

# IBCB: Efficient Inverse Batched Contextual Bandit for Behavioral Evolution History

Yi Xu, Weiran Shen, Jun Xu, *Member, IEEE*, Xiao Zhang,  
and Ji-Rong Wen, *Senior Member, IEEE*

**Abstract**—Traditional imitation learning focuses on modeling the behavioral mechanisms of experts, which requires a large amount of interaction history generated by some fixed expert. However, in many streaming applications, such as streaming recommender systems, online decision-makers typically engage in online learning during the decision-making process, meaning that the interaction history generated by online decision-makers includes their behavioral evolution from *novice expert* to *experienced expert*. This poses a new challenge for existing imitation learning approaches that can only utilize data from experienced experts. To address this issue, this paper proposes an *inverse batched contextual bandit* (IBCB) framework that can efficiently perform estimations of environment reward parameters and learned policy based on the expert’s behavioral evolution history. Specifically, IBCB formulates the inverse problem into a simple quadratic programming problem by utilizing the behavioral evolution history of the batched contextual bandit with inaccessible rewards, and it can be extended to fairness-aware expert limitation. We demonstrate that IBCB is a unified framework for both deterministic and randomized bandit policies. The experimental results indicate that IBCB outperforms several existing imitation learning algorithms on synthetic and real-world data and significantly reduces running time. Additionally, empirical analyses reveal that IBCB exhibits better imitation ability for fairness-aware experts, out-of-distribution generalization and is highly effective in learning the bandit policy from the interaction history of novice experts. The code is publicly available.

**Index Terms**—batched contextual bandit, imitation learning, behavioral evolution, fairness-aware expert, quadratic programming



## 1 INTRODUCTION

TRADITIONAL imitation learning (IL) focuses on learning the decision-making policy of an experienced and fixed expert by using its historical behaviors. IL utilizes experts’ demonstrations for policy fitting and aims to extract knowledge from experts’ demonstrations to replicate their behaviors or environments. It can be divided into two main categories: Behavior Cloning (BC) [1] and Inverse Reinforcement Learning (IRL) [2]. BC approaches aim to learn a policy that directly maps states to actions, while IRL approaches focus on recovering the reward parameters from demonstrations to recover the experts’ policy [3].

A major challenge in imitation learning (IL) is the substantial requirement of expert demonstration data [4], [5]. There is a common assumption that the experts are always experienced. However, in real-world streaming scenarios, the behavior policy of the expert constantly evolves over time, transitioning from a novice expert to an experienced one. Throughout this process, we accumulate a significant amount of expert’s *behavioral evolution history*. Taking streaming recommender systems as an example [6], [7], the agent in streaming recommendation needs to make a trade-off between exploitation and exploration during the recommendation process, and continuously and incrementally update its recommendation policy [8], [9], [10].

In the behavioral evolution history, there exist a significant amount of contradictory data where an expert may

take different actions when facing the same context at different time periods. This contradicts the assumption made by traditional imitation learning (IL) approaches regarding the consistency of expert behavior data. Meanwhile, recent works like [11], [12] focus on maintaining fairness for bandit algorithms, which introduced a distribution-based random action selection, rather than simply selecting the best action, adding the difficulty to recover expert’s parameters. Therefore, directly applying existing IL methods to recover the behavior policy from the expert’s behavioral evolution history is not reasonable. For example, behavior cloning (BC) approaches are highly sensitive to distribution drift and contradictory data, and inconsistencies in the expert’s evolution history can lead to a sharp decline in the final recovery performance. Furthermore, some IL approaches rely on large-scale iterations of Metropolis-Hastings sampling to achieve accurate estimation in universal scenarios [13], [14]. However, these approaches overlook the exploitation-exploration mechanism in expert interactions and demonstrate low sample efficiency when dealing with large-scale interaction data. While recent work by [15], [16] focused on stochastic contextual bandits, it cannot be adapted to the batched bandit setting as it assumes the expert updates the policy in a fully-online manner. Additionally, existing inverse bandit approaches typically employ sampling techniques to estimate the likelihood, resulting in low training efficiency. There is an urgent need to develop inverse bandit methods specifically designed for the more general batched bandit setting and to devise novel techniques for recovering parameters tailored for bandit policies.

