

# CFRL: A Python library for counterfactually fair offline reinforcement learning using data preprocessing

Several Different Contributors<sup>1¶</sup>

<sup>1</sup> Several Different Departments, University of Michigan ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Open Journals](#) ↗

## Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

Reinforcement learning (RL) aims to learn a sequential decision-making rule, often referred to as a “policy”, that maximizes some pre-specified benefit in an environment across multiple or even infinitely many time steps. It has been widely applied to fields such as healthcare, banking, and autonomous driving. Despite their usefulness, the decisions made by RL algorithms might exhibit systematic bias due to bias in the training data. For example, when using an RL algorithm to assign treatment to patients over time, the algorithm might consistently assign treatment resources to patients of some races while ignoring patients of other races. Concerns have been raised that the deployment of such biased algorithms could exacerbate the discrimination faced by socioeconomically disadvantaged groups.

To address this problem, Wang et al. (2025) extended the concept of single-stage counterfactual fairness (Kusner et al., 2018) to the multi-stage setting and proposed a data preprocessing algorithm that ensures counterfactual fairness in offline reinforcement learning. An RL policy is counterfactually fair if, at every time step, it would assign the same decisions with the same probability for an individual had the individual belong to a different subgroup defined by some sensitive attribute (such as race and gender). At its core, counterfactual fairness views the observed states and rewards as biased proxies of the (unobserved) true underlying states and rewards, where the bias can often be seen as a result of the observed sensitive attribute. In this light, the data preprocessing algorithm ensures counterfactual fairness by removing this bias from the input offline trajectories.

The CFRL library is built upon this definition of RL counterfactual fairness introduced in Wang et al. (2025). It implements the data preprocessing algorithm proposed by Wang et al. (2025) and provides a set of tools to evaluate the value and counterfactual fairness achieved by a given policy. In particular, it takes in an offline RL trajectory and outputs a preprocessed, bias-free trajectory, which could be passed to any off-the-shelf offline RL algorithms to learn a counterfactually fair policy. Additionally, it could also take in an RL policy and return its value and level of counterfactual fairness.

## Statement of Need

Many existing Python libraries implement algorithms that ensure fairness in machine learning. For example, Fairlearn (Weerts et al., 2023) and aif360 (Bellamy et al., 2018) provide tools for mitigating bias in single-stage machine learning predictions under statistical association-based fairness criterion such as demographic parity and equal opportunity. However, they do not focus on counterfactual fairness, which defines fairness from a causal perspective, and they cannot be easily extended to the reinforcement learning setting in general. Additionally, ml-fairness-gym (D’Amour et al., 2020) allows users to simulate unfairness in sequential decision-making, but it neither implement algorithms that reduce unfairness nor address counterfactual fairness. To our current knowledge, Wang et al. (2025) is the first work to

study counterfactual fairness in reinforcement learning. Correspondingly, CFRL is also the first code library to address counterfactual fairness in the reinforcement learning setting.

The contribution of CFRL is two-fold. First, it implements a data preprocessing algorithm that removes bias from offline RL training data. For each individual (or sample) in the data, the preprocessing algorithm estimates the counterfactual states under different sensitive attribute values and concatenates all of the individual's counterfactual states into a new state variable. The preprocessed data can then be directly used by existing RL algorithms for policy learning, and the learned policy should be approximately counterfactually fair. Second, it provides a platform for assessing RL policies based on counterfactual fairness. After passing in a policy and a trajectory dataset from the environment of interest, users can assess how well the policy performs in the environment of interest in terms of the discounted cumulative reward and a counterfactual fairness metric. This not only allows stakeholders to test their fair RL policies before deployment but also offers RL researchers a hands-on tool to evaluate newly developed counterfactually fair RL algorithms.

## High-level Design

The CFRL library is composed of 5 major modules. The functionalities of the modules are summarized in the table below.

Module	Functionalities
reader	Implements functions that read tabular trajectory data from either a .csv file or a pandas.DataFrame into a format required by CFRL. Also implements functions that export trajectory data to either a .csv file or a pandas.DataFrame.
preprocessor	Implements the data preprocessing algorithm introduced in Wang et al. (2025).
agents	Implements a fitted Q-iteration (FQI) algorithm, which learns RL policies and makes decisions based on the learned policy. Users can also pass a preprocessor to the FQI; in this case, the FQI will be able to take in unprocessed trajectories, internally preprocess the input trajectories, and directly output counterfactually fair policies.
environment	Implements a synthetic environment that produces synthetic data as well as a simulated environment that simulates the transition dynamics of the environment underlying some real-world RL trajectory data. Also implements functions for sampling trajectories from the synthetic and simulated environments.
fqe	Implements a fitted Q-evaluation (FQE) algorithm, which can be used to evaluate the value of a policy.
evaluation	Implements functions that evaluate the value and counterfactual fairness of a policy. Depending on the user's needs, the evaluation can be done either in a synthetic environment or in a simulated environment.

