policy π_b to learn a reward-maximizing policy in an unknown environment (Markov Decision Process, defined in Section 3). Most work studying the sampling complexity and efficiency of offline RL (Xie and Jiang, 2021; Yin et al., 2021) do not depend on the structure of a particular problem, but empirical performance may vary with some pathological models that are not necessarily Markovian. Shi et al. (2020) have precisely developed a model selection procedure for testing the Markovian hypothesis and help explain different performance on different models and MDPs. To address this problem, it is inherently important to have a fully adaptive characterization in RL because it could save considerable time in designing domain-specific RL solutions (Zanette and Brunskill, 2019). As an answer to a variety of problems, OPL is rich with many different methods ranging from policy gradient (Liu et al., 2019), model-based (Yu et al., 2020; Kidambi et al., 2020), to model-free methods (Siegel et al., 2020; Fujimoto et al., 2019; Guo et al., 2020; Kumar et al., 2020) each based on different assumptions on the system dynamics. Practitioners thus dispose of an array of algorithms and corresponding hyperparameters with no clear consensus on a generally applicable evaluation tool for offline policy selection.

Offline Policy Evaluation (OPE). OPE is concerned with evaluating a target policy's performance using only pre-collected historical data generated by other (behavior) policies (Voloshin et al., 2021). Each of the many OPE estimators has its unique properties, and in this work, we primarily consider two main variants (Voloshin et al., 2021): Weighted Importance Sampling (WIS) (Precup, 2000) and Fitted Q-Evaluation (FQE) (Le et al., 2019). Both WIS and FQE are sensitive to the partitioning of the evaluation dataset. WIS is undefined on trajectories where the target policy does not overlap with the behavior policy and self-normalizes with respect to other trajectories in the dataset. FQE learns a Q-function using the evaluation dataset. This makes these estimators very different from mean-squared errors or accuracy in the supervised learning setting – the choice of partitioning will first affect the function approximation in the estimator and then cascade down to the scores they produce.

Offline Policy Selection (OPS). Typically, OPS is approached via OPE, which estimates the expected return of candidate policies. Zhang and Jiang (2021) address how to improve policy selection in the offline RL setting. The algorithm builds on the Batch Value-Function Tournament (BVFT) (Xie and Jiang, 2021) approach to estimating the best value function among a set of candidates using piece-wise linear value function approximations and selecting the policy with the smallest projected Bellman error in that space. Previous work on estimator selection for the design of OPE methods include Su et al. (2020); Miyaguchi (2022) while Kumar et al. (2021); Lee et al. (2021); Tang and Wiens (2021); Paine et al. (2020) focus on offline hyperparameter tuning. Kumar et al. (2021) give recommendations on when to stop training a model to avoid overfitting. The approach is exclusively designed for Q-learning methods with direct access to the internal Q-functions. On the contrary, our pipeline does policy training, selection, and deployment on any offline RL method, not reliant on the Markov assumption, and can select the best policy with potentially no access to the internal approximation functions (black box). We give a brief overview of some OPS approaches in Table 1.

3 Background and Problem Setting

We define a stochastic Decision Process $M = \langle \mathcal{S}, A, T, r, \gamma \rangle$, where $\mathcal{S}$ is a set of states; A is a set of actions; T is the transition dynamics (which might depend on the full history); r is the reward function; and $\gamma \in (0, 1)$ is the discount factor. Let $\tau = \{s_i, a_i, s'_i, r_i\}_{i=0}^{L}$ be the trajectory sampled from π on M. The optimal policy π is the one that maximizes the expected discounted return $V(\pi) = \mathbb{E}_{\tau \sim \rho_\pi}[G(\tau)]$ where $G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ and ρ_π is the distribution of τ under policy π. For simplicity, in this paper we assume policies are Markov $\pi : S \to A$, but it is straightforward to consider policies that are a function of the full history. In an offline RL problem, we take a dataset: $\mathcal{D} = \{\tau_i\}_{i=1}^{n}$, which can be collected by one or a group of policies which we refer to as the behavior policy π_b on the decision process M. The goal in offline/batch RL is to learn a decision policy π from a class of policies with the best expected performance V^π for future use. Let $\mathcal{A}_i$ to denote an `AH` pair, i.e. an offline policy learning algorithm and its hyperparameters and model architecture. An offline policy estimator takes in a policy π_e and a dataset $\mathcal{D}$, and returns an estimate of its performance: $\widehat{V} : \Pi \times \mathcal{D} \to \mathbb{R}$. In this work, we focus on two popular Offline Policy Evaluation (OPE) estimators: Importance Sampling (IS) (Precup, 2000) and Fitted Q-Evaluation (FQE) (Le et al., 2019) estimators. We refer the reader to Voloshin et al. (2021) for a more comprehensive discussion.

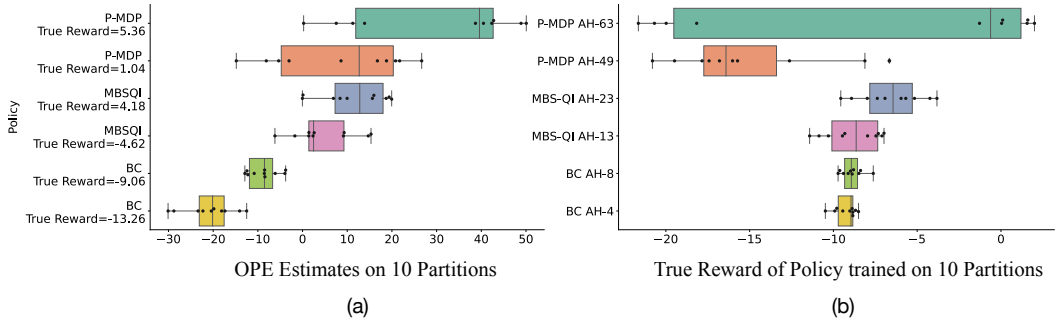

Figure 1: True performance and evaluation of 6 $\mathcal{A}_i$ pairs on the Sepsis-POMDP (N=1000) domain. (a) shows the OPE estimations and (b) shows the variation in terms of true performance. The variations are due to the different AH pairs of the policies *but also* to the sensitivity to the training/validation splits.

4 The Challenge of Offline RL $\mathcal{A}_i$ Selection

An interesting use-case of offline RL is when domain experts have access to an existing dataset (with potentially only a few hundred trajectories) about sequences of decisions made and respective outcomes, with the hope of leveraging the dataset to learn a better decision policy for future use. In this setting, the user may want to consider many options regarding the type of RL algorithm (model-based, model-free, or direct policy search), hyperparameter, or deep network architecture to use.

Automated algorithm selection is important because different $\mathcal{A}_i$ (different AH pairs) may learn very diverse policies, each with significantly different performance $V^{\mathcal{A}_i}$. Naturally, one can expect that various algorithms lead to diverse performance, but using a case-study experiment on a sepsis simulator (Oberst and Sontag, 2019), we observe in Figure 1(b) that the sensitivity to hyperparameter selection is also substantial (cf. different average values in box plots for each method). For example, MBS-QI (Liu et al., 2020) learns policies ranging from over -12 to -3 in their performance, depending on the hyperparameters chosen.

Precisely, to address hyperparameter tuning, past work often relies on executing the learned policies in the simulator/real environment. When this is not feasible, as in many real-world applications, including our sepsis dataset example, where the user may only be able to leverage existing historical data, we have no choice but to rely on off-policy evaluation. Prior work (Thomas et al., 2015b; Farajtabar et al., 2018; Thomas et al., 2019; Mandlekar et al., 2021) have suggested doing so using a hold-out method, after partitioning the dataset into training and validation sets.

Unfortunately, the partitioning of the dataset itself may result in substantial variability in the training process (Dietterich, 1998). We note that this problem is particularly prominent in offline RL where high-reward trajectories are sparse and affect both policy learning and policy evaluation. To explore this hypothesis, we consider the influence of the train/validation partition in the same sepsis domain, and we evaluate the trained policies using the Weighted Importance Sampling (WIS) (Precup, 2000) estimator. Figure 1(a) shows the policies have drastically different OPE estimations with sensitivity to randomness in the dataset partitioning. We can observe the same phenomena in Figure 1(b) with largely different true performances depending on the dataset splitting for most of the policies $\mathcal{A}_i$. This is also illustrated on the left sub-figure of Figure 4 where in the case where a single train-validation split is used, an $\mathcal{A}_i$ that yields lower-performing policies will often be selected over those that yield higher-performing policies when deployed.

4.1 Repeated Experiments for Robust Hyperparameter Evaluation in Offline RL

We now demonstrate why it is important to conduct repeated random sub-sampling on the dataset in offline RL. Consider a finite set of J offline RL algorithms $\mathcal{A}$. Let the policy produced by algorithm $\mathcal{A}_j$ on training dataset $\mathcal{D}$ be π_j, its estimated performance on a validation set $\hat{V}^{\pi_j}$, and its true (unknown) value be V^{π_j}. Denote the true best resulting policy as $\pi_{j^*} = \arg\max_j V^{\pi_j}$ and the corresponding algorithm $\mathcal{A}_{j^*}$. Let the best policy picked based on its validation set performance as $\pi_{\hat{j}^*} = \arg\max_j \hat{V}^{\pi_j}$ and the corresponding algorithm $\mathcal{A}_{\hat{j}^*}$.

Theorem 1. *There exist stochastic decision processes and datasets such that (i) using a single train/validation split procedure that selects an algorithm-hyperparameter with the best performance*

on the validation dataset will select a suboptimal policy and algorithm with significant finite probability, $P(\pi_{\hat{j}^*} \neq \pi_{j^*}) \geq C$, with corresponding substantial loss in performance $O(V_{max})$, and, in contrast, (ii) selecting the algorithm-hyperparameter with the best average validation performance across N_s train/validation splits will select the optimal algorithm and policy with probability 1: $\lim_{N_s \to \infty} P(\pi_{\hat{j}^*} = \pi_{j^*}) \to 1$.

Proof Sketch. Due to space constraints we defer the proof to Appendix A.3. Briefly, the proof proceeds by proof by example through constructing a chain-like stochastic decision process and considers a class of algorithms that optimize over differing horizons (see e.g. Jiang et al. (2015); Cheng et al. (2021); Mazoure et al. (2021)). The behavior policy is uniformly random meaning that trajectories with high rewards are sparse. This means there is a notable probability that in a single partition of the dataset, the resulting train and/or validation set may not contain a high reward trajectory, making it impossible to identify that a full horizon algorithm, and resulting policy, is optimal.

In the proof and our experiments, we focus on when the training and validation sets are of equal size. If we use an uneven split, such as $80/20\%$, the failure probability can further increase if only a single partition of the dataset is used. We provide an illustrative example in the Appendix. Note that Leave-one-out Cross-Validation (LooCV) will also fail in our setting if we employ, as we do in our algorithm, WIS, because as a biased estimator, WIS will return *the observed return of the behavior policy if averaging over a single trajectory, independent of the target policy to be evaluated*. We explain this further in Appendix A.10.

5 SSR: Repeated Random Sampling for $\mathcal{A}_i$ Selection and Deployment

In this paper, we are interested in the following problem: *If offline RL training and evaluation are very sensitive to the partitioning of the dataset, especially in small data regimes, how can we reliably produce a final policy that we are confident is better than others and can be reliably deployed in the real-world?*

Instead of considering the sensitivity to data partition as an inherent obstacle for offline policy selection, we view this as statistics to leverage for $\mathcal{A}_i$ selection. We propose a general pipeline: *Split Select Retrain* (SSR) (of which we provide a pseudo-code in Algorithm 1, Appendix A.4) to reliably optimize for a good deployed policy given only: an offline dataset, an input set of AH pairs and an off-policy evaluation (OPE) estimator. This deployment approach leverages the random variations created by dataset partitioning to select algorithms that perform better *on average* using a robust hyperparameter evaluation approach which we develop below.