In this paper, we focus on the more general batched contextual bandit (BCB) setting, and propose inverse batched

- Corresponding author: Xiao Zhang
- Y. Xu, W. Shen, J. Xu, X. Zhang, and J. Wen are with the Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China.  
E-mail: {yixu00, shenweiran, junxu, zhangx89, jrwen}@ruc.edu.cn

contextual bandit (ICB) to overcome several issues mentioned above. To learn from the evolving behaviors made by online decision-making experts with BCB policies, we design ICB with a unified framework for both deterministic [17] and randomized [18] bandit policies. Under this unified framework, we take linear constraints with expectation relaxation in BCB’s exploitation-exploration pairs and batched updating policy into consideration. We formulate the inverse problem of ICB into a simple quadratic programming problem without notifying expert’s behavior rewards. We take the assumption of BCB setting, rather than simply cloning behaviors with no assumption or blindly using large-scale samplings. Combining all these improvements together accelerates the train speed of ICB and insures the robustness of ICB for fairness-aware scenarios, out-of-distribution and contradictory data. Due to that original BCB learns the reward parameter through online learning, ICB can capture both expert’s policy parameters and reward parameters at the same time.

We summarize our major contributions: (1) We define a new inverse bandit problem with behavioral evolution history in the BCB setting; (2) We introduce a unified framework called ICB, designed for both deterministic and randomized BCB policies, which can efficiently learn from the interaction history data of novice experts without requiring experts’ feedbacks; (3) ICB is capable of simultaneously learning the parameters of the expert policy and the reward feedback, which outperforms various existing baseline IL approaches in scenarios such as fairness-aware experts, out-of-distribution data and contradictory data, with notable improvements in training speed.

## 2 RELATED WORKS

**Contextual Bandits** have been extensively utilized for solving sequential decision-making problems in online learning [19], [20], [21]. Recently, there has been increased research focus on a more general setting called batched contextual bandit (BCB) [17], [22], [23], where actions within the same batch share fixed policy parameters. In this regard, [17] proposed a UCB-like update policy for BCB, which is applicable to both random and adversarial contextual data.

**Imitation Learning (IL)** is typically used to learn the expert’s action selection policy or estimate the reward parameter from the expert’s behavior [3], [4], [5], [24]. IL can be divided into two categories. The first category is Behavior Cloning (BC) [1], which directly constructs a direct mapping from state to action [25], [26], [27]. The second category is the inverse reinforcement learning (IRL) method [2], which attempts to recover environment’s reward parameter from the expert’s behavior, that is, imitating the reward parameters. Existing works mainly study IRL based on Bayesian [13], [14], [28], maximum entropy (max-ent) [29], [30], [31], and generative [32], [33] approaches. Traditional IL approaches mostly assume that the expert’s behavior is optimal at every step [5], [29], [34], but ignore that early interactions are likely to be the evolutionary data of novice experts learned to become experienced experts.

**Offline Reinforcement Learning (Offline RL)** has been widely used to improve the model’s ability to generalize to out-of-distribution data, i.e., extrapolation error [35], [36].

There are two main research focuses in existing offline RL. One is to restrict the policy during training to avoid out-of-distribution generalization issues, which can be further subdivided into explicitly restricted policies [37], [38], [39], [40] and implicitly restricted policies [41], [42]. The other is to allow the learning model to adaptively estimate the uncertainty of the environment or data [43], [44]. Unlike Offline RL, which requires knowledge of the reward feedback, the proposed ICB can learn about reward and expert policy’s parameters from the history of expert behavioral evolution without requiring reward feedback information from the environment.

