# Workflow Experiments

## Simulations Goals

The primary goal of this paper is to provide a practical guide to solving real-world offline reinforcement learning (ORL) problems for applied researchers. To this end, section 4 of the overleaf document proposes a potential workflow indicating effective ways to train policies as well as perform offline hyperparameter tuning (OHT) for different kinds of situations. These recommendations do not need to perform optimally in all scenarios. Rather, we only need our proposed strategies to perform decently on most scenarios. We can also spend some time in our paper discussing other promising ORL and OHT strategies not in our workflow for applied researchers to consider or future benchmarks to explore. Tuning the MDP (including the state, reward and discount factor) is important but orthogonal to the ORL and OHT strategy. Therefore, we can safely assume a known MDP for now when investigating effective ORL and OHT strategies, and extend our workflow to tune the MDP at a later time.

Our simulations serve three purposes. First, previous research has focused almost exclusively on benchmarking different training algorithms. In contrast, our simulations will investigate the best-performing OHT strategies for each training algorithm and situation. Second, most of the proposed training algs have only been evaluated with hyperparms tuned using online interactions, and relatedly most popular benchmarks for ORL have only been solved with online hyperparm tuning. In contrast, our simulations will demonstrate that popular training algorithms can solve standard RL problems even when an online simulator or previous literature is unavailable to choose hyperparms. Third, as our simulations will be using offline datasets to solve control problems from scratch, and as we are not aware of previous demonstrations doing this in recent literature, our empirical experiments can serve as a guide in teaching readers how they should solve their own ORL problems.

## FQI Experiments on CartPole and Healthcare

For discrete control, high coverage, low-dimensional datasets we recommend *fitted Q-iteration (FQI)* as the training algorithm. Recall that FQI consists of updates $\hat{Q}_{k+1} \leftarrow \text{argmin}_{Q\in\mathcal{Q}}\mathbb{E}[(R_t + \gamma \max_{a'} \hat{Q}_k(S_{t+1}, a') - Q(S_t, A_t))^2]$. FQI allows for arbitrary function approximators, including tree-based ensembles and neural networks. Moreover, as FQI solves a regression problem at each iteration, the regression algorithm can be re-tuned at each iteration to minimize mean squared error (MSE) on a validation set (we refer to this approach as *per-iteration tuning*). Policy performance can be bounded by controlling the excess risk at each iteration (Munos 2008) giving this strategy theoretical validity. Moreover, the strategy avoids estimating additional unknown quantities. In contrast, model-based evaluation requires estimating an unknown transition kernel, *fitted Q-evaluation (FQE)* requires estimating an unknown value function and the *mean squared Bellman error (MSBE)* requires estimating an unknown Bellman backup.

Per-iteration tuning has been used in the previous literature, but only when tuning one or two hyperparms, like the minimum node size for extremely randomized trees (ERTs), with other hyperparms tuned using online interactions or previous literature (Ersnt 2005, Guez 2006). It is thus unclear how well per-iteration tuning performs when tuning a large number of hyperparms, including selecting the regression algorithm itself. Moreover, while per-iteration tuning has shown to yield decent policies (at least when restricted to tuning a single hyperparm), it has not been thoroughly benchmarked against competing OHT approaches. It is thus currently unclear whether per-iteration tuning even performs better than random policy selection.

We first apply our proposed strategy to a dataset of $n = 1000$ episodes generated by applying a completely random behavior policy to the *CartPole* environment. Here policy value is defined as mean number of time steps per the agent falls over. We used an 80-20 split for constructing the training and validation sets. The discount factor $\gamma = 0.95$ is similar to prior work. We used per-iteration tuning to select between regression algorithms $\mathcal{R}$, which included different hyperparm configurations associated with polynomial ridge regression, random forests and ERTs.

We compare per-iteration tuning to two alternative OHT strategies based on the MSBE and FQE. For these latter two OHT strategies, the candidate set $\mathcal{Q}$ include value functions estimated by FQI using regression algorithms in $\mathcal{R}$, ridge-regularized *least squared policy iteration (LSPI)* w/ polynomial basis expansions, and *residual gradient* w/ polynomial basis expansions. Note that this benchmark not only compares per-iteration tuning against other OHT strategies, but it implicitly compares FQI to LSPI and residual gradient when hyperparms are tuned offline. As to how MSBE and FQE were implemented, CartPole has deterministic dynamics, and thus an unbiased estimator of the MSBE results from using the empirical Bellman errors (empirical MSBE or EMSBE) on a validation set (Zitovsky et. al. 2023). For FQE, we consider the functional approximators in $\mathcal{R}$, with the best functional approximator selected using validation EMSBE. Finally, we compared against performance of the behavioral policy as well as the oracle (the performance we would obtain if we tuned hyperparms to maximize policy value directly).