First, we split and create different partitions of the input dataset. For each train/validation split, each algorithm-hyperparameter (AH) is trained on the training set and evaluated using the input OPE method to yield an estimated value on the validation set. These estimated evaluations are then averaged, and the best AH pair ($\mathcal{A}^*$) is selected as the one with the highest average score. Now the last step of the SSR pipeline is to re-use the entire dataset to train one policy π^* using $\mathcal{A}^*$.

Repeated Random Sub-sampling (RRS). As Theorem 1 suggests, one should ensure a sufficient amount of trajectories in the evaluation partition to lower the failure rate C. We propose to create RRS train-validation partitions. This approach has many names in the statistical model selection literature, such as Predictive Sample Reuse Method (Geisser, 1975), Repeated Learning-Test Method (Burman, 1989) or Monte-Carlo Cross-Validation (Dubitzky et al., 2007). It has also been referred to as Repeated Data Splitting (Chernozhukov et al., 2018) in the heterogeneous treatment effect literature. We randomly select trajectories in $\mathcal{D}$ and put them into into two parts: R^{train} and R^{valid}. We repeat this splitting process K times to generate paired datasets: $(R_1^{\text{train}}, R_1^{\text{valid}}), (R_2^{\text{train}}, R_2^{\text{valid}}), ..., (R_K^{\text{train}}, R_K^{\text{valid}})$. We compute the generalization performance estimate as follows:

$$\mathcal{G}_{\mathcal{A},\text{RS}_K} = \frac{1}{K} \sum_{k=1}^{K} \left[\hat{V}(\mathcal{A}(R_k^{\text{train}}); R_k^{\text{valid}}) \right] \tag{1}$$

A key advantage of overlap partitioning is that it maintains the size of the validation dataset as K increases. This might be favorable since OPE estimates are highly dependent on the state-action coverage of the validation dataset – the more data in the validation dataset, the better OPE estimators can evaluate a policy's performance. As $K \to \infty$, RRS approaches the leave-p-out cross-validation

(CV), where p denotes the number of examples in the validation dataset. Since there are $\binom{n}{p}$ possible selections of p data points out of n in our dataset, it is infeasible to use exact leave-p-out CV when $p > 2$, but a finite K can still offer many advantages. Indeed, Krzanowski and Hand (1997) point out that leave-p-out estimators will have lower variance compared to leave-one-out estimators, which is what the more commonly used M-fold cross-validation method converges to when $M = n - 1$. We discuss more in Appendix A.2.

6 Experiments

In this section, we answer the following questions: (a) how does the pipeline SSR-RRS compare to other methods? (b) does the proposed pipeline for $\mathcal{A}_i$ selection and policy deployment allow us to generate the best policy trained on the whole dataset? (c) does re-training on the whole dataset (data efficiency) generate better policies than policies trained on half of the dataset when $\mathcal{A}_i$ is selected by the pipeline? In addition, we conduct two ablation studies to answer to: what number of splits should we use for SSR-RRS and what is the impact of dataset size on the pipeline results?

6.1 Task/Domains

The experimental evaluation involves a variety of real-world and simulated domains, ranging from tabular settings to continuous control robotics environments. We evaluate the performance of SSR in selecting the best algorithm regardless of task domains and assumptions on task structure. We conduct experiments on eight datasets (Figure 2) from five domains (details in Appendix A.13), which we give a short description below, and use as many as 540 candidate AH pairs for the Sepsis POMDP domain.

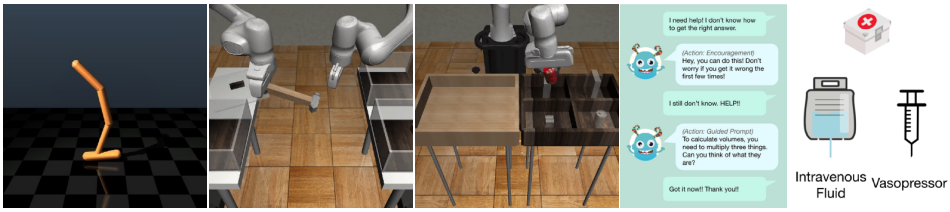

Figure 2: Illustrations from left to right of the D4RL, Robomimic, TutorBot and Sepsis domains.

Sepsis. The first domain is based on the simulator and work by Oberst and Sontag (2019) and revolves around treating sepsis patients. The goal of the policy for this simulator is to discharge patients from the hospital. In this domain, we experiment on two tasks: Sepsis-MDP and Sepsis-POMDP, a POMDP version of Sepsis-MDP.

TutorBot. The second domain includes a TutorBot simulator that is designed to support 3-5th grade elementary school children in understanding the concept of volume and engaging them while doing so. An online study was conducted using policy-gradient-based RL agent which interacted with about 200 students. We took the observations from this online study and built a simulator that reflects student learning progression, combined with some domain knowledge.

Robomimic. Robomimic (Mandlekar et al., 2021) is composed of various continuous control robotics environments with suboptimal human data. We use the Can-Paired and Transport dataset composed of 200 mixed quality human demonstrations. Mandlekar et al. (2021) attempted to use the RL objective loss on a 20% split validation set to select the best AH pair, but reported that the selected AH did not perform well in the simulator, which makes this task an interesting testbed for our pipeline.

D4RL. D4RL (Fu et al., 2020) is an offline RL standardized benchmark designed and commonly used to evaluate progress of offline RL algorithms. We use 3 datasets (200k samples each) with different qualities from the Hopper task: hopper-random from a randomly initialized policy, hopper-medium from a policy trained to approximately 1/3 the performance of a policy trained to completion with SAC ("expert"), and hopper-medium-expert from a 50-50 split of medium and expert data.

6.2 Baselines

One-Split OPE. The simplest method to train and verify an algorithm's performance without access to any simulator is to split the data into a train $\mathcal{D}_{\text{train}}$ and valid set $\mathcal{D}_{\text{valid}}$. All policies are trained

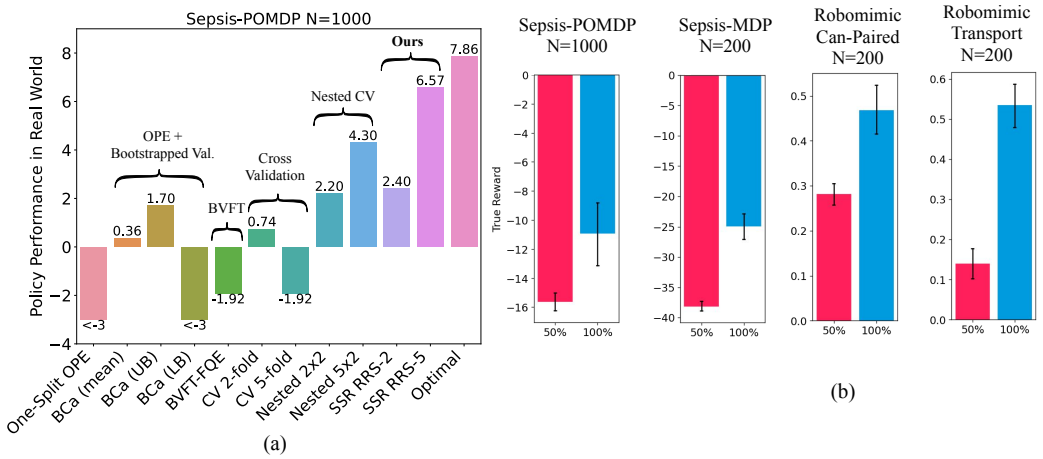

Figure 3: **(a)** We first compare our proposed pipeline to various other policy selection approaches in the Sepsis POMDP task. Our approach SSR-RRS 5-split consistently obtains policies that on average perform close to the optimal policy, significantly outperforming other approaches. **(b)** We investigate the importance of re-training in the small (N=200) to medium (N=1000) data regime. We show the true reward obtained by all policies from all AH pairs either trained on 50% of the data or 100% of the data. 95% confidence intervals are depicted as error bars. Most policies achieve higher rewards when trained on more data, even more so when the dataset is small or when tasks are more difficult (Robomimic).

on the same training set and evaluated on the same valid set. As we explained before, this method has the high potential of overfitting the chosen hyperparameter for one data partition – it might pick the best policy, but does not guarantee we can use the same hyperparameter to re-train on the full dataset.

OPE on Bootstrapped Val. Bootstrapping is a popular re-sampling technique that estimates prediction error in supervised learning models (Efron, 1983, 1986). The idea of using bootstrapping for OPE estimate is first utilized in HCOPE (Thomas et al., 2015b). Compared to the one-split method and the SSR pipeline, bootstrapping trains all policies on the same training dataset, and only considers variations in the validation set by creating bootstrapped samples. We refer to the considered Bias-corrected accelerated (BCa) bootstrap method as **BCa** in the experiments.

Cross-Validation (CV). One other natural alternative of repeated experiment validation is the popular M-fold Cross-Validation method (Stone, 1974). M-fold CV constructs a non-overlapping set of trajectories from the original dataset. For example, a 5-fold CV will train a policy on 80% of data and evaluate the policy on 20% of data, as it divides the dataset into 5 non-overlapping partitions. However, as we increase the number of splits M, which allows us to train/test our algorithms under more data split variations, each non-overlapping set $\mathcal{D}_m$ becomes smaller. When $M = n - 1$, M-fold CV becomes leave-one-out CV (LooCV). In this extreme case, many OPE estimators will not work properly, as we have shown in Appendix A.3. We further investigate a variant of M-fold CV called Nested M-fold CV (**Nested CV**), which repeats the M-fold non-overlapping partitioning K times. This procedure is computationally very expensive. Considering the fairness of comparison and computational efficiency, we only evaluate $K \times 2$-fold CV.

OPE with BVFT. Batch Value Function Tournament is the closest competitor to our method, which is a meta-algorithm for hyperparameter-free policy selection (Xie and Jiang, 2021; Zhang and Jiang, 2021). For a set of Q-value functions, BVFT does pairwise comparisons of each (tournament-style) to select the best out of the entire set based on the BVFT-Loss. Compared to our method, BVFT incurs $\mathcal{O}(J^2)$ comparison given J AH pairs, practically infeasible for large J. The original BVFT can only compare Q-functions, therefore only usable with OPL that directly learns Q-functions. Zhang and Jiang (2021) offers an extension to BVFT by using BVFT to compare between FQEs, therefore allowing BVFT to be OPL-agnostic. We adopt the two strategies recommended by the paper. Given J AH pairs and B FQEs, strategy 1 compares $J \times B$ FQE's Q-functions jointly (π **x FQE**) and strategy 2 compares B FQEs within each AH and pick the best FQE as the estimate of the AH's value estimate (π **+ FQE**). We discuss more in Appendix A.5 (calculations) and A.6 (time complexity).

6.3 Training and Evaluation

Offline Policy Learning. In the following, we outline a variety of Offline RL algorithms used in the evaluation of the SSR pipeline to demonstrate the generality of our approach and that it can reliably

Category	Algorithms	Internal Q-function
Imitation Learning	BC (Pomerleau, 1991) BCRNN (Mandlekar et al., 2018)	✗
Conservative Model-Free	BCQ (Fujimoto et al., 2019), CQL (Kumar et al., 2020), IRIS (Mandlekar et al., 2020)	✓
Policy Gradient	POIS (Metelli et al., 2018), BC+POIS (ours), BC+mini-POIS (ours)	✗
Offline Model-Based	MOPO (Yu et al., 2020), P-MDP (ours)	✓

Table 2: List of AH we are comparing in our experiments.

