OPT2024: 16th Annual Workshop on Optimization for Machine Learning

Policy Optimization for Strictly Batch Imitation Learning

Rishabh Agrawal RISHABHA@USC.EDU
University of Southern California

Nathan Dahlin NDAHLIN@ALBANY.EDU
University at Albany, SUNY

Rahul Jain RAHUL.JAIN@USC.EDU
University of Southern California

Ashutosh Nayyar ASHUTOSN@USC.EDU
University of Southern California

Abstract

Imitation Learning (IL) offers a compelling framework within the broader context of Reinforcement Learning (RL) by eliminating the need for explicit reward feedback, a common requirement in RL. In this work, we address IL based solely on observed behavior without access to transition dynamics information, reward structure, or, most importantly, any additional interactions with the environment. Our approach leverages conditional kernel density estimation and performs policy optimization to ensure the satisfaction of the Markov balance equation associated with the environment. This method performs effectively in discrete and continuous state environments, providing a novel solution to IL problems under strictly offline optimization settings. We establish that our estimators satisfy basic asymptotic consistency requirements. Through a series of numerical experiments on continuous state benchmark environments, we show consistently superior empirical performance over many state-of-the-art IL algorithms.

1. Introduction

Reinforcement Learning (RL) has achieved significant breakthroughs over the past decade, from surpassing human performance in games like Atari [35], Go [45], and StarCraft [52] to advancing fields such as protein structure prediction [19]. Despite these successes, a major challenge persists in real-world applications: RL depends on well-defined reward functions, which often do not exist, especially when only offline data is available. While considerable effort has been invested in designing reward models, particularly in areas like generative AI, these efforts often introduce issues such as reward hacking, over-optimization, and reduced robustness in the learned policies [46].

This challenge is particularly pronounced in human-in-the-loop systems, where demonstrators or evaluators provide data. Expert demonstrations rarely reveal a clear reward function, making it hard to infer their true objectives. While IRL methods like MaxEntropy-IRL [62] first infer a reward function from expert data and then use it with RL algorithms to design near-optimal policies, they suffer from two key drawbacks: RL performance is highly sensitive to errors in reward estimation, and the expert may not be following an optimal reward-based policy. This underscores the need for imitation learning (IL) approaches that bypass reward inference entirely [3].

Behavioral Cloning (BC) is a simple IL method that learns expert policies through supervised learning [39]. However, it overlooks the sequential nature of decision-making, which results in

covariate shift and leads to error accumulation in underexplored states [42]. Recent attempts to address these limitations involve adding online interactions or supplementary data [38, 43], but this is often impractical in real-world settings like autonomous driving or healthcare, where interactions can be costly or unsafe.

Imitation learning centers on matching the state-action distribution of expert demonstrations to that of the imitator's policy. Adversarial Imitation Learning (AIL) achieves this through adversarial optimization [12, 16, 20], but it relies on on-policy samples, making it unsuitable for offline learning where further environment interactions are not possible. Recent approaches aim to improve IL by adding cheaply generated data from suboptimal policies to supplement expert demonstrations [57, 59]. However, this can introduce distribution shifts, degrading policy performance when the data deviates from the expert distribution.

In this paper, we present an imitation learning algorithm that eliminates the need for *reward feedback*, does not rely on *distribution matching* from on-policy samples, and avoids *behavioral cloning*. Our approach leverages the Markovian nature of dynamics, requires no *generative model*, accommodates continuous state spaces, and supports batch processing of offline data. This framework is particularly relevant for applications in healthcare, robotics, and autonomous vehicles [27], where experimentation can be costly or unsafe.

We introduce a novel framework for policy optimization based on the balance equation between the demonstration policy, the Markov decision process (MDP) transition density, and the induced Markov chain. We utilize conditional kernel density estimators for estimating these densities and establish their universal consistency. We demonstrate strong performance across various tasks with continuous state spaces. Extension to continuous action spaces is conceptually straightforward but requires further work for robustness. A further review of related work is provided in Appendix A.

2. Preliminaries

The Imitation Learning Problem. An infinite horizon discounted MDP M is defined by the tuple (S, A, T, r, γ) with states $s \in S$, actions $a \in A$ and next states $s' \in S$ sampled from the transition function $T(s'|s, a)$. The reward function $r : S \times A \to \mathbb{R}$ maps state-action pairs to scalar rewards, and γ is the discount factor. Policy π is a probability distribution over actions conditioned on state and is given by $\pi(a_t|s_t) = P_\pi(A_t = a_t|S_t = s_t)$, where $a_t \in A$, $s_t \in S$, $\forall t = 0, 1, 2, \cdots$. The induced occupancy measure of a policy is given as $\rho_\pi(s, a) := \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t \mathbf{1}_{s_t=s, a_t=a}]$, where the expectation is taken over $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim T(\cdot|s_t, a_t)$ for all t, and the initial state s_0. The corresponding state-only occupancy measure is given as $\rho_\pi(s) = \sum_a \rho_\pi(s, a)$. In the offline imitation learning (IL) framework, the agent is provided with trajectories generated by a demonstration policy π_D, collected as $D = \{(s_0, a_0), (s_1, a_1), (s_2, a_2), ...\}$; and is not allowed any further interaction with the environment. The data D does *not* include any reward r_t at each time step. Indeed, rather than long-term reward maximization, the IL objective is to learn a policy $\pi^\star$ that is close to π_D in the following sense [60]:

$$\pi^\star \in \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim \rho_\pi}[\mathcal{L}(\pi(\cdot|s), \pi_D(\cdot|s))], \tag{1}$$

where Π is the set of all randomized (Markovian) stationary policies, and $\mathcal{L}$ is a chosen loss function. In practice, (1) can only be solved approximately since π_D is unknown and only transitions are observed in the dataset D.

Conditional Kernel Density Estimation (CKDE). The imitation learning approach we introduce depends on transition density estimation. Though statistical theory exists for it, conditional density estimation is a difficult problem due to a lack of clarity on what parametric families of density functions are good candidates. Thus, we adopt kernel density estimation (KDE), a nonparametric framework for the estimation of general continuous distributions [53].