## 3 PROBLEM FORMULATION

Let  $[k] = \{1, 2, \dots, k\}$ ,  $\mathcal{S} \subseteq \mathbb{R}^d$  be the context space of dimension  $d$ ,  $[A; B] = [A^\top, B^\top]^\top$ ,  $\|x\|_2$  denotes the  $l_2$ -norm of a vector  $x$ .  $\langle \cdot, \cdot \rangle$  denotes inner product of two vectors with same dimension.

**Batched Contextual Bandit (BCB) Setting.** Following the setups in linear contextual bandit literature [18], [19], [21], for any context  $s_i \in \mathcal{S} \subseteq \mathbb{R}^d$ , we assume that the expectation of the observed reward  $R_i$  from environment is determined by unknown *true reward parameters*  $\theta^* \in \mathbb{R}^d$ :  $\mathbb{E}[R_i | s_i] = \langle \theta^*, s_i \rangle$ . In this paper, our focus is on the generalized version of the traditional contextual bandit called the *batched contextual bandit* (BCB) setting, which has gained considerable attention in the field of bandit theory and applications [6], [17]. For an expert policy in BCB setting, we define expert’s decision-making progress is partitioned into  $N$  episodes, and in each episode, expert consists of two phases: (1) the *online decision-making* chooses the action (i.e., candidate context, and also *behavior*) for execution from each step’s candidate context set following the updated and fixed expert policy  $f$  for  $B$  steps ( $B$  is also called *batch size*), and finally stores all context set & action pairs and the observed rewards of the executed actions (contexts) into the data buffer  $\mathcal{D}$  (*expert’s behavioral evolution history with rewards*); (2) the *policy updating* approximates the optimal policy based on the received actions and rewards in  $\mathcal{D}$ . Note that for expert policy itself to evolve, expert can observe the reward of executed actions. Also, data buffer stores the indexes of the executed actions rather than actions themselves since each step’s context set contains all actions, including the executed actions. This saves the storage space for practical applications.

**Inverse Bandit Problem.** Then, we introduce the formal definition of *inverse bandit* problem with behavioral evolution history in BCB setting as follows. Figure 1 shows the information that inverse batched contextual bandit (ICB) can observe, where reward of executed actions (*behaviors*) is inaccessible. If expert do not update its parameter after each episode, this problem setting will degenerate to the traditional imitation learning problem.

**Definition 1** (Inverse Bandit Problem with Behavioral Evolution History in BCB Setting). *Denote that  $s_I, I \in \mathcal{I}$ , is a candidate context,  $\mathcal{I} = \{I_j\}_{j \in [M]}$  is the action’s index space containing  $M$  action indexes,  $\mathcal{S} \subseteq \mathbb{R}^d$  is the context space whose dimension is  $d$ ,  $\mathcal{S}_{n,b} = \{s_I\}_{I \in \mathcal{I}} \subseteq \mathcal{S}$  is the candidate context set for expert at step  $b$  in the  $n$ -th episode,  $I_{n,b} \in \mathcal{I}$  is the*

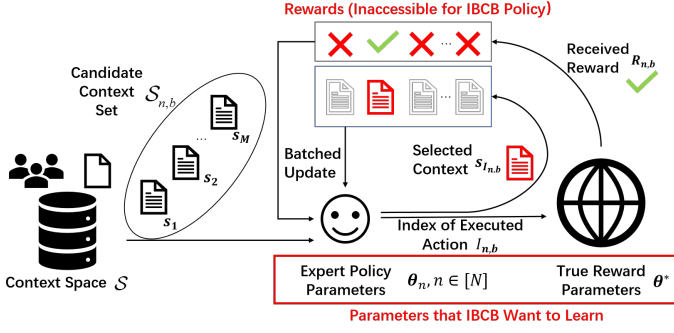


Fig. 1. Inverse bandit problem with behavioral evolution history in BCB setting at step  $b$  in  $n$ -th episode, where the rewards are inaccessible and our goal is to infer and estimate the policy parameters as well as the reward parameters.