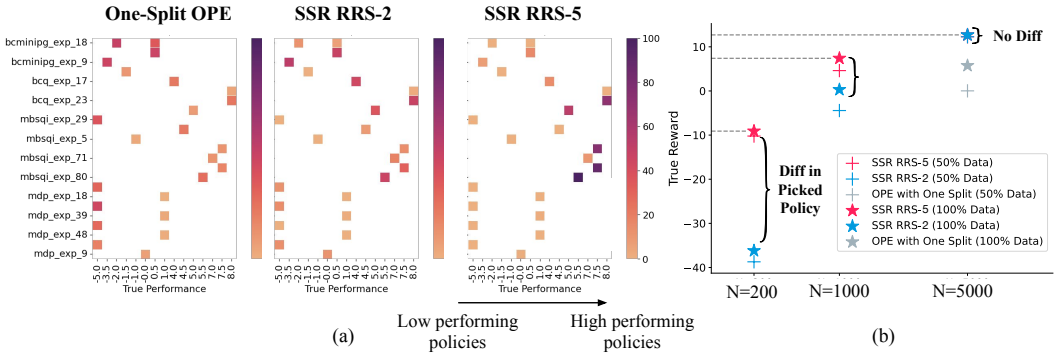

Figure 4: **(a)** We show the effect of choosing K (the number of train/valid splits) for SSR-RRS. We ablate K and run the simulation 500 times. The heatmaps shows the frequency at which each policy is chosen by our method and its final performance in the environment. In this experiment, SSR-RRS chooses over 540 AH pairs. As we can see, when the number of splits K is larger, SSR-RRS consistently pick better policies. **(b)** In the Sepsis-POMDP domain, we show that as the number of trajectories (N) in the offline dataset increases, data partitioning becomes less important. Though RRS still outperforms the 1-split policy selection.

produce the optimal final policy using the selected AH pair. This marks a departure from workflows designed for specific algorithms, such as Kumar et al. (2021). We experiment with popular offline RL methods (see Table 2 and we provide algorithmic and hyperparameter details in Table A.2). **Offline Policy Evaluation.** We use WIS estimators for the tabular and discrete action domains: Sepsis and TutorBot. We use FQE for continuous action domains: Robomimic and D4RL. For each task, with a given dataset, we use the splitting procedure described in Section 5 to generate the partitioning. We describe how we compute the results for each figure in Appendix A.12.

7 Results

$\mathcal{A}_i$ **Selection Comparison.** In Figure 3(a), we compare five approaches (One-Split OPE, BCa, BVFT-FQE, CV, and SSR-RRS) on the Sepsis-POMDP domain with 1000 patients. For each approach, we compute a score per AH pair, and select the best algorithm according to each. For fairness in comparison, all selected $\mathcal{A}_i$ are re-trained on the full dataset and we report the final performance in the real environment. As expected, **One-Split OPE** performed the worst. Surprisingly, using the lower bound of bootstrapped (**BCa (LB)**) confidence interval also does not allow to pick good policies, LB being perhaps too conservative. We see that **CV 2-fold** and **CV 5-fold** do not perform well either. **CV 2-fold** does not allow enough repetition and **CV 5-fold** makes the validation set size too small. We observe clearly that SSR-RRS **5-split** performs the best and selected policies that are on average very close to the optimal policy's performance. **BVFT-FQE** relies on FQE, which is a misspecified model on the Sepsis domain and difficult to optimize given the small dataset size, hence it does not select good policies in Sepsis. However, in Robomimic Can-Paired and D4RL Hopper, BVFT-FQE is able to pick good policies, albeit not significantly better or worse than other methods, and still worse than SSR-RRS **5-split** in the mixed (more realistic) "medium-expert" dataset. We show more analysis of BVFT compared to our method in the Appendix. Table 3 aggregates the results for all the considered domains in our study. Our approach SSR-RRS **5-split** is distinctly able to more consistently select policies that, once deployed, perform close to the optimal policy across all tasks.

Re-trained on full dataset	BVFT-FQE π x FQE	BVFT-FQE π + FQE	CV-2	CV-5	SSR RRS-2	SSR RRS-5	Optimal Policy
Robomimic:							
Can-Paired	0.65	0.71	0.72	0.72	0.71	**0.73**	0.75
Transport	0.21	0.0	0.42	0.42	0.62	**0.70**	0.74
D4RL (Hopper):							
random	321.75	317.72	**325.37**	**325.37**	324.92	**325.37**	325.37
medium	934.71	1227.81	1227.81	1304.53	1296.87	**1304.54**	1392.93
medium-expert	2677.93	2677.93	2530.04	2530.04	3481.34	**3657.80**	3657.80
Sepsis:							
MDP (n=200)	—	-19.32	-10.26	-20.32	-13.01	**-7.85**	-1.94
POMDP (n=1000)	—	-1.92	0.74	-1.92	2.40	**6.75**	7.86
TutorBot:							
POMDP (n=200)	—	—	1.34	1.19	1.30	**1.38**	1.43

Table 3: Comparison of the performance obtained by a policy deployed using the SSR pipeline vs. using 1-split policy selection approaches on a wide range of application domains. Cells = average true return. We note that (π x FQE) is very computationally expensive when we search through a large AH space (in Sepsis and TutorBot), therefore we exclude them.

The Benefits of Re-training Policies Selected with SSR. In Figure 3(b), we plot the true reward of a selected policy $\mathcal{A}_i$ when only trained on 50% of the dataset (the training set) compared to when trained on 100% of the dataset. As expected, in the small data regime, every single trajectory matters. Policies trained on the full dataset significantly outperform policies trained only on half of it. This experiment provides strong evidence in favor of AH selection (done with RRS on the full dataset) over policy selection (done on the training set) in offline RL.

The Impact of Number of Repeats for SSR-RRS. The proposed pipeline SSR-RRS has a hyperparameter K for the number of repeated data splitting. In Figure 4(a), we show the true performance of the policy that is being selected by SSR-RRS with $K = 1, 2, 5$ by running 500 simulations with heatmaps on the frequency each policy is selected. We observe that when $K = 1$ (equivalent to the One-Split OPE method), policies are picked quite uniformly; many of which are performing poorly. When $K = 5$, higher-performing policies are selected much more frequently. From Table 3, we conclude that $K = 5$ generally works well across various domains. Naturally, the number of split K will be chosen in line with the computing budget available; $K = 5$ appears to be a reasonable choice.

The Impact of Dataset Size. Finally, we investigate to which extent the proposed pipeline is necessary when the dataset size is sufficiently large. We use the Sepsis-POMDP domain with 200, 1000 and 5000 patients. We show the best policies that are most frequently selected by our approach in Figure 4(b). Unsurprisingly, policies trained on larger datasets perform better. In the 200-patient dataset, having SSR-RRS **5-split** is crucial in picking the best policy, as most policies perform quite poorly. The gap between different approaches becomes smaller with 1000 patients, and even smaller when there are 5000 patients in the dataset. However, it is worth noting that even in the large dataset regime (N=5000), SSR-RRS still outperforms the One-Split OPE method in selecting the best algorithm.

Additional Analysis. Our method SSR-RRS can also be used to select hyperparameters for a single algorithm, as we demonstrate in Appendix A.8. One might also wonder how sensitive is SSR-RRS pipeline to the choice of OPE method used inside the pipeline. OPE methods are known to significantly vary in accuracy for different domains, and unsurprizingly, using a reasonable OPE method for the domain is important (see Appendix A.7). Note though the OPE estimators we use in our results are very popular ones, and it is possible to use standard approaches, though additional benefits may come from using even better OPE methods. Finally, related to this question, one might wonder if particular OPE methods might be biased towards certain OPL algorithms which make similar assumptions (such as assuming a Markov structure): interestingly in preliminary experiments, FQE estimators did not seem to give FQI algorithms higher performance estimations (see Appendix A.9).

8 Discussion and Conclusion

We presented SSR, a pipeline for training, comparing, selecting and deploying offline RL policies in a small data regime. The approach performs automated AH selection with a robust hyperparameter

evaluation process using repeated random sub-sampling. SSR allows to consistently and reliably deploy best-performing policies thanks to jointly avoiding overfitting on a single dataset split and being data efficient in re-using the whole dataset for final training. We prove that a single split has a high failure rate of discovering the optimal AH because of reward sparsity. We have demonstrated its strong empirical performance across multiple and various challenging domains, including real-world applications where AH tuning cannot be performed online.

There exist many interesting areas for future work. The proposed offline RL pipeline assumes the user/practitioner has selected a particular OPE method. OPE is an important subarea of its own and different approaches have different bias/variance tradeoffs. Recent work on automated model selection algorithms for OPE (Su et al., 2020; Lee et al., 2021) are a promising approach for producing good internal estimators. A second issue is that while our approach aims to produce a high-performing policy, it does not also produce an accurate estimate of this policy since the entire dataset is used at the end for training. An interesting issue is whether cross-splitting (Chernozhukov et al., 2016) or other methods could be used to compute reliable estimators as well as perform policy optimization.

9 Acknowledgment

Research reported in this paper was supported in part by a Hoffman-Yee grant, NSF grant #2112926 and the DEVCOM Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S.Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. We would like to thank Jonathan N. Lee, Henry Zhu, Matthew Jorke, and Tong Mu for discussions.

References

Bassen, J., Balaji, B., Schaarschmidt, M., Thille, C., Painter, J., Zimmaro, D., Games, A., Fast, E., and Mitchell, J. C. (2020). Reinforcement learning for the adaptive scheduling of educational activities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12.

Burman, P. (1989). A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514.

Cheng, C.-A., Kolobov, A., and Swaminathan, A. (2021). Heuristic-guided reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. (2016). Double/debiased machine learning for treatment and causal parameters. *arXiv preprint arXiv:1608.00060*.

Chernozhukov, V., Demirer, M., Duflo, E., and Fernandez-Val, I. (2018). Generic machine learning inference on heterogeneous treatment effects in randomized experiments, with an application to immunization in india. Technical report, National Bureau of Economic Research.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923.

Dubitzky, W., Granzow, M., and Berrar, D. P. (2007). *Fundamentals of data mining in genomics and proteomics*. Springer Science & Business Media.

Efron, B. (1983). Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American statistical association*, 78(382):316–331.

Efron, B. (1986). How biased is the apparent error rate of a prediction rule? *Journal of the American statistical Association*, 81(394):461–470.

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

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

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

Futoma, J., Hughes, M. C., and Doshi-Velez, F. (2020). Popcorn: Partially observed prediction constrained reinforcement learning. *arXiv preprint arXiv:2001.04032*.

Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American statistical Association*, 70(350):320–328.

Gilpin, A. R. (1993). Table for conversion of kendall's tau to spearman's rho within the context of measures of magnitude of effect for meta-analysis. *Educational and psychological measurement*, 53(1):87–92.

Guo, Y., Feng, S., Le Roux, N., Chi, E., Lee, H., and Chen, M. (2020). Batch reinforcement learning through continuation method. In *International Conference on Learning Representations*.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer.

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

Komorowski, M., Celi, L. A., Badawi, O., Gordon, A. C., and Faisal, A. A. (2018). The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11):1716–1720.

Krzanowski, W. and Hand, D. (1997). Assessing error rate estimators: the leave-one-out method reconsidered. *Australian Journal of Statistics*, 39(1):35–46.

Kumar, A., Singh, A., Tian, S., Finn, C., and Levine, S. (2021). A workflow for offline model-free robotic reinforcement learning. In *5th Annual Conference on Robot Learning*.

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

Le, H., Voloshin, C., and Yue, Y. (2019). Batch policy learning under constraints. In *International Conference on Machine Learning*, pages 3703–3712. PMLR.

Lee, J. N., Tucker, G., Nachum, O., and Dai, B. (2021). Model selection in batch policy optimization. *arXiv preprint arXiv:2112.12320*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Liao, P., Greenewald, K., Klasnja, P., and Murphy, S. (2020). Personalized heartsteps: A reinforcement learning algorithm for optimizing physical activity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(1):1–22.

Liu, Y., Swaminathan, A., Agarwal, A., and Brunskill, E. (2019). Off-policy policy gradient with stationary distribution correction. In Globerson, A. and Silva, R., editors, *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, volume 115 of *Proceedings of Machine Learning Research*, pages 1180–1190. AUAI Press.

Liu, Y., Swaminathan, A., Agarwal, A., and Brunskill, E. (2020). Provably good batch reinforcement learning without great exploration. *Advances in neural information processing systems*, 33.