We next outline the method for two continuous random variables, X and Y for the sake of simplicity. Let f and g denote the joint density of (X, Y) and the marginal density of X, respectively. The conditional distribution of Y, given X, is denoted as $h_{Y|X}(y|x) = f_{X,Y}(x, y)/g_X(x)$. Selecting a pair of kernel functions $K : \mathbb{R} \to \mathbb{R}$ and $K' : \mathbb{R} \to \mathbb{R}$ with respective scalar bandwidth parameters $h > 0$ and $h' > 0$ and given a set of n samples $\{(x_i, y_i)\}_{i=1}^n$, the KDE approximations $\hat{f}$ and $\hat{g}$ for the joint and marginal distributions, respectively, are obtained as follows:

$$\begin{aligned} \hat{f}_{X,Y}(x, y) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - x_i}{h}\right) \frac{1}{h'} K'\left(\frac{y - y_i}{h'}\right), \\ \hat{g}_X(x) &= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - x_i}{h}\right). \end{aligned} \tag{2}$$

Using the approximations in (2), the approximate conditional density $\hat{h}_{Y|X}$ can be computed as

$$\hat{h}_{Y|X}(y|x) = \frac{\hat{f}_{X,Y}(x, y)}{\hat{g}_X(x)}. \tag{3}$$

In more general cases involving random vectors, analogous estimates to those in (2) and (3) may be obtained using kernel functions defined according to

$$K_H(x) = |H|^{-\frac{1}{2}} K(H^{-\frac{1}{2}} x), \tag{4}$$

where H is a symmetric positive definite *bandwidth matrix* of appropriate dimension, m, with determinant $|H|$, and K is a real-valued function satisfying $\int_{\mathbb{R}^m} K(x)dx = 1$. For example, the KDE estimate for the marginal distribution of random vector X is defined as

$$\hat{g}_X(x; H) = \frac{1}{n}\sum_{i=1}^{n} K_H\big(x - x_i\big). \tag{5}$$

We note that conditional density estimation is quite difficult numerically, and conditional kernel density estimation (CKDE), the adaptation of KDE to conditional density estimation, is amongst the most effective methods available (see [7] for more details).

3. Conditional Kernel Imitation Learning

We present our imitation learning algorithm, which leverages a fundamental principle: demonstration trajectories from an expert must satisfy the Markov balance equation under the expert policy π_D. This guides the agent's learning process. To implement this, we estimate the required transition (conditional probability) densities using conditional kernel density estimation. The task then becomes to identify policies that best fit the balance equation. Algorithm 1 outlines this approach, and we provide theoretical guarantees and experimental results on benchmark environments.

Algorithm 1 Conditional Kernel Imitation Learning (CKIL)

Input: Expert dataset of trajectories $D = \{(s_i, a_i)\}_{i=1}^n$
Output: θ^*

1: Initialize policy parameter θ
2: Transform dataset D into (s, a, s', a') tuples, then store them in buffer B.
3: **for** $iter = 0, 1, \ldots$ **do**
4: Sample a batch b_{iter} of (s, a, s', a') tuples from B
5: Obtain $\hat{P}, \hat{T}$ in (9) via CKDE on b_{iter}
6: Calculate empirical estimate of the objective function in (8) using all $(s, a, s', a') \in b_{iter}$ as:

$$\sum_{(s',a')} \sum_{(s,a)} \left[\hat{P}(s', a'|s, a) - \pi_\theta(a'|s')\hat{T}(s'|s, a)\right]^2 + \lambda \sum_{s'} \sum_{a'} \pi_\theta(a'|s') log(\pi_\theta(a'|s')) \quad (6)$$

7: Update the policy parameter θ using gradient update to minimize the calculated empirical estimate of the objective function
8: **end for**
9: **return** θ^*

The Markov Balance Equation. Consider a demonstration policy π_D, a randomized Markovian stationary policy. This induces a Markov chain (MC) on the state state-action with transition density $P(s', a'|s, a)$. Then, the Markov balance equation is given by

$$P_{\pi_D}(s', a'|s, a) = \pi_D(a'|s')T(s'|s, a). \quad (7)$$

This balance equation forms the foundation of our IL approach. By estimating P_{π_D} and T in (7) (denoted as $\hat{P}$ and $\hat{T}$ respectively), we can deduce a policy π_D that satisfies the equation. However, due to the ill-conditioned nature of the problem, additional criteria like regularization are necessary.

We consider a class of policies parameterized by θ and set up the following optimization problem:

$$\begin{aligned} \min_{\theta \in \Theta} \int_{(s',a')} \int_{(s,a)} \left[\hat{P}(s', a'|s, a) - \pi_\theta(a'|s')\hat{T}(s'|s, a)\right]^2 d\mu(s, a)\, d\mu(s', a') \\ -\lambda \int_{s'} H(\pi_\theta(\cdot|s'))\, d\nu(s'). \end{aligned} \quad (8)$$

In (8), the first term, a squared loss, ensures approximate satisfaction of the balance equation—a novel loss function rarely used in imitation learning in conjunction with a balance equation. The second term, $H(\pi_\theta(\cdot|s'))$, represents the entropy of the action distribution $\pi_\theta(\cdot|s')$ at state s', encouraging more randomized policies. The regularization parameter $\lambda \geq 0$ controls this trade-off. Here, μ and ν are reference probability measures on state-action pairs and states, often based on empirical data, while Θ represents parameters set, such as the weights of a neural network.

Transition Density Estimation. We now discuss how to estimate the two conditional densities P_{π_D} and T using kernel density estimation methods for continuous spaces. The discussion for discrete spaces is deferred to the Appendix C.

Estimating transition densities in continuous spaces is challenging since no visited state would appear twice, and many remain unvisited in the dataset. This necessitates the use of advanced

conditional density estimation methods. Options include parametric approaches like mixture density networks [5], normalizing flows [51]; non-parametric methods like Gaussian process conditional density estimation [11], CKDE [30]; and semi-parametric methods like least squares conditional density estimation [48]. In this work, we choose CKDE for its closed-form, non-parametric nature, ease of implementation, and consistency under appropriate conditions [7].

As explained in Section 2, kernel functions rely on the difference between sample points (e.g., $x - x_i$) (5), which can be substituted by distance metrics [15]. We define three metrics: $d_1 : (S \times A) \times (S \times A) \to \mathbb{R}_+$ for (next state, next action) pairs, $d_2 : (S \times A) \times (S \times A) \to \mathbb{R}_+$ for (state, action) pairs, and $d_3 : S \times S \to \mathbb{R}_+$ for next states. Corresponding square bandwidth matrices H_1, H_2, and H_3 adjust the kernels K_{H_1}, K_{H_2}, and K_{H_3}, and the CKDE approximations $\hat{P}$ and $\hat{T}$ are then computed as

$$\hat{P}(s', a'|s, a) = \frac{\sum_{l=1}^{n} K_{H_1}\big(d_1((s', a'), (s'_l, a'_l))\big) K_{H_2}\big(d_2((s, a), (s_l, a_l))\big)}{\sum_{l=1}^{n} K_{H_2}\big(d_2((s, a), (s_l, a_l))\big)},$$
$$\text{and } \hat{T}(s'|s, a) = \frac{\sum_{l=1}^{n} K_{H_3}\big(d_3(s', s'_l)\big) K_{H_2}\big(d_2((s, a), (s_l, a_l))\big)}{\sum_{l=1}^{n} K_{H_2}\big(d_2((s, a), (s_l, a_l))\big)}. \tag{9}$$

We integrate the transition estimation methods from (9) with the Markov balance equation optimization from (8) in our conditional kernel imitation learning (CKIL) algorithm, detailed in Algorithm 1.

Theoretical Guarantees. Given standard assumptions, we demonstrate that as the training dataset size n approaches infinity, the CKDE estimates in (9) converge in probability to the true conditional distributions in Theorem 1.