index of the expert policy's executed action (behavior) at step  $b$  in the  $n$ -th episode. Observe the behavioral evolution history generated by an expert with the format of  $\mathcal{D}^U = \bigcup_{n \in [N]} \mathcal{D}_n^U$ , where  $\mathcal{D}_n^U = \{(S_{n,b}, I_{n,b})\}_{b \in [B]}$ ,  $\forall n \in [N]$ , i.e., the reward feedbacks cannot be observed,  $B$  is the batch size. We assume expert policy gets its parameters as  $\{\theta_n\}_{n \in [N]}$  through its online update method, and assume  $\theta_N$  is the closest estimation for reward parameter generated by expert policy in the evolution. We want to learn both expert policy parameter  $\theta_N$  and reward parameter  $\theta^*$  accurately through  $\mathcal{D}^U$ .

## 4 IBCB: THE PROPOSED APPROACH

In this section, we propose Inverse Batched Contextual Bandit approach named IBCB which can efficiently learn from expert's behavioral evolution history.

### 4.1 Policy Updating in BCB Setting

At the end of the  $n$ -th episode in the online learning period, the context vectors (corresponding to the executed actions) and their rewards are observed, and are stored into a context matrix  $S_n \in \mathbb{R}^{B \times d}$  and a reward vector  $R_n \in \mathbb{R}^B$ , respectively. In particular, each row index of  $S_n$  and  $R_n$  corresponds to the index of the step in which the context-reward pair was received. We introduce the updating process of the reward parameter vector  $\theta_n$  in BCB. We first concatenate the context and reward matrices from the previous episodes:

$$\begin{aligned} L_n &= [S_0; S_1; \dots; S_n] \in \mathbb{R}^{nB \times d}, \\ T_n &= [R_0; R_1; \dots; R_n] \in \mathbb{R}^{nB}. \end{aligned}$$

Then, the updated parameter vector  $\theta_{n+1}$  can be obtained by solving the following ridge regression: for  $n = 0, 1, \dots, N-1$ ,

$$\theta_{n+1} = \arg \min_{\theta \in \mathbb{R}^d} \|L_n \theta - T_n\|_2^2 + \lambda \|\theta\|_2^2, \quad (1)$$

where  $\lambda > 0$  is the regularization parameter. The closed least squares solution of Eq.(1) is used for estimating  $\theta_{n+1}$ :

$$\theta_{n+1} = (\Psi_{n+1})^{-1} b_{n+1}, \quad (2)$$

where  $\Phi_{n+1} = \Phi_n + S_n^\top S_n$ ,  $b_{n+1} = b_n + S_n^\top R_n$ ,  $\Psi_{n+1} = \lambda I_d + \Phi_{n+1}$ .

### Algorithm 1 Batched Policy Updating in the $n$ -th episode in BCB Setting

INPUT: Policy  $f_n$ , data buffer  $\mathcal{D}_n$ ,  $\lambda \geq 0$ ,  $\alpha \geq 0$ ,  $\Phi_1 = O_d$ ,  $b_1 = 0$ , batch size  $B$ , context space  $\mathcal{S}$ , action space  $\mathcal{I}$

OUTPUT: Updated policy  $f_{n+1}$

- 1: Store the selected context vectors and the observed rewards into  $S_n \in \mathbb{R}^{B \times d}$  and  $R_n \in \mathbb{R}^B$ , respectively
- 2:  $\Phi_{n+1} \leftarrow \Phi_n + S_n^\top S_n$ ,  $b_{n+1} \leftarrow b_n + S_n^\top R_n$ ,  $\Psi_{n+1} \leftarrow \lambda I_d + \Phi_{n+1}$ ,  $\theta_{n+1} \leftarrow (\Psi_{n+1})^{-1} b_{n+1}$
- 3: // Batched UCB Policy (Step 5)
- 4:  $f_{n+1}$  selects the context as  $I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \theta_{n+1}, s_I \rangle + \alpha [s_I^\top (\Psi_{n+1})^{-1} s_I]^{\frac{1}{2}}$ , where  $s_I \in S_{n,b}$
- 5: // Batched Thompson Sampling Policy (Step 7 and 8)
- 6: Draw  $\tilde{\theta}_{n+1}$  from  $\mathcal{N}(\theta_{n+1}, \alpha^2 (\Psi_{n+1})^{-1})$
- 7:  $f_{n+1}$  selects the context as  $I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \tilde{\theta}_{n+1}, s_I \rangle$ , where  $s_I \in S_{n,b}$
- 8: **Return**  $\theta_{n+1}$ ,  $(\Psi_{n+1})^{-1}$