Mandlekar, A., Ramos, F., Boots, B., Savarese, S., Fei-Fei, L., Garg, A., and Fox, D. (2020). Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE.

Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., and Martín-Martín, R. (2021). What matters in learning from offline human demonstrations for robot manipulation. In *5th Annual Conference on Robot Learning*.

Mandlekar, A., Zhu, Y., Garg, A., Booher, J., Spero, M., Tung, A., Gao, J., Emmons, J., Gupta, A., Orbay, E., et al. (2018). Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR.

Mazoure, B., Mineiro, P., Srinath, P., Sedeh, R. S., Precup, D., and Swaminathan, A. (2021). Improving long-term metrics in recommendation systems using short-horizon reinforcement learning. *arXiv preprint arXiv:2106.00589*.

Metelli, A. M., Papini, M., Faccio, F., and Restelli, M. (2018). Policy optimization via importance sampling. *Advances in Neural Information Processing Systems*, 31.

Miyaguchi, K. (2022). A theoretical framework of almost hyperparameter-free hyperparameter selection methods for offline policy evaluation. *arXiv preprint arXiv:2201.02300*.

Oberst, M. and Sontag, D. (2019). Counterfactual off-policy evaluation with gumbel-max structural causal models. In *International Conference on Machine Learning*, pages 4881–4890. PMLR.

Owen, A. B. (2013). Monte carlo theory, methods and examples.

Paine, T. L., Paduraru, C., Michi, A., Gulcehre, C., Zolna, K., Novikov, A., Wang, Z., and de Freitas, N. (2020). Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97.

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

Shi, C., Wan, R., Song, R., Lu, W., and Leng, L. (2020). Does the markov decision process fit the data: Testing for the markov property in sequential decision making. In *International Conference on Machine Learning*, pages 8807–8817. PMLR.

Siegel, N., Springenberg, J. T., Berkenkamp, F., Abdolmaleki, A., Neunert, M., Lampe, T., Hafner, R., Heess, N., and Riedmiller, M. (2020). Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133.

Su, Y., Srinath, P., and Krishnamurthy, A. (2020). Adaptive estimator selection for off-policy evaluation. In *International Conference on Machine Learning*, pages 9196–9205. PMLR.

Tang, S. and Wiens, J. (2021). Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Machine Learning for Healthcare Conference*, pages 2–35. PMLR.

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

Thomas, P., Theocharous, G., and Ghavamzadeh, M. (2015a). High confidence policy improvement. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2380–2388, Lille, France. PMLR.

Thomas, P., Theocharous, G., and Ghavamzadeh, M. (2015b). High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388. PMLR.

Thomas, P. S., Castro da Silva, B., Barto, A. G., Giguere, S., Brun, Y., and Brunskill, E. (2019). Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004.

Voloshin, C., Le, H. M., Jiang, N., and Yue, Y. (2021). Empirical study of off-policy policy evaluation for reinforcement learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Xie, T. and Jiang, N. (2021). Batch value-function approximation with only realizability. In *International Conference on Machine Learning*, pages 11404–11413. PMLR.

Yin, M., Bai, Y., and Wang, Y.-X. (2021). Near-optimal offline reinforcement learning via double variance reduction. *Advances in neural information processing systems*, 34.

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

Zanette, A. and Brunskill, E. (2019). Tighter problem-dependent regret bounds in reinforcement learning without domain knowledge using value function bounds. In *International Conference on Machine Learning*, pages 7304–7312. PMLR.

Zhang, S. and Jiang, N. (2021). Towards hyperparameter-free policy selection for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 7 and Section 8.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4.1 and Section A.3.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 4.1 and Section A.3.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Supplementary Material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Sections 6, A.13, A.15, A.16, A.17 and A.18
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] Open-source
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix

A.1 Prelude Experiment

In this section, we put ourselves in a situation where model selection would be performed by comparing different AH pairs on their internal objective or value function estimates on a given dataset, as described near the beginning of Section 1. We use three datasets of different qualities (random, medium, and medium-expert) of the popular Hopper task from the D4RL benchmark (see Appendix A.13 for a detailed description) to train a total of 36 policies with different AH pairs and then calculate the resulting TD-Errors and Q-values on the whole dataset at the end of training.

To evaluate the performance one would obtain by employing such an approach to select the best policy, we report in Table A.1 the performance (true return in the environment) of the selected policies and compare them with the performance of the optimal policy for each of the datasets. The policies are selected either by finding the one which corresponds to the lowest TD-Error, or the one which corresponds to the highest Q-value. We also include the Kendall rank correlation coefficient (Gilpin, 1993) for each of the ranking methods (ranking with respect to TD-Error or Q-value) compared with the "true ranking" of policies ranked with respect to the performance in the environment:

$$\tau = \frac{(\text{ number of concordant pairs }) - (\text{ number of discordant pairs })}{\binom{n}{2}}$$

where n is the number of policies, and where "concordant pairs" are pairs from the two compared rankings for which the sort order agrees. A coefficient of 1 means the agreement between the two rankings is perfect.

	TD-Error		Q-value		
	Policy Selected (True Return)	Kendall	**Policy Selected (True Return)**	Kendall	**Optimal Policy (True Return)**
random	334.24	-0.09	333.65	-0.15	345.39
medium	1475.82	0.42	2381.37	0.21	2469.81
medium-expert	327.97	-0.18	327.97	-0.09	3657.80

Table A.1: Average Return (True Return obtained in the simulator) of the policy selected with respect to min(TD-Error) or max(Q-value) on the training dataset with a comparison to the True Return obtained by the Optimal Policy. Kendall rank correlation coefficient when ranking with respect to the same metrics. Policies are **trained and validated on the same dataset**. Task: Hopper.

Unsurprisingly, Table A.1 shows that one cannot rely on this straightforward pipeline to select a best-performing AH pair. Actually, for most of the datasets (the medium-expert dataset should resemble the most to what a dataset would look like in a real-world situation as it is composed of both high-quality and medium-quality data), following such an approach would produce and deploy a very bad performing policy.

A.2 Connection between Leave-p-Out CV and RRS

Our RSS is a finite approximation of Leave-p-out (Lp0) cross-validation[1]. LpO is known in supervised learning, but rarely used due to the computational burden. The correctness of LpO is proved itecelisse2014optimal in a supervised learning setting with projection estimators. Unlike K-fold cross-validation, Leave-p-out CV selects p data points for evaluation and the rest for training. In our proposed RSS method, we set $p = n/2$, and instead of exhaustively enumerating all possible selections of p data points out of n data points, we only repeat this process K times. Asymptotically as the amount of data goes to infinity, this approach should be correct, but also a single train/test split will also be correct in such a setting. The key challenges arise in the finite data setting, where the choice of dataset partitioning is key.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeavePOut.html

A.3 Proof of Theorem 1

Consider a finite set of J offline RL algorithms $\mathcal{A}$. Let the policy produced by algorithm $\mathcal{A}_j$ on training dataset $\mathcal{D}$ be π_j, its estimated performance on a validation set $\hat{V}^{\pi_j}$, and its true (unknown) value be V^{π_j}. Denote the true best resulting policy as $\pi_{j^*} = \arg\max_j V^{\pi_j}$ and the corresponding algorithm $\mathcal{A}_{j^*}$. Let the best policy picked based on its validation set performance as $\pi_{\hat{j}^*} = \arg\max_j \hat{V}^{\pi_j}$ and the corresponding algorithm $\mathcal{A}_{\hat{j}^*}$.

Theorem 1. *Then there exist stochastic decision processes and datasets such that (i) using a single train/validation split procedure will select a suboptimal policy and algorithm with significant finite probability, $P(\pi_{\hat{j}^*} \neq \pi_{j^*}) \geq C$, with corresponding substantial loss in performance $O(V_{max})$, and, in contrast, (ii) averaging across N_s train/validation splits will select the optimal policy with probability 1: $\lim_{N_s \to \infty} P(\pi_{\hat{j}^*} = \pi_{j^*}) \to 1$.*

Proof. We proceed by constructing a stochastic decision process. A common domain to illustrate the importance of strategic exploration is a chain MDP. Here consider an episodic, finite horizon, finite chain, deterministic decision process with 6 states, $s_1, \ldots, s_H$, $(H = 6)$ with two actions. a_1 moves the state one down except for at the starting state, and a_2 increments the state one up except for the final state: more formally, $p(s_{i-1}|s_i, a_1) = 1$ except for $p(s_1|s_1, a_1) = 1$; $p(s_{i+1}|s_i, a_2)$ except for $p(s_H|s_H, a_2) = 1$. The reward is 0 in all states except $R(s_1) = 1/6$ and $R(s_H) = 201$. All episodes are length $H = 6$ and start in state s_1. The optimal policy always takes a_2 and achieves $V_{max} = R(s_H)$. Any other policy achieves at most $H * 1/6 = 1$ reward.

The behavior policy is uniform random over the two actions, $\pi_b(a_1|s) = 0.5 = \pi_b(a_2)$. Let the available offline dataset D consist of 200 episodes gathered using π_b. Given the behavior policy, each of the 64$= 2^H$ unique trajectories has an equal probability of being observed, and only one of these $\tau_h = (s_1, 0, a_2, s_2, 0, a_2, , s_3, 0, a_2, s_4, 0, a_2, s_5, a_2, s_H, R(s_H))$ achieves the highest return. On average out of 200 episodes[2], n_{τ_h}=3(=$round(|\mathcal{D}|/(2^H))$) episodes will match τ_h. All other episodes will have a return of 1 or less.

Let there be a set of H offline RL algorithms $\mathcal{A}_h$, each which optimizes the reward over a different horizon $h = 1 : H$, by constructing a maximum-likelihood estimate (MLE) MDP model $\mathcal{M}$ given a training dataset D_{tr}, and then computing a policy π_h that optimizes the h-step value given the learned MDP $\mathcal{M}$ model[3] For example, algorithm $\mathcal{A}_2$ will take the MLE MDP model and construct a policy to optimize the sum over rewards for the next two time steps $\pi_2(s) = \arg\max_a r(s) + \sum_{s'} p(s'|s, a) r(s')$. We think this is a reasonable set of algorithms to consider as an illustrative example: the horizon length can directly influence the amount of data needed to compute an optimal policy, and recent work has explored using shorter horizons (Cheng et al., 2021; Mazoure et al., 2021; Liao et al., 2020), so choosing the right horizon can be viewed as a bias/variance tradeoff, suitable for automatic model selection.

Observe that even if given access to the true (unknown) MDP parameters, algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_{H-1}$ will compute a policy that is suboptimal: due to the shortened horizon length, to optimize the expected total reward, the resulting policy computed for s_1 will be $\pi_h(s_1) = \pi_1(s_1) = a_1$ for these algorithms $\mathcal{A}_h$, $h = 1 : H - 1$. As the MDP is deterministic, this will also be true for any input dataset.

We now consider the impacts of partitioning the input dataset into a training dataset D_{tr} taken as input by each algorithm $\mathcal{A}_h$ to compute a policy $\pi_{\hat{h}}$, and an evaluation/test dataset D_{te}: $D = D_{tr} \cup D_{te}$. For algorithm $\mathcal{A}_H$ to learn the optimal policy π^* which achieves V_{max}, it must learn over a dataset D_{tr} that includes one or more examples of the highest return trajectory τ_h. Note that a single episode of τ_h in the training set is sufficient to learn the optimal policy[4].

[2]Our calculations can easily be extended to cases where there are different numbers of observed τ_h, but for simplicity we assume a dataset where the average expected number of τ_h are observed.

[3]During planning with the learned MDP model, we restrict taking the maximum value over actions for a given state s to only actions that have been taken at least once in that state in the dataset, e.g. $\max_{a\, s.t.\, n(s,a) \geq 1}$, where $n(s, a)$ is the counts of the number of times action a was taken in state s in the dataset. Note that in a finite dataset, some states and/or actions may not be observed, and this common choice simply ensures that the algorithm does not overestimate the value of untried actions.