Across 10 seeds, FQI + per-iteration tuning yielded an average policy value of 289($\pm$90). While performance did not compete with the oracle value of 443($\pm$37), it significantly outperformed that of the behavioral policy (21($\pm$4)), as well as random policy selection. Moreover, we found that as sample size increased, the performance of FQI + per-iteration became closer to the oracle. For example, w/ $n = 8000$ observed episodes, policy value increased to 471 (the truly optimal value is 500). Finally, this strategy was robust to different settings. For example, when removing ERTs w/ minimum node size one (the best-performing function approximators according to per-iteration tuning), performance only degraded to 244($\pm$59), and when removing remaining ERT candidates (which included the next best-performing function approximators), performance only degraded to 194($\pm$51). Moreover, when increasing the discount to $\gamma = 0.99$, FQI + per-iteration tuning only degraded slightly, to 255($\pm$67).

In contrast, validation EMSBE performed worse and was far less robust to the candidate set and discount factor. For example, with $\gamma = 0.95$, policy value was 244($\pm$65), and this declined to 50($\pm$17) w/ $\gamma = 0.99$ (which is worse than random policy selection). The work of Zitovsky et. al. 2023 suggest that the MSBE will perform well provided $\mathcal{Q}$ include value functions that are sufficiently accurate. However, we were unable to improve EMSBE's performance on the $\gamma = 0.99$ case despite trying to a wide variety of regression algorithms for FQI, including ridge-regularized polynomial regression, random forests, ERTs, XGBoost, KNN and support vector regression. Increasing the sample size to $n = 8000$ also did not lead to acceptable performance. We note that the EMSBE consistently chose linear policies estimated by residual gradient, which directly minimizes EMSBE on the training set. Thus it is possible that combining more complex function approximators with residual gradient would lead to better OHT performance by validation EMSBE. Nonetheless, the fact that FQI combined with state-of-the-art supervised learning methods and very large datasets did not yield accurate enough value functions is concerning.

FQE performed similarly to EMSBE, and it tended to select similar policies as well. Following the analysis of other works (Zitovsky et. al. 2023), FQE is very sensitive to its hyperparms, and tends to choose ORL training algorithms with similar hyperparms. As we are tuning FQE w/ EMSBE and the EMSBE is condcuting suboptimal OHT, this leads FQE to be suboptimal. Finally, we note that, because CartPole has determinstic dynamics, the MSBE can be estimated quickly, easily and with high accuracy. On stochastic environments, an easy-to-compute unbiased estimator of the MSBE does not exist. In these cases, Bellman error-based methods suffer from additional estimation error and their implementation would be much more time-consuming, making them even less appealing.

We then applied FQI + per-iteration to simulated micro-randomized trials w/ $n = 30$ patient trajectories of length $T = 25$, based on the generative model of Luckett et. al. 2019. Here we used FQI + per-iteration tuning using a similar regression algorithm set $\mathcal{R}$ that included ridge-regularized polynomial regression and ERTs. We used $\gamma = 0.9$ similar to prior work. Across 10 seeds, our strategy achieved a mean reward of 4.99($\pm$0.07), while the maximum achievable mean reward is only slightly over 5. Thus per-iteration tuning appears to perform well even when the validation set is very small.

While FQI traditionally uses the same function approximator and training algorithm at each iteration, applying per-iteration tuning means different function approximators can be used for different iterations. Alternating function approximators at each iteration hinders convergence and can lead to training instabilities. Moreover, FQI is non-convex and is sensitive to initial values. Thus, if the regression algorithm selected at the first iteration is a poor choice, it could degrade performance even many iterations later. These issues were likely not noticed by prior work, as prior work used this strategy to tune only one or two hyperparms, where regression algorithms are more similar between iterations. To deal with potential training instabilities and poor initial values, we applied a simple heuristic which we dub the *homogenous heuristic*: If a particular regression algorithm is chosen by per-iteration tuning for most iterations (e.g. >95% of iterations), we recommend re-running FQI using that regression algorithm for *all* iterations. Such a strategy led to more robust performance (median performance was similar but minimum performance improved), and while such gains were only moderate, they were consistent across settings (environments, dataset sizes, discount factors, candidate sets). For example, on CartPole, the homogeneous heuristic increased the minimum policy value over 10 seeds from 171 to 186.