Finally, as shown in Algorithm 1, the batched UCB policy updates the policy parameters and then selects the context  $s_{I_{n,b}}$  by applying the policy  $f_{n+1}$  according to the following rule:  $I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \theta_{n+1}, s_I \rangle + \alpha [s_I^\top (\Psi_{n+1})^{-1} s_I]^{\frac{1}{2}}$ , where  $s_I \in S_{n,b}$ . On the other hand, the batched Thompson sampling policy is a randomized policy that samples the policy parameters from a Gaussian distribution. However, as demonstrated in Theorem 1, we can prove that the batched Thompson sampling policy can also be expressed in the same regularized form as the UCB policy. Detailed proof can be found in Section 4.1.1.

**Theorem 1.** For batched Thompson sampling policy in batched policy updating (i.e., step 7 and 8 in Algorithm 1), we can obtain the reparameterized result of  $f_{n+1}$  as:

$$I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \theta_{n+1}, s_I \rangle + \alpha [s_I^\top \Psi_{n+1}^{-1} s_I]^{\frac{1}{2}} z \quad (3)$$

where  $z$  is a Gaussian random number drawn from  $\mathcal{N}(0, 1)$ .

#### 4.1.1 Proof of Theorem 1

*Proof.* From Algorithm 1 we get the original form of SBTS's parameter:

$$\tilde{\theta}_{n+1} \sim \mathcal{N}(\theta_{n+1}, \alpha^2 (\Psi_{n+1})^{-1}) \quad (4)$$

Since  $\Psi_{n+1}^{-1}$  is symmetric and positive-definite, so we can reparameterize  $\tilde{\theta}_{n+1}$  as follows:

$$\tilde{\theta}_{n+1} = \theta_{n+1} + \alpha B_{n+1} z \quad (5)$$

where  $B_{n+1} B_{n+1}^\top = \Psi_{n+1}^{-1}$ , and  $z \sim \mathcal{N}(0, I_d)$ .

Then we can rewrite the step 5 of Algorithm 1 as:

$$I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \theta_{n+1}, s_I \rangle + \alpha s_I^\top B_{n+1} z \quad (6)$$

Note that for any matrix  $A$  with dimension of  $r \times p$ ,  $C$  with dimension of  $s \times q$ , any random variable  $x$  with dimension of  $p \times 1$ ,  $y$  with dimension of  $q \times 1$  and any constant vector  $b$  with dimension of  $r \times 1$ ,  $d$  with dimension of  $s \times 1$ , where  $r, p, s, q$  is constant number, we have:

$$\begin{aligned} Cov(A_{r \times p} x_{p \times 1} + b_{r \times 1}, C_{s \times q} y_{q \times 1} + d_{s \times 1}) \\ = ACov(x, y)C^\top \end{aligned} \quad (7)$$

where  $Cov(\cdot, \cdot)$  means the covariance matrix.