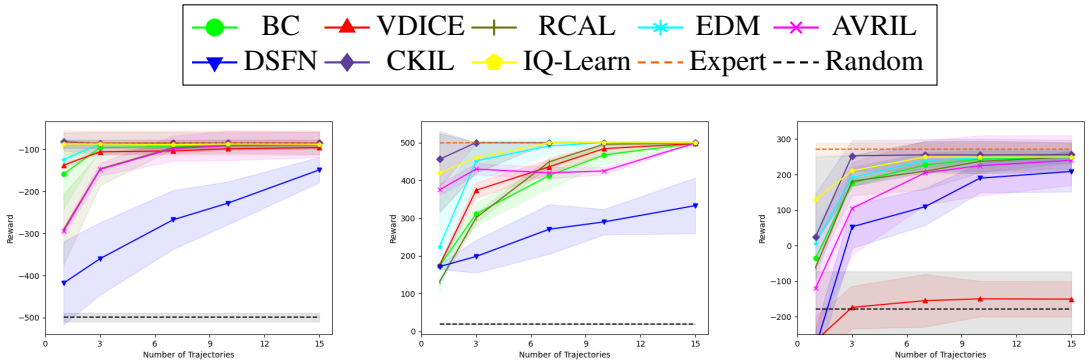
Theorem 1 *Let $\widehat{P}_n$ and $\widehat{T}_n$ be the CKDE estimates constructed using (9) and a buffer B with n tuples. Then, under appropriate conditions, for each (s, a, s', a'), as $n \to \infty$,*

$$\widehat{P}_n(s', a'|s, a) \xrightarrow{P} P_{\pi_D}(s', a'|s, a), \quad \text{and } \widehat{T}_n(s'|s, a) \xrightarrow{P} T(s'|s, a). \tag{10}$$

We prove the Theorem 1 and outline the assumptions made therein in the Appendix B.

4. Experimental Results

Experimental Setup and Benchmark Algorithms. We evaluate our algorithm's performance in various benchmark environments from OpenAI Gym [6], with details on the environments and data collection provided in Appendix D. The policy implementation, kernel choice, and bandwidths for CKIL are discussed in Appendix E. We compare CKIL (Algorithm 1) against several offline IRL/IL/AIL baselines such as Behavioral Cloning (BC), ValueDICE (VDICE) [25], reward-regularized classification (RCAL) [37], Energy-based Distribution Matching (EDM) [17], AVRIL [9], Deep Successor Feature Network (DSFN) [29], and IQ-Learn [13], a state-of-the-art model-free offline IRL algorithm. While newer methods like CLARE [58] exist, many either require additional diverse data or allow for environment interactions, diverging from our strict assumption of using only expert data without further interactions. Thus, we exclude these methods to ensure a fair comparison. More details on the baselines and hyperparameters for CKIL can be found in Appendices G and F, respectively.



(*a*) Acrobot (*b*) CartPole (*c*) LunarLander

Figure 1: Average rewards achieved by benchmark IRL/IL/AIL and CKIL policies during real-time deployment plotted against the number of trajectories included in demonstration dataset D (higher values indicate better performance).

Results. Figure 1 illustrates the average rewards all algorithms achieve as the demonstration dataset size increases in the Acrobot, CartPole, and LunarLander environments. CKIL consistently outperforms the baseline algorithms across all tasks, particularly in data-scarce scenarios, demonstrating its ability to learn effective policies. Notably, CKIL achieves expert-level performance in the CartPole, Acrobot, and LunarLander environments using just three trajectories and impressively reaches near-expert performance in CartPole with only a single trajectory. While the IQ-Learn algorithm shows performance close to CKIL's, it incurs significantly higher computational costs due to its IRL nature, and its instabilities are discussed in [2]. In contrast, the off-policy adaptations of online algorithms (VDICE, DSFN) do not maintain the same level of consistent performance as their offline counterparts, underscoring the limitations of using online algorithms in offline contexts. Additionally, VDICE's potential underperformance compared to the behavioral cloning baseline may stem from challenges in estimating the expectation of an exponential distribution. Moreover, ablation study is done in Appendix H.

5. Conclusions

In this paper, we presented Conditional Kernel Imitation Learning (CKIL), a novel approach for policy optimization in strictly offline imitation learning settings. CKIL eliminates the need for reward modeling by utilizing the Markov balance equation and conditional kernel density estimators to estimate transition densities in the MDP and induced Markov chain. The algorithm demonstrates strong empirical performance against nearly all state-of-the-art offline IL, IRL, and AIL algorithms and is supported by theoretical consistency results. Importantly, CKIL does not require access to a generative model or additional datasets, distinguishing it from many other imitation learning methods. While imitation learning algorithms often struggle with distribution shifts when trained on limited datasets, our experimental results indicate that CKIL manages this challenge more effectively than other strictly batch methods. For more details, please see Appendix I. Future work could explore density estimation methods, such as normalizing flows, and develop non-asymptotic sample complexity bounds, which are rarely available in current IL literature.

References

[1] Ibrahim A Ahmad and Iris S Ran. Data based bandwidth selection in kernel density estimation with parametric start via kernel contrasts. *Journal of Nonparametric Statistics*, 16(6):841–877, 2004.

[2] Firas Al-Hafez, Davide Tateo, Oleg Arenz, Guoping Zhao, and Jan Peters. Ls-iq: Implicit reward regularization for inverse reinforcement learning. *arXiv preprint arXiv:2303.00599*, 2023.

[3] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[4] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[5] Christopher M. Bishop. Mixture density networks. Technical report, Aston University, 1994.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[7] José E. Chacón and Tarn Duong. *Multivariate kernel smoothing and its applications*. Chapman and Hall/CRC, 2018.

[8] Alex James Chan and Mihaela van der Schaar. Scalable bayesian inverse reinforcement learning. https://github.com/XanderJC/scalable-birl, 2021.

[9] Alex James Chan and Mihaela van der Schaar. Scalable bayesian inverse reinforcement learning. In *International Conference on Learning Representations*, 2021.

[10] Jonathan D Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data without great coverage. *arXiv preprint arXiv:2106.03207*, 2021.

[11] Vincent Dutordoir, Hugh Salimbeni, James Hensman, and Marc Deisenroth. Gaussian process conditional density estimation. *Advances in neural information processing systems*, 31, 2018.

[12] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[13] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. *Advances in Neural Information Processing Systems*, 34:4028–4039, 2021.

[14] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. https://github.com/Div99/IQ-Learn, 2021.

[15] Bernard Haasdonk and Claus Bahlmann. Learning with distance substitution kernels. In *Joint pattern recognition symposium*, pages 220–227. Springer, 2004.

[16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

[17] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Strictly batch imitation learning by energy-based distribution matching. *Advances in Neural Information Processing Systems*, 33: 7354–7365, 2020.

[18] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Strictly batch imitation learning by batch imitation learning by energy-based distribution matching. https://github.com/vanderschaarlab/mlforhealthlabpub/tree/main/alg/edm, 2020.

[19] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[20] Liyiming Ke, Sanjiban Choudhury, Matt Barnes, Wen Sun, Gilwoo Lee, and Siddhartha Srinivasa. Imitation learning as f-divergence minimization. In *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*, pages 313–329. Springer, 2021.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

[22] Edouard Klein, Matthieu Geist, and Olivier Pietquin. Batch, off-policy and model-free apprenticeship learning. In *European Workshop on Reinforcement Learning*, pages 285–296. Springer, 2011.