[4]A single example of τ_h will induce a MLE $\hat{\mathcal{M}}$ with the correct reward model for all states, and the dynamics model for action a_2. From the procedure used to compute an optimal policy $\hat{\mathcal{M}}$, this will result in an optimal policy.

Assume that the offline evaluation of the policies learned by the algorithm on D_{tr} is performed using importance sampling on D_{te}: note, our results will still apply, with minor modifications, if off policy evaluation is performed on D_{te} using fitted Q evaluation (Le et al., 2019) or using a certainty-equivalent MDP constructed from D_{te}.

Then the off policy evaluation of the policy learned by the full horizon algorithm $\mathcal{A}_H$, $\hat{V}^{\pi_H}(s_1)$, will only be greater than 1 if there also exists at least one episode of the highest return trajectory τ_h.

Assume the training dataset and validation dataset are constructed by randomly sampling 50% of the episodes to be in each. By assumption, there are n_{τ_h} samples of τ_h, which have an equal chance of being in either the training or validation set. There are $n_{\tau_h} + 1$ ways to partition the n_{τ_h} exchangable episodes of τ_h into the training and validation sets, here $([3,0],[2,1],[1,2],[0,3])$. Note the training and validation set are identical in size ($|\mathcal{D}|/2$ trajectories each), and we only care about whether a trajectory τ is identical to τ_h or not. The probability that each of these partitions occurs is : $P([3,0]) = P([0,3]) = \frac{100}{200} * \frac{99}{199} * \frac{98}{198} \approx 0.123$.

From the above analysis, $\mathcal{A}_H$ can only learn an optimal policy, and its estimated value $\hat{V}^{\pi_5} > 1$ on D_{te} if there is at least one τ_h in both the training and validation set datasets, which occurs in partitions $([2,1],[1,2])$. This occurs with probability 0.754. Otherwise, either (a) $\mathcal{A}_H$ will not learn an optimal policy, and instead will learn $\pi_H(s_1) = \pi_1(s_1) = a_1$, or (b) $\mathcal{A}_H$ will learn an optimal policy $\pi_H(s_1) = a_2$ but as the validation dataset does not contain τ_h, $\hat{V}^{\pi_H} = 1/H < \hat{V}^{\pi_1}$. In both cases, the selected policy given its performance on the validation set will be $\pi_1(s_1) = a_1$. The resulting loss in performance is $V_{max} - V^{\pi_1} = V_{\max} - 1 = O(V_{\max})$. This failure occurs with substantial probability 24.6%. This proves part (i) of the theorem.

To prove part (ii) of the proof, we consider cases where at least one τ_h is in both D_{tr} and D_{te}. Note $\hat{V}^{\pi_1} \leq \frac{R(s_1)}{1/2^H} = \frac{1}{1/2^H}$. Define E_{ss} as a "successful split": the event that 1 or more of τ_h (high returns) episodes are in D_{te}, but not all n_{τ_h}. On event E_{ss}, the optimal policy (which will be computed by $\mathcal{A}_H$ on the training set), will have an estimated value on D_{te}, using importance sampling:

$$\hat{V}^{\pi^*}_{ss} \geq \frac{1}{|D_{te}|}\frac{R(s_H)}{1/2^H} = \frac{1}{1/2^H} * \frac{201}{100} > 2\hat{V}^{\pi_1} \tag{2}$$

since there are at least 1 τ_h trajectories, each with propensity weight $\frac{1}{1/2^H}$ and reward $R(s_H)$. Therefore on Event E_{ss} the optimal policy can be learned and estimated as having high reward. The probability of event E_{ss} is greater than 0.5: $P(E_{ss}) = 0.754$.

In the repeated train-validation split setting, the algorithm selected is the one that has the best performance on the validation set, on average across all N_s splits. Let E_h be the event that at least half the train-validation dataset splits are successful (Event E_{ss} holds for that split). In this case then the average performance of $\mathcal{A}_5$ will be at least

$$\begin{aligned}
\hat{V}_{\mathcal{A}_5} &\geq \frac{1}{N_s}\left(\frac{N_s}{2}\hat{V}^{\pi^*}_{ss} + 0\right) \\
&\geq \frac{1}{N_s}\left(\frac{N_s}{2}2\hat{V}^{\pi_1} + 0\right) \\
&= \hat{V}^{\pi_1},
\end{aligned}$$

where the first line uses a lower bound of 0 when the event E_{ss} fails to hold, and substitutes in Equation 2. Therefore as long as event E_h holds, the optimal policy π^* (which will be computed by algorithm $\mathcal{A}_H$ will be selected. Since $P(E_{ss}) > 0.5$, the probability[5] as the number of splits goes to infinity that E_{ss} holds on least half of those splits goes to 1: $\lim_{N_s \to \infty} P(E_h) \to 1$.

□

[5] **calculate for finite S.

A.4 SSR pseudo-code

Algorithm 1 SSR-RRS: $\mathcal{A}_i$ Selection with Repeated Random Sub-sampling

Input: offline RL data $\mathcal{D}$; set of AH pairs $[\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_z]$, OPE estimator $\widehat{V}$, split number $K \in \mathbb{N}$.

Output: policy $\hat{\pi}^*$ for deployment

$\mathcal{R} = \emptyset$
for $i \leftarrow 1...K$ **do**
  $R_i^{\text{train}}, R_i^{\text{valid}} = \texttt{Subsample}\ (\mathcal{D}, 0.5)$
  $\mathcal{R} = \mathcal{R} \cup (R_i^{\text{train}}, R_i^{\text{valid}})$
end
$\mathcal{G} = []$
for $i \leftarrow 1...z$ **do**
  $\mathcal{S} = []$
  for $j \leftarrow 1...K$ **do**
    $\pi_i = \mathcal{A}(R_j^{\text{train}})$
    $\mathcal{S}_{ij} = \widehat{V}(\pi_i; R_j^{\text{valid}})$
  end
  $\mathcal{G}_i = \frac{1}{K}\sum_{j=1}^{K} \mathcal{S}_{ij}$
end
$\mathcal{A}^* = \mathcal{A}_{o+}$ where $o = \arg\max(\mathcal{G})$
$\pi^* = \mathcal{A}^*(\mathcal{D})$
return π^*

A.5 Experiment Detail Summary

We choose different sets of algorithms to evaluate our pipeline in every domain to demonstrate the generality of our approach and because some algorithms have limitations inherent to certain types of domains to which they can be applied. We list them in Table A.2.

Running a large number of algorithm-hyperparameter pairs many times is very computationally expensive. In order to save time and resources, we leverage the fact that multiple approaches can share resources. We describe how we compute the numbers for each approach as follows:

For each offline RL dataset in Sepsis, TutorBot, Robomimic, and D4RL, we produce the following partitions (we refer to this as the "partition generation procedure"):

1. 2-fold CV split (2 partitions consisted of (S_i))
2. 5-fold CV split (5 partitions consisted of (S_i))
3. 5 RRS split (5 partitions consisted of $(R_i^{\text{train}}, R_i^{\text{valid}})$)

Here, we briefly describe how to use these data partitions to select algorithms with alternative approaches.

One-Split OPE. The One-Split OPE method can be conducted to train and evaluate an algorithm on any of the RRS splits being produced, but only look at one split, without considering other splits. We let for a particular i, we let $\mathcal{D}_{\text{train}} = R^{\text{train}_i}$ and $\mathcal{D}_{\text{valid}} = R_i^{\text{valid}}$.

BCa Bootstrap. Similar to the One-Split OPE method, we can use RRS split for bootstrap. For a particular i, we let $\mathcal{D}_{\text{train}} = R^{\text{train}_i}$ and $\mathcal{D}_{\text{valid}} = R_i^{\text{valid}}$. Bootstrapping will re-sample with replacement on trajectories in $\mathcal{D}_{\text{valid}}$ to create (largely) overlapping subsets $B_1, B_2, ..., B_N$, with $|B_i| = n$. We then evaluate π_e on each subset using $\widehat{V}$. The final score is computed through a bias correction process with an added acceleration factor (BCa).

Experiment Domain	Number of Trajectories (N)	Average Trajectory Length	Number of Transitions in Total	AH Pairs Evaluated	Algorithms in Experiment
Sepsis-POMDP	200	14	2792	540	BC, POIS, BC+POIS, BC+mini-POIS, BCQ, MBSQI, pMDP, MOPO
Sepsis-POMDP	1000	14	13708	540	BC, POIS, BC+POIS, BC+mini-POIS, BCQ, MBSQI, pMDP, MOPO
Sepsis-POMDP	5000	14	68576	148	BC, POIS, BCQ, MBS-QI, pMDP, MOPO
Sepsis-MDP	200	14	2792	383	BC, BCQ, MBSQI, pMDP, POIS, BC + POIS BC+mini-POIS
TutorBot	200	5	987	81	BC, POIS, BC+POIS, BC+mini-POIS
Robomimic Can-Paired	200	235	47,000	35	BC, BCRNN, CQL, IRIS, BCQ
Robomimic Transport	200	470	94,000	10	BC, BCRNN, CQL, IRIS, BCQ
D4RL Hopper	500	1000	500,000	4 x 4	BCQ
D4RL HalfCheetah	500	1000	500,000	4 x 4	BCQ

Table A.2: List of algorithms being used in which domain. 4 x 4 means we evaluate 4 AH pairs for the policy learning and 4 AH pairs for the policy evaluation estimators (FQE).

Nested K $\times$ 2-fold Cross-Validation. We can also use the RRS split partitions to produce $K \times 2$ Nested CV by taking one RRS split $(R_i^{\text{train}}, R_i^{\text{valid}})$ by doing the following procedure:

$$s_i = \frac{\widehat{V}(\mathcal{A}(R_i^{\text{train}}); R_i^{\text{valid}}) + \widehat{V}(\mathcal{A}(R_i^{\text{valid}}); R_i^{\text{train}})}{2} \tag{3}$$

$$\mathcal{G}_{\mathcal{A},\text{NCV}_K} = \frac{1}{K}\sum_{i=1}^{K} s_i \tag{4}$$

Intuitively, for $K \times 2$ Nested CV, we just need to swap the train and valid set produced by repeated sub-sampling and average to produce the algorithm performance score for a particular split i. Then we average the scores to get a final score for the algorithm.

2-fold Cross-Validation. Similar to the $K \times 2$ Nested CV, we can choose the i-th partition generated by the 10 RRS split procedure, and compute the score according to Equation 3. We do this for the Sepsis and TutorBot domains, but we do not do this for the Robomimic domain.

Batch Value Function Tournament (BVFT) Xie and Jiang (2021); Zhang and Jiang (2021) proposed to use pairwise Q-function comparisons to select the optimal Q-function from a set of Q-functions. Given Q_i, Q_j, let $\mathcal{G}_{ij}$ be the piecewise constant function class induced by binning (s, a) and (s', a') if $Q_i(s, a) = Q_j(s', a')$. Given an offline dataset D, we can compute the BVFT loss as follow:

$$\hat{\mathcal{T}}_{\mathcal{G}_{ij}} Q := \arg\min_{g \in \mathcal{G}_{ij}} \frac{1}{|D|} \sum [(g(s,a) - r - \gamma \max_{a'} Q(s', a'))^2] \tag{5}$$

$$\mathcal{E}_{\epsilon_k}(Q_i, Q_j) = \|Q_i - \hat{\mathcal{T}}_{\mathcal{G}_{ij}} Q_j\|_{2,D} \tag{6}$$

$$\mathcal{E}_{\epsilon_k}(Q_i) = \max_j \mathcal{E}_{\epsilon_k}(Q_i, Q_j) \tag{7}$$

Zhang and Jiang (2021) proposed a method to automatically search through different discretization resolutions (ϵ_k). In our experiment, we search through $[0.1, 0.2, 0.5, 0.7, 1.0, 3.0, 10.0]$. We use the BVFT code provided by Xie and Jiang (2021). Because BVFT can only compare Q-functions, Zhang and Jiang (2021) offered two strategies to perform policy selection for any model/algorithm. Here we briefly describe two strategies:

- Strategy 1 (π x FQE): if we have 4 policies, and each policy is evaluated by 4 FQEs, then this strategy will compare 16 Q-functions (4 π x 4 FQE).
- Strategy 2 (π + FQE): if we have 4 policies, and each policy is evaluated by 4 FQEs, then this strategy will first run BVFT to compare 4 Q-functions (1 π x 4 FQE), select the best Q-function for each π (4 π x 1 FQE), then we select the best policy by the average Q-value computed by each FQE.