Combining  $\mathbf{s}_I^\top \mathbf{B}_{n+1} \mathbf{z}$  and (7) yields that:

$$\begin{aligned} \text{Cov}(\mathbf{s}_I^\top \mathbf{B}_{n+1} \mathbf{z}, \mathbf{s}_I^\top \mathbf{B}_{n+1} \mathbf{z}) &= \mathbf{s}_I^\top \mathbf{B}_{n+1} \text{Cov}(\mathbf{z}, \mathbf{z}) \mathbf{B}_{n+1}^\top \mathbf{s}_I \\ &= \mathbf{s}_I^\top \mathbf{B}_{n+1} \mathbf{B}_{n+1}^\top \mathbf{s}_I \\ &= \mathbf{s}_I^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_I \end{aligned} \quad (8)$$

And  $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_d)$ , so we have:

$$\mathbf{s}_I^\top \mathbf{B}_{n+1} \mathbf{z} \sim \mathcal{N}(0, \|\mathbf{s}_I^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_I\|^2) \quad (9)$$

Note that:

$$\left[ \mathbf{s}_I^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_I \right]^{\frac{1}{2}} z \sim \mathcal{N}(0, \|\mathbf{s}_I^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_I\|^2) \quad (10)$$

where  $z \sim \mathcal{N}(0, 1)$

So finally, after combining (9) and (10), we can rewritten (6) as:

$$I_{n,b} \leftarrow \arg \max_{I \in \mathcal{I}} \langle \boldsymbol{\theta}_{n+1}, \mathbf{s}_I \rangle + \alpha [\mathbf{s}_I^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_I]^{\frac{1}{2}} z \quad (11)$$

□

## 4.2 IBCB's Formulation

We will illustrate the formulation using two actions as an example, which can be easily extended to accommodate multiple actions by adding additional constraints. Specifically, given  $\mathcal{S}_{n+1,b} = \{\mathbf{s}_1, \mathbf{s}_2\}$ ,  $\mathcal{I} = \{1, 2\}$ , if action 1 is executed at step  $b$  in the  $(n+1)$ -th episode, we have

$$\begin{aligned} &\langle (\mathbf{\Psi}_{n+1})^{-1} \mathbf{b}_{n+1}, \mathbf{s}_1 \rangle + \alpha [\mathbf{s}_1^\top (\mathbf{\Psi}_{n+1})^{-1} \mathbf{s}_1]^{\frac{1}{2}} \\ &\geq \langle (\mathbf{\Psi}_{n+1})^{-1} \mathbf{b}_{n+1}, \mathbf{s}_2 \rangle + \alpha [\mathbf{s}_2^\top (\mathbf{\Psi}_{n+1})^{-1} \mathbf{s}_2]^{\frac{1}{2}}, \end{aligned}$$

which is equivalent to

$$\begin{aligned} &\langle \mathbf{b}_{n+1}, (\mathbf{\Psi}_{n+1})^{-1} (\mathbf{s}_1 - \mathbf{s}_2) \rangle \\ &\geq \alpha \left\{ [\mathbf{s}_2^\top (\mathbf{\Psi}_{n+1})^{-1} \mathbf{s}_2]^{\frac{1}{2}} - [\mathbf{s}_1^\top (\mathbf{\Psi}_{n+1})^{-1} \mathbf{s}_1]^{\frac{1}{2}} \right\}. \end{aligned} \quad (12)$$

Setting  $n = 1$  in Eq.(12), from  $\mathbf{b}_2 = \mathbf{b}_1 + \mathbf{S}_1^\top \mathbf{R}_1 = \mathbf{S}_1^\top \mathbf{R}_1$  we get

$$\begin{aligned} &\langle \mathbf{R}_1, \mathbf{S}_1 \mathbf{\Psi}_2^{-1} (\mathbf{s}_1 - \mathbf{s}_2) \rangle \\ &\geq \alpha \left\{ [\mathbf{s}_2^\top \mathbf{\Psi}_2^{-1} \mathbf{s}_2]^{\frac{1}{2}} - [\mathbf{s}_1^\top \mathbf{\Psi}_2^{-1} \mathbf{s}_1]^{\frac{1}{2}} \right\}, \end{aligned} \quad (13)$$

where  $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{S}_{2,b}, \forall b \in [B]$ . More generally, for all  $n \in [N-1]$ , given a candidate action set  $\mathcal{S}_{n+1,b} = \{\mathbf{s}_1, \mathbf{s}_2\}$ , if action 1 is executed at step  $b$  in the  $(n+1)$ -th episode (i.e.,  $\mathbf{s}_1$  is selected), we can obtain that, for  $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{S}_{2,b}, \forall b \in [B]$ .