A general CFRL workflow is as follows: First, simulate a trajectory using environment or read in a trajectory using reader. Then, train a preprocessor using preprocessor to remove the bias in the trajectory data. After that, pass the preprocessed trajectory into the FQI algorithm in agents to learn a counterfactually fair policy. Finally, use functions in evaluation to evaluate the value and counterfactual fairness of the trained policy.

## 64 Data Example

65 We provide a data example to demonstrate how CFRL learns a counterfactually fair policy from  
66 real-world trajectory data with unknown underlying transition rules. We also show how CFRL  
67 evaluates the value and counterfactual fairness of the learned policy. We note that this is  
68 only one of the many workflows that CFRL can perform. For example, CFRL can also generate  
69 synthetic trajectory data and use it to evaluate the value and counterfactual fairness resulting  
70 from some custom data preprocessing methods. We refer interested readers to the “Example  
71 Workflows” section of the CFRL documentation for more workflow examples.

### 72 Data Loading

73 In this demonstration, we use an offline trajectory generated from a `SyntheticEnvironment`  
74 following some pre-specified transition rules. Although it is actually synthesized, we treat it as  
75 if it is from some unknown environment for pedagogical convenience.

76 The trajectory contains 500 individuals (i.e.  $N = 500$ ) and 10 transitions (i.e.  $T = 10$ ). The  
77 sensitive attribute variable and the state variable are both univariate. The sensitive attributes  
78 are binary (0 or 1). The actions are also binary (0 or 1) and were sampled using a policy that  
79 selects 0 or 1 randomly with equal probability. The trajectory is stored in a tabular format in a  
80 `.csv` file. We use `read_trajectory_from_csv()` to load the trajectory from the `.csv` format  
81 into the array format required by CFRL.

```
zs, states, actions, rewards, ids = read_trajectory_from_csv(
    path='../data/sample_data_large_uni.csv', z_labels=['z1'],
    state_labels=['state1'], action_label='action', reward_label='reward',
    id_label='ID', T=10)
```

82 We then split the trajectory data into a training set (80%) and a testing set (20%) using  
83 scikit-learn's `train_test_split()`. The training set is used to train the counterfactually fair  
84 policy, while the testing set is used to evaluate the value and counterfactual fairness metric  
85 achieved by the policy.

```
(
    zs_train, zs_test, states_train, states_test,
    actions_train, actions_test, rewards_train, rewards_test
) = train_test_split(zs, states, actions, rewards, test_size=0.2)
```

### 86 Preprocessor Training & Trajectory Preprocessing

87 We now train a `SequentialPreprocessor` and preprocess the trajectory. The `SequentialPreprocessor`  
88 ensures the learned policy is counterfactually fair by removing the bias from the training  
89 trajectory data. Due to limited trajectory data, the data to be processed will also be the  
90 data used to train the preprocessor, so we set `cross_folds=5` to reduce overfitting. In this  
91 case, `train_preprocessor()` will internally divide the training data into 5 folds, and each  
92 fold is preprocessed using a model that is trained on the other 4 folds. We initialize the  
93 `SequentialPreprocessor`, and `train_preprocessor()` will take care of both preprocessor  
94 training and trajectory preprocessing.

```
sp = SequentialPreprocessor(z_space=[[0], [1]], num_actions=2, cross_folds=5,
                           mode='single', reg_model='nn')
states_tilde, rewards_tilde = sp.train_preprocessor(
    zs=zs_train, xs=states_train, actions=actions_train, rewards=rewards_train)
```

95 As an aside, we remark that in the case where the trajectories to be preprocessed are  
96 separate from the trajectories used to train the preprocessor, we should typically set  
97 `cross_folds=1`. Then we use `train_preprocessor()` to train the preprocessor and use  
98 `preprocess_multiple_steps()` to preprocess the trajectories.

## 99 Counterfactually Fair Policy Learning

100 Now we train a counterfactually fair policy using the preprocessed data and FQI with `sp` as its  
101 internal preprocessor. Note that the training data `state_tilde` and `rewards_tilde` are already  
102 preprocessed. Thus, we set `preprocess=False` during training so that the input trajectory will  
103 not be preprocessed again by the internal preprocessor (i.e. `sp`).

```
agent = FQI(num_actions=2, model_type='nn', preprocessor=sp)
agent.train(zs=zs_train, xs=states_tilde, actions=actions_train,
           rewards=rewards_tilde, max_iter=100, preprocess=False)
```

## 104 SimulatedEnvironment Training

105 Before moving on to the evaluation stage, there is one more thing to do: We need to train a  
106 `SimulatedEnvironment` that mimics the transition rules of the true environment that generated  
107 the training trajectory, which will be used by the evaluation functions to simulate the true  
108 data-generating environment. To do so, we initialize a `SimulatedEnvironment` and train it on  
109 the whole trajectory data (i.e. training set and testing set combined).

```
env = SimulatedEnvironment(num_actions=2, state_model_type='nn',
                          reward_model_type='nn')
env.fit(zs=zs, states=states, actions=actions, rewards=rewards)
```