[23] Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. *Advances in neural information processing systems*, 25, 2012.

[24] Oleg Klimov. Openai gym: Rocket trajectory optimization is a classic topic in optimal control. https://github.com/openai/gym, 2019.

[25] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. *arXiv preprint arXiv:1912.05032*, 2019.

[26] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. https://github.com/google-research/google-research/tree/master/value_dice, 2020.

[27] Luc Le Mero, Dewei Yi, Mehrdad Dianati, and Alexandros Mouzakitis. A survey on imitation learning techniques for end-to-end autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):14128–14147, 2022.

[28] Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly batch apprenticeship learning with deep successor features. https://github.com/dtak/batch-apprenticeship-learning, 2019.

[29] Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly batch apprenticeship learning with deep successor features. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 5909–5915, 2019.

[30] Qi Li and Jeffrey Scott Racine. *Nonparametric Econometrics: Theory and Practice*. Number 8355 in Economics Books. Princeton University Press, 2006.

[31] Minghuan Liu, Tairan He, Minkai Xu, and Weinan Zhang. Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*, 2020.

[32] Enno Mammen, Maria Dolores Martinez Miranda, Jens Perch Nielsen, and Stefan Sperlich. Do-validation for kernel density estimation. *Journal of the American Statistical Association*, 106(494):651–660, 2011.

[33] Henry B Mann and Abraham Wald. On stochastic limit and order relationships. *The Annals of Mathematical Statistics*, 14(3):217–226, 1943.

[34] Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7(12):2651–2667, 2006.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[36] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

[37] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted and reward-regularized classification for apprenticeship learning. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, page 1249–1256, 2014.

[38] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Bridging the gap between imitation learning and inverse reinforcement learning. *IEEE transactions on neural networks and learning systems*, 28(8):1814–1826, 2016.

[39] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1:305–313, 1988.

[40] Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[41] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: imitation learning via regularized behavioral cloning. *arXiv preprint arXiv:1905.11108*, 2(5), 2019.

[42] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[43] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635, 2011.

[44] Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[45] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484–489, 2016.

[46] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[47] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

[48] Masashi Sugiyama, Ichiro Takeuchi, Taiji Suzuki, Takafumi Kanamori, Hirotaka Hachiya, and Daisuke Okanohara. Least-squares conditional density estimation. *IEICE Transactions on Information and Systems*, 93(3):583–594, 2010.

[49] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

[50] Gokul Swamy, Sanjiban Choudhury, J Andrew Bagnell, and Zhiwei Steven Wu. A critique of strictly batch imitation learning. *arXiv preprint arXiv:2110.02063*, 2021.

[51] Brian L Trippe and Richard E Turner. Conditional density estimation with bayesian normalising flows. *arXiv preprint arXiv:1802.04908*, 2018.

[52] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[53] Matt P Wand and M Chris Jones. *Kernel smoothing*. CRC press, 1994.

[54] Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *International Conference on Machine Learning*, pages 6536–6544. PMLR, 2019.

[55] Zhiqing Xiao. *Reinforcement Learning: Theory and Python Implementation*. China Machine Press, 2019.

[56] Haoran Xu, Xianyuan Zhan, Honglei Yin, and Huiling Qin. Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *International Conference on Machine Learning*, pages 24725–24742. PMLR, 2022.

[57] Haoran Xu, Xianyuan Zhan, Honglei Yin, and Huiling Qin. Discriminator-weighted offline imitation learning from suboptimal demonstrations. In *International Conference on Machine Learning*, pages 24725–24742. PMLR, 2022.

[58] Sheng Yue, Guanbo Wang, Wei Shao, Zhaofeng Zhang, Sen Lin, Ju Ren, and Junshan Zhang. CLARE: Conservative model-based reward learning for offline inverse reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[59] Sheng Yue, Guanbo Wang, Wei Shao, Zhaofeng Zhang, Sen Lin, Ju Ren, and Junshan Zhang. CLARE: Conservative model-based reward learning for offline inverse reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=5aT4ganOd98.

[60] Yisong Yue and Hoang M. Le. Imitation learning (tutorial). *International Conference on Machine Learning (ICML)*, 2018.

[61] Siliang Zeng, Chenliang Li, Alfredo Garcia, and Mingyi Hong. Understanding expertise through demonstrations: A maximum likelihood framework for offline inverse reinforcement learning. *arXiv preprint arXiv:2302.07457*, 2023.

[62] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, page 1433–1438. AAAI Press, 2008.

Appendix A. Other Related Work

To improve BC, methods such as incorporating online interactions, demonstrator feedback, or leveraging model dynamics and reward sparsity have been explored, but these are impractical with only offline data. Recent work [56] attempts to address this by using supplementary data from non-expert policies, but such data may not always be available. The EDM approach [17] models the expert's state occupancy with an energy-based framework, though it has significant limitations [50].

Efforts to improve IRL introduced methods like *LSTD-μ* [22, 23], a temporal difference technique prone to the same weaknesses as least squares estimators, including sensitivity to feature selection and data distribution. [29] propose *DSFN*, a transition-regularized network that generates policies close to expert behavior, but it unrealistically assumes full knowledge of reward features [3]. [37] introduced *RCAL*, a non-parametric algorithm using boosting to minimize a large margin objective with MDP-aware regularization. [9] propose *AVRIL*, which jointly estimates the posterior of reward and policy, while [13] introduce IQ-Learn, an off-policy IRL method that implicitly learns both via a soft Q-function. However, both face covariate shift issues, leading to poor reward extrapolation in novel environments. *CLARE* [58] addresses this by adding conservatism to reward estimates but assumes access to a diverse dataset, limiting performance when data quality is low [61].

Adversarial Imitation Learning (AIL) methods [16] marked a significant advancement but rely on online interactions, limiting their use in offline settings. [25] proposed *ValueDICE*, which uses distribution matching between expert and imitator policies through a complex optimization procedure. However, it struggles with gradient estimation, introducing bias when using mini-batches [17]. Our algorithm diverges from these approaches, offering a simpler, more efficient solution with promising preliminary results.

Appendix B. Theoretical Guarantees

We state the following technical lemma for convergence in probability of kernel density estimators from i.i.d samples which we used for Theorem 1.