We generally find strategy 2 more computationally efficient (because it makes a smaller number of comparisons). BVFT generally has $O(J^2)$ time complexity where J is the number of Q-functions that need to be compared – it's easy to see that $16^2 = 256$ is much larger than $4^2 = 16$.

Our repeated experiment protocol (RRS) is reliant on choosing a good FQE. In order to compare fairly, for π x FQE strategy, we only use the optimal FQE (the ones used in RRS and CV and one-split). We can see that in this condition, BVFT can do pretty well (even outperforming RRS in the D4RL-Hopper medium setting). For π + FQE, because it focuses on the selection of FQE, we try 4 different FQE hyperparameters. We discuss this more in D4RL Experiment Details (in Section A.18).

A.6 Computational Complexity

Most of the approaches we discussed in Section A.5 leverage multiple repetitions (resampling) to account for data allocation randomness. We provide a time complexity table below and define the following terms:

- H = number of AH pairs to evaluate
- N = total data samples. We assume the training time for each trajectory is N_1 and evaluation time for each trajectory is N_2, where $N = N_1 + N_2$
- M = number of folds in multi-fold cross-validation
- B = number of bootstraps (this number is 100 in our experiment)
- P = number of resolutions for BVFT's grid (proposed in Zhang and Jiang (2021))
- F = number of FQE hyperparameters (proposed in Zhang and Jiang (2021))

	Training Complexity	Evaluation Complexity
One-Split	H $\times$ N_1	H $\times$ N_2
Bootstrapping (BCa)	H $\times$ N_1	H $\times$ B $\times$ N_2
M-Fold Cross-Validation	(H $\times$ M $\times$ N $\times$ (M-1))/M = H $\times$ N $\times$ (M-1)	(H $\times$ M $\times$ N $\times$ 1)/M = H $\times$ N
K-Repeat RRS	H $\times$ K $\times$ N_1	H $\times$ K $\times$ N_2
BVFT (Xie and Jiang, 2021)	H $\times$ N_1	(H $\times$ H) $\times$ N_2 or (H $\times$ H) $\times$ n_2
BVFT-auto (Zhang and Jiang, 2021)	H $\times$ N_1	P $\times$ (H $\times$ H) $\times$ N_2 or P $\times$ (H $\times$ H) $\times$ n_2
BVFT-FQE (Zhang and Jiang, 2021)	H $\times$ N_1	P $\times$ H $\times$ (F $\times$ F) $\times$ N_2 or P $\times$ H $\times$ (F $\times$ F) $\times$ n_2

For BVFT, one can amortize the computational cost by caching (storing $Q(s, a)$ for all (s, a) in the dataset). If caching is done only once, we treat the actual computation time for the validation data set as n_2. P is usually between 5 and 10. When H is relatively large, for example, H = 540 (in our experiment), H * H = 2.916e5. It's easy to see that RRS is slightly more expensive than M-Fold

CV but less expensive than the pairwise comparison tournament algorithm (BVFT). Zhang and Jiang (2021) proposed BVFT-FQE that only makes pairwise tournament comparison between FQE hyperparameters – F is 5 in our experiments. It's also worth noting that BCa has a high evaluation cost when B is large – when B = 100, BCa evaluation cost is significantly higher than CV and RRS.

A.7 Sensitivity to OPE Methods

OPE is often a critical part of OPL, which has motivated significant research into OPE. Thus the employed OPE method will likely impact the performance of our proposed pipeline. As has been demonstrated in a recent bake-off paper (Voloshin et al., 2021), minimal-assumption OPE methods like weighted doubly robust methods (e.g. Jiang et al. (2015); Thomas and Brunskill (2016)) may be most consistently accurate for many domains. However if the domain is known to be Markov and the models are well specified, FQE methods will likely be more accurate in small data regimes.

To explore further the impact of the choice of OPE method, we conducted an additional experiment on the Sepsis-POMDP domain. The aim to was to look at the sensitivity of SSR-RRS for picking the best AH to the choice of OPE estimators. In addition to the prior OPE methods used in the main text, we included clipped IS (importance sampling), CWPDIS (Thomas and Brunskill, 2016), and 8 different FQE OPE variants, in which different networks, learning rate and epochs were used.

Sepsis-POMDP	Parameters	Best AH Performance Chosen by SSR-RRS K=5
FQE-1	[64], lr=3e-4, epoch=20	2.84
FQE-2	[64], lr=1e-5, epoch=20	-74.26
FQE-3	[64], lr=3e-4, epoch=50	-20.88
FQE-4	[64], lr=1e-5, epoch=50	-14.16
FQE-5	[128], lr=3e-4, epoch=20	-75.26
FQE-6	[128], lr=1e-5, epoch=20	-14.48
FQE-7	[128], lr=3e-4, epoch=50	-75.54
FQE-8	[128], lr=1e-5, epoch=50	-74.26
IS	N/A	4.47
CWPDIS	N/A	4.68
WIS	N/A	6.75

Table A.3: Using different OPE estimators in the SSR-RRS pipeline. FQE-1 denotes the FQE with the optimal FQE hyperparameter (heuristically chosen).

First, using FQE does generally much worse in this setting which is not very surprizing: FQE assumes the domain is Markov, which Sepsis-POMDP is not.

All importance-sampling based OPE methods yield quite similar performing algorithm-hyperparameter choices in this setting.

While there are some clear differences, if some basic information about the domain is known (Markov or not), it is likely possible to select a pretty good OPE. In addition, prior work has proposed heuristics (Voloshin et al., 2021) or automatic methods for automatic OPE selection (Su et al., 2020; Lee et al., 2021). An interesting direction for future work would be to include such methods in the pipeline.

We highlight that while it is well known that OPE methods are important, our paper focused on an under-explored issue: that the dataset partitioning can also introduce a substantial amount of *additional* impact on learning good policies / selecting good AH.

A.8 Robustness of SSR-RRS

In Table 3, we only show the performance of the best policy among all AH pairs. Here we show that SSR-RRS can still robustly select a good hyperparameter for a given offline RL policy learning algorithm (the gap between best AH selected and true best AH is relatively small).

Sepsis-POMDP	Range of True Policy Performance (95%CI)	Percentile of AH Chosen by SSR-RRS	Performance of AH Chosen by SSR-RRS	True Best AH Performance
BCQ	[-10.8, -0.73]	94%	5.98	7.86
MBSQI	[-7.34, -2.26]	95%	6.40	7.42
BC	[-8.98, -8.37]	58%	-8.46	-7.42
BC+PG	[-5.55, -4.26]	78%	-3.68	2.52
P-MDP	[-31.17, -21.26]	83%	0.23	2.82

Table A.4: We show the relative position (percentile) of the AH selected by SSR-RRS K=5 pipeline.

For each algorithm, we evaluate over 24 to 72 hyperparameters, and we compute the 95% confidence interval of all these policies' true performance. Except for behavior cloning, we are picking hyperparameters that are out-performing 78%-95% of other hyperparameters in the same algorithm.

A.9 Is FQE biased towards FQI algorithms?

In our evaluation on the Sepsis domain, FQE is used to evaluate both BCQ and MBSQI (both FQI-based) and BC and BCPG (policy-gradient algorithms).

We designed the following analysis experiment using our logged results. We first rank all AH pairs (540 of them) with their true performance in the simulator, and then we count the percentage of FQI (BCQ, MBSQI) algorithms that appear in the top 10%, 20%, and 50% percentile. The number in each cell should be read as: "90.7% of AH pairs in the top-10% based on True Performance are FQI-based". If FQE is biased towards FQI algorithms, we expect to see a higher percentage of BCQ and MBSQI AH pairs selected than the true performance baseline and compared to other OPE methods.

Sepsis-POMDP OPE Method	% of BCQ and MBSQI AHs in Top-10% AHs	% of BCQ and MBSQI AHs in Top-20% AHs
True Performance	90.7%	61.1%
FQE-1	0%	0%
WIS	9.4%	35.5%
RRS-5 WIS	68.5%	58.3%

Table A.5: Examining whether FQE as an estimator will prefer FQI policy learning algorithms.

Based on this analysis, we believe that FQE is not biased to select FQI-based algorithms in the Sepsis-POMDP domain. However, our analysis is limited to one domain and only on two FQI-based algorithms. Further investigation is needed but beyond the scope of our paper.

A.10 Additional Discussions

Sensitivity to K in small and large datasets In general, we expect the issue of data partitioning into a train and test split is most important in small datasets: as the dataset gets very large, a single train/test split will generally work well. Therefore, we suggest using a larger K for smaller datasets, but for larger datasets, a smaller K will likely be sufficient. Using our theoretical example in the appendix (chain-MDP), this can also be observed – with a larger N, the failure probability for smaller numbers of repeats decreases. This N-K tradeoff has computational benefits if there is a limited computational budget (larger datasets will require more training, therefore, it is harder to use a larger K).

Weighted importance sampling (WIS) as a biased estimator WIS is a self-normalizing importance sampling estimator. We refer readers to Owen (2013) Chapter 9 for a more detailed discussion on the statistical properties of this type of estimator. In Section 4.1 (line 174), we state:

WIS will return the observed return of the behavior policy if averaging over a single trajectory, independent of the target policy to be evaluated.

In brief, WIS works by first computing the probability of the dataset trajectory appearing under the evaluation policy and behavior policy:

$$w_i = \prod_{t=1}^{L} \frac{\pi_e(a_t|s_t)}{\pi_b(a_t|s_t)}$$

Then, this coefficient is normalized before multiplying with the trajectory return, therefore:

$$\text{WIS}(D) = \frac{1}{n} \sum_{i=1}^{n} \frac{w_i}{\sum_{j=1}^{n} w_j} (\sum_{t=1}^{L} \gamma^t R_t^i).$$

Perhaps surprisingly, if there is a single trajectory, $n = 1$, this implies

$$\text{WIS}(D) = \frac{w_i}{w_i} (\sum_{t=1}^{L} \gamma^t R_t^i) = \sum_{t=1}^{L} \gamma^t R_t^i.$$

Here WIS is a biased estimator that returns the trajectory weighted reward, independent of w_i.

A.11 Additional Experiment

We report the D4RL HalfCheetah result over the same setting as D4RL Hopper, where the result is averaged over 20 runs.

Re-trained on full dataset	BVFT π x FQE	BVFT π + FQE	CV-2	CV-5	SSR RRS-2	SSR RRS-5	Optimal Policy
D4RL (HalfCheetah):							
random	-1.14	1106.94	-1.13	-1.13	**1922.07**	**1922.07**	1922.07
medium	**4421.95**	4290.33	4290.33	4290.33	4290.33	4290.33	4517.96
medium-expert	8118.84	8799.66	8118.84	8118.84	**9681.78**	**9681.78**	10364.36

Table A.6: Additional comparison of the performance obtained by a policy deployed using the SSR pipeline vs. using 1-split policy selection approaches on D4RL HalfCheetah. Cells = average true return.

A.12 Figure Generation Procedure

Given our partition generation procedure, there are some methods (One-Split OPE, $K \times 2$ Nested CV, and SSR-RRS K when $K < 5$) that have a few different partitions to choose from. For example, out of the 5 RRS split partitions, which partition should we choose for the One-Split OPE method? If we choose one partition, and the One-Split method cannot select the best algorithm, does that mean the One-Split method is bad, or could the 9 other partitions do better for the One-Split method? In order to evaluate these approaches fairly, we exhaustively train and evaluate on the 5 RRS splits, swap the train/valid set, and train/evaluate on them again, generating 20 scores. For the aforementioned methods, we randomly sample from these 10 (or 20, if Nested CV is being evaluated) scores to simulate the setting that we happen to get one particular split. We run this sampling procedure multiple times and compute the average performance of the policies that are chosen by conditioning on one or K particular partitions.