$$\begin{aligned} &\sum_{i \in [n]} \langle \mathbf{R}_i, \mathbf{S}_i \mathbf{\Psi}_{n+1}^{-1} (\mathbf{s}_1 - \mathbf{s}_2) \rangle \\ &\geq \alpha \left\{ [\mathbf{s}_2^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_2]^{\frac{1}{2}} - [\mathbf{s}_1^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_1]^{\frac{1}{2}} \right\}, \end{aligned} \quad (14)$$

otherwise action 2 is selected for execution at step  $b$  in the  $(n+1)$ -th episode.

Taking expectations of both sides of the inequality Eq.(14), we have

$$\begin{aligned} &\mathbb{E} \left[ \sum_{i \in [n]} \langle \mathbf{R}_i, \mathbf{S}_i \mathbf{\Psi}_{n+1}^{-1} (\mathbf{s}_1 - \mathbf{s}_2) \rangle \middle| \mathbf{S}_1, \mathbf{S}_2, \dots, \mathbf{S}_n \right] \\ &\geq \alpha \left\{ [\mathbf{s}_2^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_2]^{\frac{1}{2}} - [\mathbf{s}_1^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_1]^{\frac{1}{2}} \right\}, \end{aligned} \quad (15)$$

Combining linear contextual bandit's reward assumption in section 3, Eq.(15) is equivalent to

$$\begin{aligned} &\sum_{i \in [n]} \left\langle \boldsymbol{\theta}^*, \mathbf{S}_i^\top \mathbf{S}_i \mathbf{\Psi}_{n+1}^{-1} (\mathbf{s}_1 - \mathbf{s}_2) \right\rangle \\ &\geq \alpha \left\{ [\mathbf{s}_2^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_2]^{\frac{1}{2}} - [\mathbf{s}_1^\top \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}_1]^{\frac{1}{2}} \right\}. \end{aligned} \quad (16)$$

Under the constraints Eq.(16), we can obtain the minimum-norm solution  $\boldsymbol{\theta}^*$  by solving the following optimization problem with inequality constraints:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \frac{1}{2} \|\boldsymbol{\theta}\|_2^2 \quad (17)$$

$$\text{s.t.} \sum_{i \in [n]} \left\langle \boldsymbol{\theta}, \mathbf{S}_i^\top \mathbf{S}_i \left[ \phi_n(\mathbf{s}_{I_{n,b}}) - \phi_n(\mathbf{s}_{I_{n,b}}) \right] \right\rangle \quad (18)$$

$$\leq \alpha \left[ H_n(\mathbf{s}_{I_{n,b}}) - H_n(\mathbf{s}_{\bar{I}_{n,b}}) \right], \quad n \in [N], b \in [B],$$

where  $\phi_n(\mathbf{s}) = \mathbf{\Psi}_{n+1}^{-1} \mathbf{s}$ ,  $H_n(\mathbf{s}) = [\mathbf{s}^\top \phi_n(\mathbf{s})]^{\frac{1}{2}}$ . Eq.(17) is the final formulation of IBCB, which proposed a simple quadratic programming (QP) problem with large scale linear constraints. Several tools have been introduced to efficiently solve QP problems ([45], [46], [47], [48], [49]). Among these approaches, OSQP [49] is the best to achieve almost linear complexity growth in problems' dimension for large scale low-accuracy Random QP problems, which can be applied to our IBCB. We also provide more detailed solving time comparison of IBCB on different problem scales in Section 5.6.9.

When we assume that original expert algorithm is SBTS, we can simply apply a random parameter for  $H_n(\mathbf{s})$  as  $H_n(\mathbf{s}) = [\mathbf{s}^\top \phi_n(\mathbf{s})]^{\frac{1}{2}} z$ , where  $z \sim \mathcal{N}(0, 1)$  by following Theorem 1 and clear  $H_n(\mathbf{s})$  to 0 by following Eq.(15)'s expectation setting. This indicates that IBCB is a unified framework for both deterministic and randomized BCB.