Lemma 2 *[7] Suppose $X_1, X_2, \ldots, X_n$ are i.i.d vectors with probability density g. Let $\hat{g}(\cdot; H)$, as given in Eq. (5), be the kernel density estimator constructed from these samples using kernel K and bandwidth matrix $H = H(n)$. Suppose the following assumptions hold.*

(C1) *Each entry of $\mathcal{H}_g(\cdot)$ be piecewise continuous and square integrable, where $\mathcal{H}_g$ is the $m \times m$ Hessian-matrix of g.*

(C2) *The kernel K, is square integrable, spherically symmetric and with a finite second order moment; this means that $\int_{\mathbb{R}^m} zK(z)dz = 0$ and $\int_{\mathbb{R}^m} zz^T K(z)dz = m_2(K)I_m$ (where $m_2(K)$ is independent of i, for $i \in \{1, 2, \ldots m\}$). Furthermore, $\int_{\mathbb{R}^m} K(z) = 1$.*

(C3) *The bandwidth matrices $H = H(n)$ form a sequence of positive definite, symmetric matrices such that as $n \to \infty$, vec $H(n) \to 0$, i.e. all entries of $H(n)$ approaches 0 and $n^{-1}|H(n)|^{-1/2} \to 0$, where vec is the vectorization operator which acts on a matrix by stacking its columns on top of one another.*

Then, $\hat{g}(\boldsymbol{x}; H)$ converges in probability to $g(\boldsymbol{x})$ for each $\boldsymbol{x}$.

We introduce the kernel functions K_i, $i = 1, 2, 3$, so that using (4), the kernels K_{H_i} for $i = 1, 2, 3$ appearing in (9) can be expressed as

$$K_{H_i}(x) = |H_i|^{-\frac{1}{2}} K_i(H^{-\frac{1}{2}} x), \tag{11}$$

where x is of appropriate dimension. We will make the following assumptions [7]:

(A1) Suppose the buffer B in Algorithm 1 consists of n iid tuples (s, a, s', a') generated according to a probability distribution $P(s, a, s', a') = \mu(s, a)P_{\pi_D}(s', a'|s, a)$, where P_{π_D} is the transition probability density of the induced Markov chain on the state-action space under the demonstration policy π_D (see (7)) and μ is a reference probability measure on (s, a). Further, P has a density function g that is square-integrable and twice differentiable, with all of its second-order partial derivatives bounded, continuous and square integrable. Also assume that the marginals $P(s', s, a)$ and $P(s, a)$ satisfy these properties.

(A2) The kernels K_i for $i \in \{1, 2, 3\}$ in (11) are square integrable, zero-mean, spherically symmetric, and with common finite second-order moment $\int_{\mathbb{R}^{m_i}} zz^T K_i(z)dz = \sigma^2 I_{m_i}$.

(A3) For each kernel K_{H_i} as defined in (4), the bandwidth matrices $H_i(n)$ (where n is the number of tuples in B) form a sequence of positive definite, symmetric matrices such that $H_i(n) \to 0$ and $n^{-1/2}|H_i(n)|^{-1/2} \to 0$ as $n \to \infty$.

With these assumptions, we re-state Theorem 1 as follows:

Theorem 1 *Suppose assumptions (A1)-(A3) are true. Let $\widehat{P}_n$ and $\widehat{T}_n$ be the CKDE estimates constructed using (9) and a buffer B with n tuples. Then, for each (s, a, s', a'), as $n \to \infty$,*

$$\widehat{P}_n(s', a'|s, a) \xrightarrow{P} P_{\pi_D}(s', a'|s, a), \quad \text{and} \quad \widehat{T}_n(s'|s, a) \xrightarrow{P} T(s'|s, a). \tag{12}$$

We now adopt Lemma 2 to give a detailed outline of proof for Theorem 1 under assumptions (A1)-(A3). First of all, we defined the following in the proof:

$$\begin{aligned} \hat{f}(s', s, a) &= \sum_{l=1}^{n} K_{H_3}\big(d_3(s', s'_l)\big) K_{H_2}\big(d_2((s, a), (s_l, a_l))\big), \\ \hat{g}(s, a) &= \sum_{l=1}^{n} K_{H_2}\big(d_2((s, a), (s_l, a_l))\big). \end{aligned} \tag{13}$$

We can then argue as follows:

1. We assume (A1) that $P(s, a, s', a')$ has a density function g that is square-integrable and twice differentiable, with all of its second-order partial derivatives bounded, continuous and square integrable and so does its marginals $P(s', s, a)$ and $P(s, a)$. This leads to the satisfaction of condition (C1).

2. From assumption (A2), $\int_{\mathbb{R}^{m_i}} zK_i(z)dz = 0$ for $i = \{2, 3\}$, where z_i is a vector of size m_i. Partition the vector z as $z = [z_3, z_2]$ and let $m = m_2 + m_3$ and $K(z) = K_3(z_3)K_2(z_2)$. Then

for $t \leq m_3$,

$$\begin{aligned}\int_{\mathbb{R}^m} z_t K(z) dz &= \int_{\mathbb{R}^m} z_t K_3(z_3) K_2(z_2) dz \\ &= \int_{\mathbb{R}^{m_2}} K_2(z_2) dz_2 \int_{\mathbb{R}^{m_3}} z_t K_3(z_3) dz_3 \\ &= \int_{\mathbb{R}^{m_3}} z_t K_3(z_3) dz_3 = 0,\end{aligned} \tag{14}$$

which follows from (A2). This can be shown for any $t \in \{1, 2, \ldots, m\}$. Hence, $\int_{\mathbb{R}^m} zK(z)dz = 0$ is satisfied corresponding to condition (C2).

Now,

$$\begin{aligned}&\int_{\mathbb{R}^m} zz^T K(z) dz \\ &\quad= \int_{\mathbb{R}^m} \begin{bmatrix} z_3 z_3^T & z_3 z_2^T \\ z_2 z_3^T & z_2 z_2^T \end{bmatrix} K_3(z_3) K_2(z_2) dz_3 dz_2 \\ &\quad= \sigma^2 \begin{bmatrix} I_{m_3} & 0 \\ 0 & I_{m_2} \end{bmatrix} = \sigma^2 I_m.\end{aligned}$$

Hence, $K(z) = K_3(z_3)K_2(z_2)$ satisfies condition (C2).

3. Consider $H(n)$ to be a block diagonal matrix with $H_3(n)$ and $H_2(n)$ as the two block diagonal entries with $H_3(n)$ and $H_2(n)$ satisfying assumption (A3). Then the matrices $H(n)$ form a sequence of positive definite, symmetric matrices. Using (5) with this H, the kernel estimate for $P(s', s, a)$ takes the product kernel form as seen for $\hat{f}(\cdot)$ in (13). Now, $|H(n)| = |H_3(n)||H_2(n)|$, this implies that as $n \to \infty$, $n^{-1}|H(n)|^{-1/2} \to 0$ because $n^{-1/2}|H_i(n)|^{-1/2} \to 0$ for $i = \{2, 3\}$. Also, vec $H(n) \to 0$ as vec $H_i(n) \to 0$ for $i = \{2, 3\}$. Therefore, condition (C3) is satisfied.

Having satisfied conditions (C1)-(C3), we may apply the argument found in Sections 2.6-2.9 of [7] and conclude that

$$\hat{f}(s', s, a) \xrightarrow{P} P(s', s, a),$$
$$\hat{g}(s, a) \xrightarrow{P} P(s, a).$$

Finally, it follows from the Continuous Mapping Theorem [33] that taking the ratio of $\hat{f}$ and $\hat{g}$ produces a consistent estimator of

$$\frac{P(s', s, a)}{P(s, a)} = T(s'|s, a),$$

i.e.,

$$\hat{T}_n(s'|s, a) = \frac{\hat{f}(s', s, a)}{\hat{g}(s, a)} \xrightarrow{P} T(s'|s, a).$$

A similar argument can be used to establish the asymptotic convergence in probability for the CKDE of $P_{\pi_D}(s', a'|s, a)$.