A.13 Domain Descriptions

Sepsis. The first domain is based on the simulator and works by Oberst and Sontag (2019) and revolves around treating sepsis patients. The goal of the policy for this simulator is to discharge patients from the hospital. There are three treatments the policy can choose from antibiotics, vasopressors, and mechanical ventilation. The policy can choose multiple treatments at the same time or no treatment at all, creating 8 different unique actions.

The simulator models patients as a combination of four vital signs: heart rate, blood pressure, oxygen concentration and glucose levels, all with discrete states (for example, for heart rate low, normal and high). There is a latent variable called diabetes that is present with a 20% probability which drives the likelihood of fluctuating glucose levels. When a patient has at least 3 of the vital signs simultaneously out of the normal range, the patient dies. If all vital signs are within normal ranges and the treatments are all stopped, the patient is discharged. The reward function is $+1$ if a patient is discharged, -1 if a patient dies, and 0 otherwise.

We follow the process described by Oberst and Sontag (2019) to marginalize an optimal policy's action over 2 states: glucose level and whether the patient has diabetes. This creates the **Sepsis-POMDP** environment. We sample 200, 1000, and 5000 patients (trajectories) from Sepsis-POMDP environment with the optimal policy that has 5% chance of taking a random action. We also sample 200 trajectories from the original MDP using the same policy; we call this the **Sepsis-MDP** environment.

Robomimic. Our approach is further evaluated on a third domain, Robomimic (Mandlekar et al., 2021), consisting of various continuous control robotics environments along with corresponding sets of suboptimal human data. More specifically, we use the **Can-Paired** dataset composed of mixed-quality human data. These 200 demonstrations include an equal combination of "good" (the can is picked up and placed in the correct bin) and "bad" trajectories (the can is picked up and thrown out of the robot workspace). The initializations of the tasks being identical, it is expected that algorithms dealing with suboptimal data will be able to filter out the good trajectories from the bad ones and achieve near-optimal performance. Interestingly, state-of-the-art batch RL algorithms do not reach maximum performance (Mandlekar et al., 2021), making this task a good testbed for our procedure. We also use the **Transport** dataset, where two robot arms must transfer an object from one bin to another. The dataset contains 200 successful trajectories collected by one human operator.

D4RL. D4RL (Fu et al., 2020) is an offline RL standardized benchmark designed and commonly used to evaluate the progress of offline RL algorithms. We use 3 datasets of different quality from the Hopper task: hopper-random with 200k samples from a randomly initialized policy, hopper-medium with 200k samples from a policy trained to approximately 1/3 the performance of a policy trained to completion with SAC ("expert"), and hopper-medium-expert with 200k samples from a 50-50 split of medium and expert data. The Hopper task is to make a hopper with three joints, and four body parts hop forward as fast as possible.

A.14 TutorBot Domain

We introduce a new TutorBot simulator that is designed to mimic 3-5th grade elementary school children in understanding the concept of calculating volume, and engaging them while doing so. We base certain aspect of this simulator on some experimental studies of this learning environment. The state space includes children's pre-test score, anxiety level, thinking time, and whether it's the last question in the tutoring. The action is to offer encouragement, a guided prompt, or a hint at each step of the tutoring.

TutorBot	Dimension	Description
State	4	Pre-test $\in \{0, 1, ..., 8\}$, Anxiety-level $\in [-1, 0]$ Thinking $\in [0, 1]+$, Pre-termination $\in \{0, 1\}$
Action	1	0 = Encourage, 1 = Guided Prompt, 2 = Hint
Reward	1	0 for all steps if not last step

Table A.7: MDP specification for TutorBot.

The dynamics of TutorBot is a 4th-order Markov transition function that takes in anxiety and the amount of thinking time and updates a latent parameter that captures learning progress. For each simulated student learning trajectory, we pre-determine how many times this student will interact with the TutorBot. We denote this as T, which is the trajectory length. We calculated the relationship between T and the pre-test score based on the aforementioned experimental study.

$$T = \text{round}(7 - 0.46 * \text{pre-test} + l), l \sim U([-1, 2])$$

$$\theta_x = [0, -0.05, -0.2, -0.5], \theta_h = [0.5, 0.3, 0.2, 0]$$

$$T(s_{t+1}|s_t, a_t) = \Big[\text{pre-test}, [s_{t-3}, s_{t-2}, s_{t-1}, s_t]\theta_x^T, [s_{t-3}, s_{t-2}, s_{t-1}, s_t]\theta_h^T, \mathbb{1}\{t+1=T\}\Big]$$

The reward is always 0 at all steps except for the final step. We use x to denote anxiety and h to denote thinking. Note that anxiety is always negative. We calculate the final reward as follows:

$$R_T = \mathbb{1}\{U[0,1] < p\} * r_{\text{improv}} + (1 - \mathbb{1}\{U[0,1] < p\}) * r_{\text{base}}, p = x + h$$

Under this simulator, a student will improve a small amount even if the chatbot fails to teach optimally.

$$r_{\text{improv}} \sim \mathcal{N}(\mu_{\text{improv}}, 1), r_{\text{base}} \sim \mathcal{N}(\mu_{\text{base}}, 0.4)$$

We provide the full simulator code in the GitHub repo.

A.15 Sepsis-POMDP and Sepsis-MDP Experiment Details

Our algorithm-hyperparameter search is trying to be as realistic as possible to the setting of offline RL practitioners. We search over hyperparameters that could potentially have a strong influence on the downstream performance. Since this is an offline RL setting, we are particularly interested in searching over hyperparameters that have an influence on how pessimistic/conservative the algorithm should be.

A.15.1 BCQ

Batch Constrained Q-Learning (BCQ) is a commonly used algorithm for batch (offline) reinforcement learning (Fujimoto et al., 2019). We search over the following hyperparameters:

BCQ	Hyperparameter Range
Actor/Critic network dimension	[32, 64, 128]
Training Epochs	[15, 20, 25]
BCQ Threshold δ	[0.1, 0.3, 0.5]

Table A.8: BCQ Hyperparams for Spesis-POMDP N=200, 1000. Sepsis-MDP N=200. TutorBot N=200.

BCQ threshold determines if the Q-network can take the max over action to update its value using (s, a) – it can only update Q-function using (s, a) if $\mu(s) > \delta$ and $\pi(a|s) > 0$. The higher δ (BCQ threshold) is, the less data BCQ can learn from. δ determines whether $(s', a') \in \mathcal{B}$.

$$\begin{aligned} Q(s,a) \leftarrow & (1-\alpha)Q(s,a) \\ & + \alpha(r + \gamma \max_{a' \text{s.t.} (s',a') \in \mathcal{B}} Q'(s',a')) \end{aligned} \tag{8}$$

We search through the cross-product of these, in total 27 combinations.

For Sepsis-POMDP N=5000, we realize the network size is too small to fit a relatively large dataset of 5000 patients. So we additionally search over Table A.9. The actor/critic network uses a 2-layer fully connected network. This resulted in 6 additional combinations for BCQ in Sepsis-POMDP N=5000.

A.15.2 MBS-QI

The MBS-QI algorithm is very similar to BCQ, but MBS-QI also clips the states (Liu et al., 2020). We searched through similar hyperparameters as BCQ.

The beta (β) hyperparameter in MBS-QI is a threshold for the VAE model's reconstruction loss. When the reconstruction loss of the next state is larger than beta, MBS-QI will not apply the Q

BCQ	Hyperparameter Range
Actor/Critic network dimension	[256, 256], [512,512], [1024,1024]
Training Epochs	[25]
VAE Latent Dim	[512]
BCQ Threshold δ	[0.3, 0.4]

Table A.9: BCQ Hyperparams for Spesis-POMDP N=5000.

MBS-QI	Hyperparameter Range
Actor/Critic network dimension	[32, 64, 128]
Training Epochs	[15, 20, 25]
BCQ Threshold δ	[0.1, 0.3, 0.5]
Beta β	[1.0, 2.0, 4.0]

Table A.10: MBS-QI Hyperparams for Spesis-POMDP N=200, 1000. Sepsis-MDP N=200. TutorBot N=200.

function on this next state to compute future reward (to avoid function approximation over unfamiliar state space).

$$\begin{aligned}&\zeta(s,a;\widehat{\mu},b) = \mathbb{1}(\widehat{\mu}(s,a) \geq \beta)\\&(\tilde{\mathcal{T}}f)(s,a) := r(s,a) + \gamma\mathbb{E}_{s'}[\max_{a'} \zeta \circ f(s',a')]\end{aligned} \tag{9}$$

We search through the cross-product of these, in total 81 combinations. Similar to BCQ situation, we realize the network size is too small to fit a relatively large dataset of Sepsis-POMDP N=5000. So we additionally search over Table A.11. The actor/critic network uses a 2-layer fully connected network. This results in 18 additional combinations for MBS-QI in Sepsis-POMDP N=5000.

A.15.3 MOPO

We also experiment with Model-based Offline Policy Optimization (MOPO) (Yu et al., 2020). The original MOPO paper only experimented on Mujoco-based locomotion continuous control tasks. We want to experiment with whether MOPO can work well in environments like the Sepsis-POMDP simulator, which is not only a healthcare domain but also partially observable with a discrete state and action space. We do not expect MOPO to do well. We re-implemented two versions of MOPO with Tensorflow 2.0 and PyTorch, and used the PyTorch version to run our experiments. Our implementation of MOPO matches the original's performance in a toy environment.

MOPO is fairly slow to run – because it needs first to train a model to approximate the original environment, and then sample from this model to train an RL algorithm. We did not evaluate it for Sepsis-POMDP N=5000.

Number of ensembles refers to MOPO Algorithm 2, which trains an ensemble of N probabilistic dynamics on batch data. N should be adjusted according to the dataset size. Each dynamics model is

MBS-QI	Hyperparameter Range
Actor/Critic network dimension	[256, 256], [512,512], [1024,1024]
Training Epochs	[25]
VAE Latent Dim	[512]
BCQ Threshold δ	[0.3, 0.4]
Beta ζ	[1.0, 2.0, 4.0]

Table A.11: MBS-QI Hyperparams for Spesis-POMDP N=5000.

MOPO	Hyperparameter Range
Actor/Critic network dimension dim	[32, 32], [64, 64], [128, 128]
Training Iterations	[1000, 2000, 3000]
MOPO Lambda λ	[0, 0.1, 0.2]
Number of Ensembles	[3, 4, 5]

Table A.12: MOPO hyperparameters for Spesis-POMDP N=200, 1000.

trained on $\frac{1}{N}$ of the data during each epoch.

$$\widehat{T}_i(s', r|s, a) = \mathcal{N}(\mu_i(s, a), \Sigma_i(s, a)) \tag{10}$$

MOPO λ hyperparameter controls how small we want the reward to be, adjusting for state-action pair uncertainty. Generally, the more uncertain we are about (s, a), the more we should ignore the reward that's outputted by the learned MDP model. Its use is also described in Algorithm 2:

$$\tilde{r}(s, a) \coloneqq r(s, a) - \lambda \max_{i=1}^{N} ||\Sigma_i(s, a)||_F \tag{11}$$

We search through the cross-product of these, in total 81 combinations.

In our initial experiments, MOPO does not seem to perform well in a tabular setting where both state and action are discrete. Therefore, we simplified the idea of MOPO to introduce Pessimistic Ensemble MDP (P-MDP).