## 110 Value and Counterfactual Fairness Evaluation

111 We now use `evaluate_value_through_fqe()` and `evaluate_fairness_through_model()` to  
112 estimate the value and counterfactual fairness achieved by the trained policy when interacting  
113 with the environment of interest, respectively. The counterfactual fairness is represented by a  
114 metric from 0 to 1, with 0 representing perfect fairness and 1 indicating complete unfairness.  
115 We use the testing set for evaluation.

```
value = evaluate_reward_through_fqe(zs=zs_test, states=states_test,
                                   actions=actions_test, rewards=rewards_test, policy=agent, model_type='nn')
cf_metric = evaluate_fairness_through_model(env=env, zs=zs_test, states=states_test,
                                           actions=actions_test, policy=agent)
```

116 The estimated value is 7.358 and CF metric is 0.042, which indicates our policy is close to  
117 being perfectly counterfactually fair. Indeed, the CF metric should be exactly 0 if we know  
118 the true dynamics of the environment of interest; the reason why it is not exactly 0 here is  
119 because we need to estimate the dynamics of the environment of interest during preprocessing,  
120 which can introduce errors.

## 121 Comparisons: Assessing a Fairness-through-unawareness Policy

122 Fairness-through-unawareness proposes to ensure fairness by excluding the sensitive attribute  
123 from the agent's decision-making. However, it might still be unfair because of the indirect bias  
124 in the states and rewards. In this section, we use the same trajectory data to train a policy  
125 following fairness-through-unawareness and estimate its value and counterfactual fairness.

```
agent_unaware = FQI(num_actions=2, model_type='nn', preprocessor=None)
agent_unaware.train(zs=zs_train, xs=states_train, actions=actions_train,
                   rewards=rewards_train, max_iter=100, preprocess=False)
value_unaware = evaluate_reward_through_fqe(zs=zs_test, states=states_test,
                                           actions=actions_test, rewards=rewards_test, policy=agent_unaware,
                                           model_type='nn')
cf_metric_unaware = evaluate_fairness_through_model(env=env, zs=zs_test,
                                                    states=states_test, actions=actions_test, policy=agent_unaware)
```

The estimated value is 8.588 and CF metric is 0.446. The fairness-through-unawareness policy is much less fair than the policy learned using the preprocessed trajectory. This suggests that the preprocessing method likely reduced the bias in the training trajectory effectively. Indeed, we can evaluate the performance of more baselines using CFRL. Some code examples of such evaluations can be found in the “Assessing Policies Using Real Data” workflow in the “Example Workflows” section of the CFRL documentation.

## Conclusions

CFRL is a Python library that empowers counterfactually fair reinforcement learning through data preprocessing. It also provides tools to evaluate the value and counterfactual fairness of a given policy. As far as we know, it is the first library to address counterfactual fairness problems in the context of reinforcement learning. Nevertheless, despite this, CFRL also admits a few limitations. For example, the current CFRL implementation requires every individual in the offline dataset to have the same number of time steps. Extending the library to accommodate variable-length episodes can improve its flexibility and usefulness. Besides, CFRL could also be made more well-rounded by integrating the preprocessor with popular offline RL algorithm libraries such as `d3rlpy` (Seno & Imai, 2022), or connecting the evaluation functions with established RL environment libraries such as `gym` (Towers et al., 2024). We leave these extensions to future updates.

## Acknowledgements

This is the acknowledgements.

## References

- Bellamy, R. K. E., Dey, K., Hind, M., Hoffman, S. C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K. N., Richards, J., Saha, D., Sattigeri, P., Singh, M., Varshney, K. R., & Zhang, Y. (2018). *AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias*. <https://arxiv.org/abs/1810.01943>
- D’Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., & Halpern, Y. (2020). Fairness is not static: Deeper understanding of long term fairness via simulation studies. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 525–534. <https://doi.org/10.1145/3351095.3372878>
- Kusner, M. J., Loftus, J. R., Russell, C., & Silva, R. (2018). *Counterfactual fairness*. <https://arxiv.org/abs/1703.06856>
- Seno, T., & Imai, M. (2022). `d3rlpy`: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315), 1–20. <http://jmlr.org/papers/v23/22-0017.html>
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., & others. (2024). *Gymnasium: A standard interface for reinforcement learning environments*. *arXiv Preprint arXiv:2407.17032*.
- Wang, J., Shi, C., Piette, J. D., Loftus, J. R., Zeng, D., & Wu, Z. (2025). *Counterfactually fair reinforcement learning via sequential data preprocessing*. <https://arxiv.org/abs/2501.06366>
- Weerts, H., Dudík, M., Edgar, R., Jalali, A., Lutz, R., & Madaio, M. (2023). Fairlearn: Assessing and improving fairness of AI systems. In *Journal of Machine Learning Research* (No. 257; Vol. 24, pp. 1–8). <http://jmlr.org/papers/v24/23-0389.html>