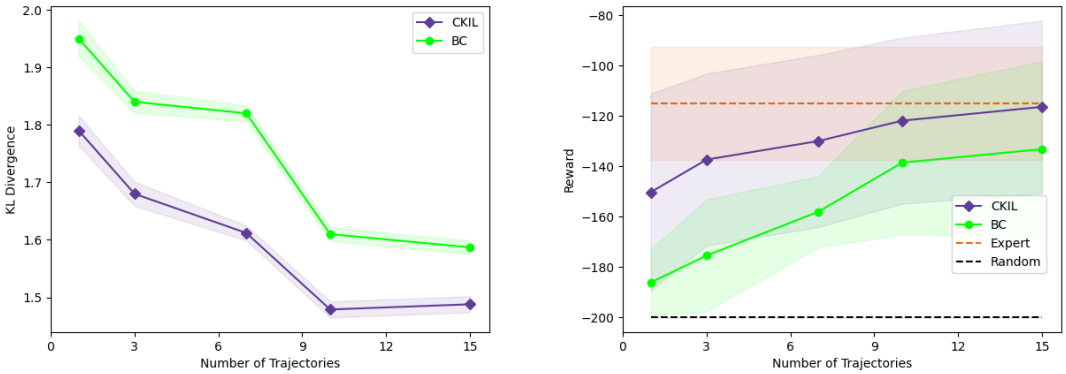


Figure 2: Plots for CKIL agent in a discretized MountainCar environment for a varying number of trajectories (a) Empirical KL divergence (lower values indicate better performance) (b) Average rewards attained (higher values indicate better performance).

Appendix C. Discrete Spaces

We discuss kernel density estimation methods for estimating the two conditional densities P_{π_D} and T for the discrete state and action space setting, where the estimation process is intuitive. When both the state and action spaces are discrete, the estimates $\hat{P}$ and $\hat{T}$ can be calculated as:

$$\hat{T}(s'|s,a) := \frac{\eta(s,a,s')}{\eta(s,a)}, \quad \text{and} \quad \hat{P}(s',a'|s,a) := \frac{\eta(s,a,s',a')}{\eta(s,a)} \tag{15}$$

, where η denotes the counting measure, i.e., the number of times a given tuple or sequence appears in the dataset D. If the counting measure in the denominator is zero, we will consider the conditional density to be uniform.

Results. The policy representation and associated hyperparameters are detailed in Appendices E and F respectively. While we are focused primarily on continuous state-space settings requiring estimation of conditional densities, we illustrate the effectiveness of our general approach in a discrete state-space problem. Our comparison evaluates the performance of CKIL against Behavior Cloning and an agent that selects random actions at each visited state. We consider the discretized MountainCar problem [36] as an example of a discrete state and action space environment. To that end, we uniformly discretized the original continuous 2-dimensional state space of the MountainCar environment into a grid configuration measuring 15 by 15. Subsequently, we estimate $\hat{P}$ and $\hat{T}$ using equation (15). The dataset D is generated using an ϵ-perturbation of the policy outlined in [55], with $\epsilon = 0.05$. We begin with one trajectory in D and increase to 15 trajectories, with observations summarized in Figure 2. The KL-divergence plot indicates increasing alignment between agent and expert policies with growing training dataset sizes. As the amount of data increases, the CKIL agent's performance improves, achieving expert-level proficiency with 15 trajectories and consistently outperforming Behavioral Cloning across various scenarios.

Appendix D. Environments and Data Collection

The environments representing a wide spectrum of complexities commonly encountered in reinforcement learning sourced from OpenAI Gym [6] include the MountainCar environment [36], CartPole [4], Acrobot [49], and LunarLander [24]. To generate demonstration datasets D, we leverage pre-trained and hyperparameter-optimized agents available in the RL Baselines Zoo [40]. Specifically, we employ a PPO agent for LunarLander-v2, a DQN agent for CartPole-v1, and an A2C agent for Acrobot-v1.

Appendix E. Policy implementation and choice of Kernels

Implementation. The policy π_θ in (6) is embodied by a neural network (NN) architecture. This NN comprises two hidden layers featuring the Rectified Linear Unit (ReLU) activation function. The final layer employs a softmax function to produce a probability distribution over actions when given a state as an input. To facilitate comparison, all benchmarks adopt a common neural network architecture consisting of two hidden layers comprising 64 units each, with Exponential Linear Unit (ELU) activation functions, unless otherwise stated in Appendix G. Training is carried out using the Adam optimizer [21] with individually tuned learning rates. The implementation details, including hyperparameters of CKIL and benchmark algorithms, can be found in Appendices F, G, respectively.

Choice of Kernel. We use the Gaussian kernel due to its ability to uniformly approximate any continuous target function on a compact subset [34] which is particularly helpful in density estimation. A standard multivariate Gaussian kernel function for m dimensions is given as:

$$K(x) := (2\pi)^{-\frac{m}{2}} \exp\left(-\frac{x^T x}{2}\right).$$

We consider a Euclidean distance metric for d_1, d_2, and d_3 and utilize a diagonal bandwidth matrix with the same values across its diagonal elements. These matrices can then be denoted as $H_i = h_i I_{m_i}$, where m_i is the corresponding appropriate dimension for $i = 1, 2, 3$. Each h_i was tuned slightly but the conclusions of our experimental results are not very sensitive to this choice. We want to emphasize that in prior research [1, 32, 44, 47], approaches for systematically selecting bandwidth parameters, which should decrease as the dataset grows, have been developed. These methods can be applied to more intricate problems where manual tuning is impractical. Specific values of h_i employed for various experiments are detailed in Appendix F.

Appendix F. Hyperparameters for CKIL

We report the hyperparameters used for CKIL. For discrete case, since number of states and number of actions are both finite, instead of using a parameterized policy, we defined a policy for each states and learnt it via optimizing the objective in (8). We used a learning rate of 0.5 for the same. We set $\lambda = 0.001$.

For continuous case, we adopted a neural network architecture for learning the policy. This neural network consisted of 2 hidden layers with 64 nodes followed by 32 nodes. Final layer consisted of a Softmax function to output the policy when a state was provided as an input. We used Adam optimizer and a learning rate of 0.01. We use the same value for bandwidth parameters h_1 and h_2.

	Environment		
Trajectories (τ)	Acrobot-v1	CartPole-v1	LunarLander-v2
$\tau = 1$	0.1	0.01	0.009
$\tau = 3$	0.05	0.005	0.005
$\tau = 7$	0.01	0.001	0.0005
$\tau = 10$	0.008	0.0008	0.00008
$\tau = 15$	0.005	0.0001	0.00005

Table 1: h_1 values used for CKIL on different environments for varying number of trajectories during training.