## RvS-G experiments on Mountain Car

For goal-directed problems we recommend *RvS-G* as the training algorithm. Recall that RvS-G estimates a policy by conditional maximum likelihood $\hat{\pi} = \text{argmin}_{\pi \in \Pi} \mathbb{E}[\pi(a|s, s_f)]$ where $\pi$ is a distribution over actions conditioned on both the current state $s$ as well as a future state $s_f$. The deployed policy at state $s$ is then $\pi(a|s, s_g)$ where $s_g$ is the goal state we wish to reach. Unlike traditional RL algorithms, RvS-G does not require specification of a discount factor, reward function or Markovian state, and its hyperparms can be tuned in principle by computing the negative log-likelihood (NLL) over a held-out validation set. While RvS-G has achieved SOTA performance on a wide variety of goal-directed problems, previous literature has tuned its hyperparms using online interactions. It is thus unclear how well RvS-G will perform if hyperparms are tuned using NLL.

To this end, we run experiments on the *mountain car (MC)* environment. Using a random behavior policy for MC led to zero succesful trajectories, and thus we instead used an auto-correlated policy that executed a random action with probability $\epsilon = 0.1$ and executed the previous action w/ prob $1 - \epsilon$. Such a behavior policy is similar to that used in previous locomotion problems (Lillicrap et. al. 2016, Khan and Abbeel 2021) to encourage agents to deviate from the initial state and better explore the environment. Such a policy can also be thought of as a very naive control attempt based on basic knowledge of the system at hand. This leads to a success rate of around 6% and a very sparse reward signal (a non-zero reward occurred only once every 1000 transitions). Here a single dataset of $n = 1000$ episodes were generated. The candidate policies $\mathcal{M}$ included polynomial logistic regression and probability random forests.

Though RvS-G performed well when hyperparms were tuned using online interactions, validation NLL correlated poorly with policy performance and its usage led to a policy w/ a success rate of only 25%. In contrast, the best-performing policy achieved a success rate of over 85%, and even a policy obtained by random selection performed better. We tried putting more accurate estimators in $\mathcal{M}$, including XGBoost classifiers, but validation NLL still failed to select high-performing policies. We also tried increasing the sample size by a factor of 10, but this also only led to marginal improvements. Finally, we tried different implementations of RvS-G, including alternative conditioning strategies (conditioning on a random future state vs. the final state vs. a binary indicator of whether the current trajectory is succesful), policy classes (deterministic vs. stochastic policies) and validation metrics (NLL vs. classification error).

Conditioning on a binary success indicator performed much better than the original implementation of conditioning on a specific state, leading to a much higher success rate of 62%, but this was mainly due to improvements in the training algorithm. In terms of validation NLL, it still performed worse than random policy selection (the latter of which yields a median performance of 70%). Further verifying the suboptimality of validation NLL was the fact that not using any conditioning information led to validation NLL changes of $< 10^{-3}$, despite the success rate dropping to under 7%. In contrast, when using a discount factor of $\gamma = 0.98$ consistent with prior literature, FQI + per-iteration tuning obtained a policy with a 100% success rate, as did FQI w/ most static function approximators.

These results contradict our initial recommendations and suggest that even for goal-directed problems, traditional RL methods may perform better. We also recall that while RvS-G performed well in previous literature, tuning hyperparms w/ validation NLL did not. We were hoping this was due to flaws in previous experiments, but it seems that this was not the case. Upon further investigation, we found that the estimated policies when conditioning on future states is very close to those estimated when not performing such conditioning. This suggests that the conditioned future states are providing a very weak learning signal, and partially explains why validation NLL is not effective here. As the behavioral policy is stochastic and uniform across trajectories, and terminal states can occur hundreds of time steps into the future, observed actions can be highly suboptimal even if they belong to a succesful trajectory. We conjecture that for this reason, conditioning on such trajectories does not lead to an optimal policy, nor does it result in significantly different validation NLLs. To support our hypothesis, we generated a dataset using a mixture of two policies, one of which selects actions randomly and the other of which uses the best-performing RvS-G policy (>85% success rate). Now behavior of succesful trajectories is very different than that of unsuccessful trajectories, and success is due to near-optimal actions worthy of imitation instead of random chance. As a result, validation NLL becomes significantly lower when conditioning on future states vs. not conditioning, and is more correlated with performance, though it still only leads to a 76% success rate. Finally, we note that with stochastic dynamics, sub-optimality of RvS-G may be furher exaserbated.