A.15.4 P-MDP

As noted in the previous section, inspired by MOPO and MoREL (Kidambi et al., 2020), we develop a tabular version of MOPO. We instantiate N tabular MDP models. For each epoch, each MDP model only updates on $1/N$ portion of the data. During policy learning time, for each timestep, we randomly sample 1 of the N MDP for the next state and reward; and use Hoeffding bound to compute a pessimistic reward, similar to MOPO's variance penalty on reward:

Let $N(s, a)$ be the number of times (s, a) is observed in the dataset:

$$\epsilon = \beta * \sqrt{\frac{2\log(1/\delta)}{N(s,a)}} \tag{12}$$
$$\tilde{r}(s, a) = \min(\max(r - \epsilon, -1), 1)$$

In the last step we bound the reward to (-1, 1) for the Sepsis setting – but it can be changed to apply to any kind of reward range. We note that Hoeffding bound is often loose when $N(s, a)$ is small, therefore, might make the reward too small to learn any good policy. However, empirically, we observe that in the Sepsis-POMDP, P-MDP is often the best-performing algorithm. We additional add a temperature hyperparameter α, that changes the peakness/flatness of the softmax distribution of the learned policy:

P-MDP	Hyperparameter Range
Training Iterations	[1000, 5000, 10000]
Penalty Coefficient β	[0, 0.1, 0.5]
Number of Ensembles	[3, 5, 7]
Temperature α	[0.05, 0.1, 0.2]

Table A.13: P-MDP Hyperparams for Spesis-POMDP N=200, 1000.

Not surprisingly, since planning algorithms (such as Value Iteration or Policy Iteration) need to enumerate through the entire state space, we find it too slow to train a policy in Sepsis-MDP domain, because Sepsis-POMDP has 144 unique states, yet Sepsis-MDP has 1440 unique states (glucose level has 5 unique states and diabetes status has 2 unique states). TutorBot and Robomimic both have continuous state space, therefore are not suitable for our P-MDP algorithm without binning.

We search through the cross-product of these, in total 81 combinations.

For Sepsis-POMDP N=5000, we realize we can increase the number of MDPs and increase training iterations to fit a relatively large dataset of 5000 patients. So we additionally search over Table A.14. This results in 16 additional combinations for P-MDP in Sepsis-POMDP N=5000.

P-MDP	Hyperparameter Range
Training Iterations	[20000, 40000]
Penalty Coefficient β	[0.05, 0.1]
Number of Ensembles	[15, 25]
Temperature α	[0.01, 0.05]

Table A.14: P-MDP Hyperparams for Spesis-POMDP N=5000.

A.15.5 BC

Behavior Cloning (BC) is a type of imitation learning method where the policy is learned from a data set by training a policy to clone the actions in the data set. It can serve as a great initialization strategy for other direct policy search methods which we will discuss shortly.

One pessimistic hyperparameter we can introduce to behavior cloning is similar in spirit to BCQ and MBS-QI, we can train BC policy only on actions that the behavior policy has a high-enough

probability to take, optimizing the following objective:

$$\zeta = \pi_b(a|s) \geq \alpha$$
$$\arg\min_{\theta} E_{(s,a)\sim D}||\pi_\theta(s) - \zeta \circ \pi_b(a|s)||^2 \tag{13}$$

We refer to α as the "safety-threshold". We search through the cross-product of these, in total 27 combinations.

BC	Hyperparameter Range
Policy network dimension	[32, 32], [64, 64], [128, 128]
Training Epochs	[15, 20, 25]
Safety Threshold α	[0, 0.01, 0.05]

Table A.15: BC Hyperparams for Spesis-POMDP N=200, 1000, 5000. Sepsis-MDP N=200. TutorBot N=200.

A.15.6 POIS

Policy Optimization via Importance Sampling (Metelli et al., 2018) uses an importance sampling estimator as an end-to-end differentiable objective to directly optimize the parameters of a policy. In our experiment, we refer to this as the "**PG**" (policy gradient) method. Similar to BC method, we can set a safety threshold α that zeros out any behavior probability of an action that's not higher than α, and then re-normalizes the probabilities of other actions. Metelli et al. (2018) also introduces another penalty hyperparameter λ to control the effective sample size (ESS) penalty. ESS measures the Renyi-divergence between π_b and π_e. Let $\widehat{V}$ be the differentiable importance sampling estimator – we write the optimization objective similar to Futoma et al. (2020), but without the generative model:

$$\mathcal{J}(\mathcal{D}_{\text{train}}) = \widehat{V}(\pi_\theta; \mathcal{D}_{\text{train}}) - \frac{\lambda_{\text{ESS}}}{\text{ESS}(\theta)}$$
$$\theta = \arg\max_{\theta} \mathcal{J}(\mathcal{D}_{\text{train}}) \tag{14}$$

We search through the following hyperparameters in Table A.16. There are 81 combinations in total.

BC	Hyperparameter Range
Policy network dimension	[32, 32], [64, 64], [128, 128]
Training Epochs	[15, 20, 25]
Safety Threshold α	[0, 0.01, 0.05]
ESS Penalty λ	[0, 0.01, 0.05]

Table A.16: POIS, BC+POIS, BC+mini-POIS Hyperparams for Spesis-POMDP N=200, 1000, 5000. Sepsis-MDP N=200. TutorBot N=200.

A.15.7 BC+POIS

BC + POIS is a method that first finds a policy using BC as an initialization strategy to make sure that the policy stayed close (at first) to the behavior policy. This is particularly useful for neural network-based policy classes, as a form of pre-training using behavior cloning objective. We use the same set of hyperparameters displayed in Table A.16, resulting in 81 combinations in total.

A.15.8 BC+mini-POIS

In both Metelli et al. (2018) and Futoma et al. (2020), the loss is computed on the whole dataset $\mathcal{D}_{\text{train}}$, which makes sense – importance sampling computes the expected reward (which requires averaging over many trajectories to have an estimation with low variance). However, inspired by the success of randomized optimization algorithms such as mini-batch stochastic gradient descent (SGD), we decided to attempt a version of BC + POIS with $\widehat{V}$ over a small batch of trajectories instead of over the entire dataset. Our batch size is 4 (4 trajectories/patients) for Sepsis-POMDP N=200 and 1000, which is very small. However, this strategy seems to be quite successful, resulting in learning high-performing policies competitive with other more principled methods. This can be seen in Figure A.2 ("BCMINIPG").

We leave the exploration of why this is particularly effective to future work, and hope others who want to try POIS style method to include our variant in their experiment. We use the same set of hyperparameters displayed in Table A.16, resulting in 81 combinations in total.

A.16 TutorBot Experiment Details

The details of this environment is shown in the code file in the supplementary material. We trained BC+POIS, POIS, and BC+mini-POIS on this domain.

A.17 Robomimic Experiment Details

We refer the reader to Mandlekar et al. (2021) for a full review of the offline RL algorithms used in our experiment. For Robomimic, we include the range of hyperparameters we have considered below:

- BC:
  - Actor NN dimension: [300,400], [1024,1024]
  - Training epochs: 600, 2000
  - GMM actions: 5, 25
- BCRNN:
  - RNN dimension: [100], [400]
  - Training epochs: 600, 2000
  - GMM actions: 5, 25
- BCQ:
  - Critic NN size: [300,400], [1024,1024]
  - Training epochs: 600, 2000
  - Action samples: [10,100], [100,1000]
- CQL:
  - Critic NN size: [300,400], [1024,1024]
  - Training epochs: 600, 2000
  - Lagrange threshold: 5, 25
- IRIS:
  - Critic NN size: [300,400], [1024,1024]
  - Training epochs: 600, 2000
  - LR critic: 0.001, 0.0001

A.18 D4RL-Hopper Experiment Details

For the D4RL experiments, we include the range of hyperparameters we have considered below:

- BCQ:
  - Policy NN size: [512,512], [64,64]
  - LR policy: 0.001, 0.0001

- CQL:
  - Policy NN size: [256,256,256], [64,64,64]
  - LR policy: 0.001, 0.0001
- AWAC:
  - Policy NN size: [256,256,256,256], [64,64,64,64]
  - LR policy: 0.001, 0.0001

For BVFT Strategy 1 π x FQE, we use the optimal FQE hyperparameter on all hyperparameters of BCQ, CQL and AWAC. For BVFT Strategy 2 π + FQE, we use 4 FQE hyperparameters but only with 4 hyperparameters of BCQ. For RRS and CV, we use the optimal FQE hyperparameter on 4 hyperparameters of BCQ as well.

A.19 Computing Resources

For the overall experimental study in this paper, an internal cluster consisting of 2 nodes with a total of 112 CPUs and 16 GPUs was used.

A.20 Additional Offline RL Sensitivity Study

A.20.1 Sensitivity to data splitting: One-Split OPE

Figure A.1 shows that procedure produces policies with drastically different estimated, and true, performances subject to randomness in data selection process. Because training and validation set randomness are directly conflated, it becomes difficult to accurately select a better `AH` pair (and its associated higher-performing policy) based on a single train/validation set partition.

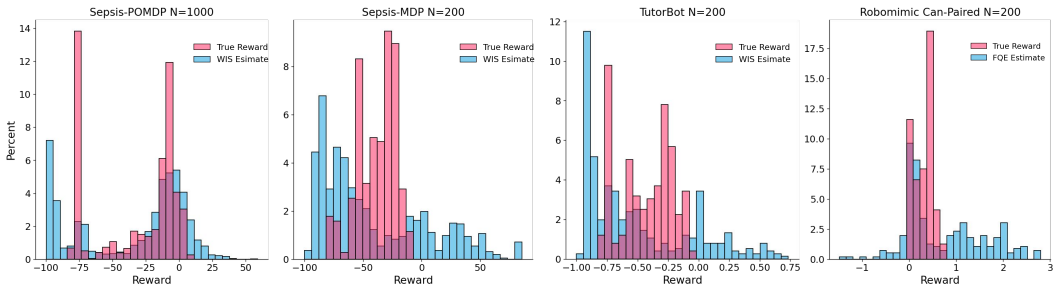

Figure A.1: We show that policies learned from offline RL algorithms are sensitive to the variation of training and validation dataset: an algorithm-hyperparameter (AH) pair can obtain wildly different policies based on which portion of the data they were trained on. We obtained 5400 policies from 540 AH combinations on Sepsis-POMDP (N=1000) domain. The variation is not just in terms of the policy's true performance in the real environment, but also in terms of OPE estimations. Note that FQE estimate on Robomimic exceeded the range of possible achievable rewards (between 0 and 1). The true reward is calculated by evaluating the policy in the real environment.

A.20.2 Sensitivity to hyperparameters

In Figure A.2, we show that offline RL algorithms are sensitive to the choice of hyperparameters. In the Sepsis-POMDP N=1000 task and the Robomimic Can-Paired N=200 task, all popular offline RL algorithms show a wide range of performance differences even when trained on a fixed partition of the dataset.

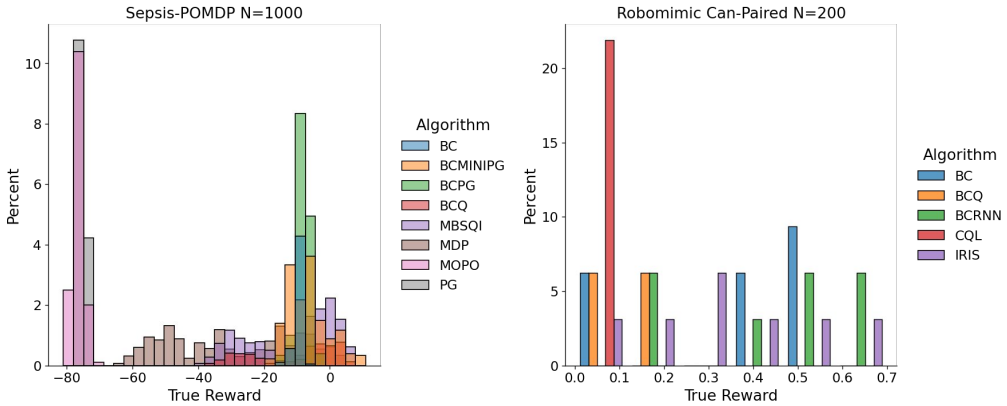

Figure A.2: Sensitivity of offline RL algorithms due to the choice of hyperparameters.