Let m_1 be the dimension of a state s, we then take the value of h_3 as $h_3 = h_1^{\frac{m_1+1}{m_1}}$. We report the values of bandwidth parameter h_1 in Table 1. We set $\lambda = 0.001$.

Appendix G. Benchmark Algorithms and Hyperparameters

Baseline Algorithms. We compare the performance of our CKIL algorithm (Algorithm 1), with a range of offline IRL/IL/AIL baselines, including several recent state-of-the-art algorithms. This comprehensive assessment covers a spectrum of methodologies, including the inherently offline Behavioral Cloning (BC); ValueDICE (VDICE), a sample-efficient AIL approach designed for offline scenarios by removing replay regularization; reward-regularized classification (RCAL), a large margin classification approach, which introduces a sparsity-based penalty on inferred rewards to exploit dynamics information; Energy-based Distribution Matching (EDM), an offline imitation learning algorithm that captures the expert's state occupancy patterns through explicit training of an energy-based model; AVRIL, a recent model-free offline IRL technique employing a variational approach to simultaneously learn an approximate posterior distribution over rewards and policies; and Deep Successor Feature Network (DSFN), an offline adaptation of the max-margin IRL algorithm that transcends linear approaches by introducing a deep network architecture and employing least-squares temporal-difference learning to produce both reward and policy outputs. Furthermore, we compare against IQ-Learn, a state-of-the-art model-free offline IRL algorithm.

Implementation details. In the case of VDICE, we used the open-sourced code provided at [26]. It is worth noting that, for VDICE, offline learning is achieved by configuring the "replay regularization" coefficient to zero. Our execution of EDM leveraged the source code accessible at [18]. It is essential to highlight that the contrast between BC and EDM predominantly stems from the introduction of L_ρ, an occupancy loss defined in the EDM work, while deriving the RCAL loss is a straightforward process involving the inversion of the Bellman equation. As for AVRIL and DSFN, the applicable source codes are accessible at [8], [28] respectively. Similarly, for IQ-Learn, we utilised the source code available at [14].

We consider the hyperparameters associated with various benchmarks, as outlined in [17]. To ensure comprehensiveness, we present them herein. When feasible, the policies trained by all imitation algorithms utilize an identical policy network structure, comprising of two fully connected hidden layers, each containing 64 units with ELU activation function. Across all environments, we adopt the Adam optimizer with a batch size of 64, conducting $10,000$ iterations, and employing

a learning rate of $1e-3$. With the exception of the explicit standardization of policy networks among imitation algorithms, all comparators are realized using the unaltered publicly accessible source code. When relevant, we employ the optimal hyperparameters as indicated in the original implementations.

G.1. VDICE

We employ the publicly accessible source code from https://github.com/google-research/google-research/tree/master/value_dice. To accommodate discrete action spaces, we incorporate a Gumbel-softmax parameterization for the final layer of the actor network. Both the actor and discriminator architecture encompass two fully connected hidden layers, each composed of 64 units activated by ReLU functions. Consistent with the original framework, the output is merged with the action and propagated through two additional hidden layers, each containing 64 units. In addition, we set the "replay regularization" coefficient at zero for strict batch learning. Furthermore, the actor network is subjected to "orthogonal regularization" with a coefficient of 1e-4. The actor network's learning rate is set at 1e-5, while the discriminator operates with a learning rate of 1e-3.

G.2. RCAL

This introduces an expansion of the policy loss by incorporating an extra sparsity-driven loss concerning the inferred rewards $\hat{R}(s,a)$, defined as $f_\theta(s)[a] - \gamma softmax_{a'} f_\theta(s')[a']$, acquired through the inversion of the Bellman equation. The policy network employed is the fully-connected type detailed previously. The coefficient for sparsity-based regularization is designated as 1e-2.

G.3. EDM

We utilize the code accessible at https://github.com/vanderschaarlab/mlforhealthlabpub/tree/main/alg/edm. Particularly for EDM, the hyperparameters for joint Energy-Based Model (EBM) training are adopted from https://github.com/wgrathwohl/JEM. These parameters include a noise coefficient of $\sigma = 0.01$, a buffer size of $\kappa = 10000$, a length of $\iota = 20$, and a reinitialization value of $\delta = 0.05$. These predefined configurations align effectively with the SGLD (Stochastic Gradient Langevin Dynamics) step size of $\alpha = 0.01$.

G.4. BC

The sole distinction between Behavior Cloning (BC) and EDM lies in the inclusion of L_ρ, which is omitted in the implementation of BC. The policy network remains consistent with the description provided earlier.

G.5. AVRIL

We use the code available at https://github.com/XanderJC/scalable-birl. The policy network remains consistent with the description provided earlier. γ, used while computing the TD error, equals 1. We used the default parameters provided in their GitHub repository.

G.6. DSFN

We adopt the source code accessible at https://github.com/dtak/batch-apprenticeship-learning. We utilize a "warm-start" policy network consisting of two shared layers with dimensions 128 and 64, employing tanh activation. The hidden layer with a size of 64 serves as the feature map within the IRL algorithm. Each multitask head within the warm-start policy network features a hidden layer comprising 128 units and is activated by tanh. The Deep Q-Network (DQN), utilized for learning the optimal policy based on a set of reward weights, comprises two fully-connected layers, each containing 64 units. Similarly, the DSFN, employed for estimating feature expectations, comprises two hidden fully-connected layers, each containing 64 units. Across all environments, the warm-start policy network undergoes training for $50,000$ steps, employing the Adam optimizer with a learning rate of 3e-4 and a batch size of 64. The DQN network is trained for $30,000$ steps, using a learning rate of 3e-4 and a batch size of 64 (with the Adam optimizer). Lastly, the DSFN network is trained for $50,000$ iterations, utilizing a learning rate of 3e-4 and a batch size of 32 (with the Adam optimizer).

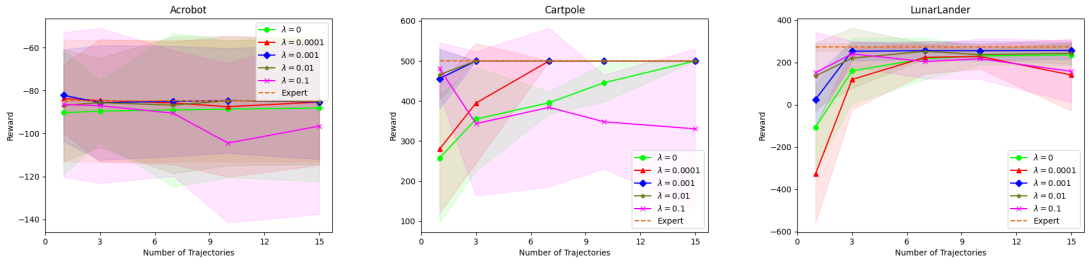
G.7. IQ-LEARN

We use the code available at https://github.com/Div99/IQ-Learn. The policy network remains consistent with the description provided earlier. As highlighted in their implementation, we use a batch size of 32 and Q-network learning rate of 1e-4 with entropy coefficient of 0.01.

All experiments involving CKIL and benchmarks were conducted on an M2 MacBook Air machine equipped with 16 GB of RAM and a 512 GB SSD.

Appendix H. Ablation Study

In this section, we present the evaluation of CKIL's performance under various ablations.



(*a*) Acrobot (*b*) CartPole (*c*) LunarLander

Figure 3: Average rewards achieved by CKIL agent when trained using different λ values during real-time deployment plotted against the number of trajectories included in demonstration dataset D (higher values indicate better performance).

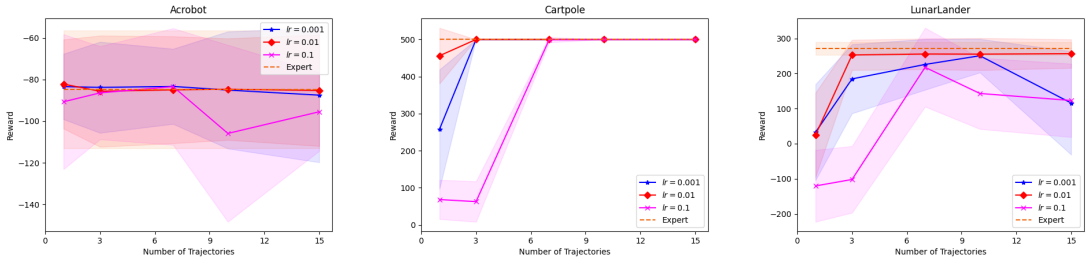
H.1. Varying λ for entropy regularization

Figure 3 shows the average cumulative reward on the considered gym environments as a function of different λ values. We observe that when the data is very scarce (eg. 1 trajectory), having a higher

λ value helps as we are less certain about which action is best to take in a given state. Conversely, we observe that having a higher λ performs poorly in comparison to lower λ values with increased data.

H.2. Varying learning rate

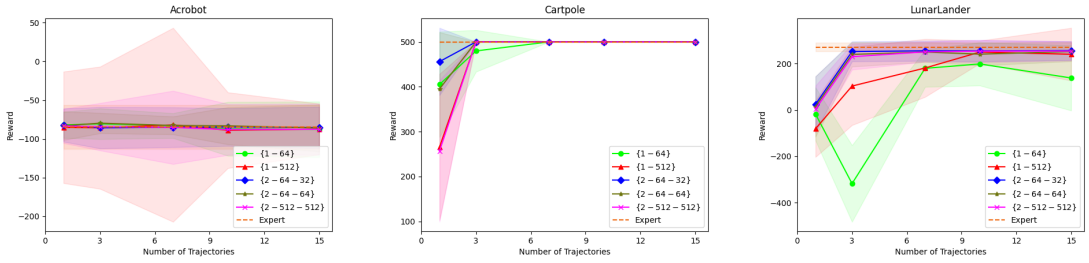
Figure 4 shows the average cumulative reward on the considered gym environments as a function of different learning rate lr values. We observe that the learning rate of 0.01 does well across tasks where the variance in performance is low across different episodes for any given environment along with a similar or better mean performances than other learning rate values.



(*a*) Acrobot (*b*) CartPole (*c*) LunarLander

Figure 4: Average rewards achieved by CKIL agent when trained using different learning rates during real-time deployment plotted against the number of trajectories included in demonstration dataset D (higher values indicate better performance).

H.3. Varying Neural Network Size



(*a*) Acrobot (*b*) CartPole (*c*) LunarLander

Figure 5: Average rewards achieved by CKIL agent when trained using different Neural Network Architectures during real-time deployment plotted against the number of trajectories included in demonstration dataset D (higher values indicate better performance).

Figure 5 shows the average cumulative reward on the considered gym environments as a function of different Neural Network size values. The legend is in the form $\{a - b\}$ or $\{a - b - c\}$ where a indicates the number of hidden layers, followed by number of hidden nodes in each layer, which

	Algorithm		
Trajectories (τ)	EDM	IQ-Learn	CKIL
$\tau = 1$	-406.37 ± 340.03	5.12 ± 190.12	-0.81 ± 169.56
$\tau = 3$	-299.21 ± 184.60	159.726 ± 120.18	202.73 ± 93.03
$\tau = 7$	-240.38 ± 202.53	205.96 ± 99.68	239.78 ± 62.59
$\tau = 10$	-38.36 ± 130.54	232.28 ± 87.66	245.41 ± 57.85
$\tau = 15$	89.32 ± 101.45	242.32 ± 77.59	247.4 ± 58.82

Table 2: Performance of EDM, IQ-Learn, and CKIL in the presence of initial distribution shift (higher values indicate better performance)

are denoted by b and c. For example, $\{1 - 64\}$ represents a neural network with one hidden layer containing 64 hidden nodes in it. Similarly, $\{2 - 64 - 32\}$ represents a neural network with two hidden layers, with 64 hidden nodes in the first layer followed by 32 hidden nodes in the second hidden layer.

For all the environments, we observe that having 2 hidden layers provides good performance, and the performance is not sensitive to the number of hidden nodes in the two hidden layers. Furthermore, with one hidden layer, spread is very high in some cases. Therefore, we selected a neural network with 2 hidden layers, containing 64 nodes in the first hidden layer and 32 nodes in the second hidden layer for our gym experiments.

Appendix I. Discussion on Distribution Shift

In this work, we have discussed the task of imitation learning in a strictly batch setting. Specifically, we assumed that only expert data was available, with no possibility for further interaction with the environment. Another line of research in imitation learning aims to incentivize the imitating policy to remain within the distribution of states encountered in expert demonstrations. This research typically follows two approaches. The first approach assumes access to additional data from a behavioral policy (which may be sub-optimal) along with the expert data. This additional data is used to provide coverage, as expert data is generally narrow. Examples of this approach include methods like CLARE [58] and MILO [10]. The second approach involves techniques such as assigning a unit reward to all demonstrated actions in demonstrated states and zero otherwise [41], such as random expert distillation [54]. Generally, these methods follow a "two-step" formula: first, a surrogate reward function is derived or defined; second, this reward function is optimized through environment interactions, making these techniques inherently online, rendering it inapplicable in our strictly batch setting [31].

Nevertheless, we investigated the effects of an initial distribution shift in the LunarLander-v2 environment, drawing inspiration from the approach in [13]. Typically, the agent starts in a small area at the center-top of the screen. However, we modified the environment so that the agent begins near the top-left corner instead. Using expert data from the standard, unmodified environment, we aimed to determine if the agent could still successfully learn to land the lunar module despite the shift in its initial conditions during testing. For comparisons, we consider EDM and IQ-Learn algorithms as these methods don't rely on additional data or further interactions with the environment, thus utilizing the same setup as ours. The findings are reported in Table 2. As anticipated,

all algorithms perform poorly when the available data is very scarce. However, as the amount of data gradually increases, the CKIL agent demonstrates the ability to effectively land the lunar lander despite the initial distribution shift, even with limited training data (approximately 10 trajectories). Additionally, CKIL outperforms baseline algorithms like EDM and IQ-Learn under these conditions. Our experimental results suggest that our algorithm handles the distribution shift problem more effectively than the other baseline algorithms in the same setup.