### 4.2.1 Algorithm Implementation

Existing inverse bandit algorithm [15] estimates parameters using an EM-like algorithm, which iteratively maximizes the likelihood estimation. However, this algorithm suffers from low computational efficiency as it requires a large number of iterations and samples to achieve accurate estimation. Furthermore, the assumption of expert updates after each action under the fully-online setting leads to a rapid increase in computation time when sampling a large number of actions, making it unsuitable for the BCB setting. In contrast, the proposed IBCB capitalizes on the BCB assumption, which provides a unified framework for both deterministic and randomized bandit policies. By formulating the inverse bandit problem as a quadratic optimization problem with a significant number of linear constraints, IBCB offers a more efficient solution through leveraging the capabilities of the OSQP optimizer. Furthermore, since solving QP problems with full constraints costs high computation resources, and to better fulfill the updating method adopted in BCB assumption, we introduce the incremental implementation of IBCB algorithm.

We provide pseudo code in the following part to describe the details of IBCB's incremental implementation in Algorithm 2.

**Algorithm 2** Incremental implementation of IBCB algorithm

**INPUT:** Initialize  $\theta_0 = \mathbf{0}_d$ , number of episodes  $N$ , batch size  $B$ , data buffer  $\{\mathcal{D}_n\}_{n \in [N]}$ ,  $\lambda = \lambda_0$ ,  $\alpha = \alpha_0$ ,  $\Phi_1 = \mathbf{O}_d$ .

- 1: **for**  $n = 1$  **to**  $N$  **do**
- 2:   // Load previous data buffer
- 3:   Load last episode's selected context vectors  $\mathbf{S}_n \in \mathbb{R}^{B \times d}$  from data buffer  $\mathcal{D}_n$ .
- 4:   Initialize current episode's constraint set  $C_n = \emptyset$ .
- 5:   **for**  $b = 1$  **to**  $B$  **do**
- 6:     Load  $b$ -th step of  $n$ -th episode's selected item  $I_{n,b}$ , other candidate items  $\bar{I}_{n,b}$  and all candidate context vectors  $\mathbf{s}$ .
- 7:      $\Phi_{n+1} \leftarrow \Phi_n + \mathbf{S}_n^\top \mathbf{S}_n$ ,
- 8:      $\Psi_{n+1} \leftarrow \lambda \mathbf{I}_d + \Phi_{n+1}$ .
- 9:     // Construct constraints
- 10:     Construct current step's constraint element  $c$  as:  

$$c \leftarrow \left\{ \sum_{i \in [n]} \left\langle \theta, \mathbf{S}_i^\top \mathbf{S}_i \left[ \phi_n(\mathbf{s}_{\bar{I}_{n,b}}) - \phi_n(\mathbf{s}_{I_{n,b}}) \right] \right\rangle \leq \alpha \left[ H_n(\mathbf{s}_{I_{n,b}}) - H_n(\mathbf{s}_{\bar{I}_{n,b}}) \right] \right\},$$
 where  $\phi_n(\mathbf{s}) = \Psi_{n+1}^{-1} \mathbf{s}$ ,  $H_n(\mathbf{s}) = [\mathbf{s}^\top \phi_n(\mathbf{s})]^\frac{1}{2}$ .  
 Update:  $C_n \leftarrow C_n \cup c$ .
- 11:   **end for**
- 12:   // Solve the QP problem
- 13:   Solve the following optimization problem with current episode's constraint set  $C_n$  through OSQP optimizer:  

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|\theta - \theta_{n-1}\|_2^2 \quad \text{s.t. } C_n$$
- 14:   // Incrementally update
- 15:   Use solved parameter  $\theta$  to update:  $\theta_n = \theta$ .
- 16:   **end for**
- 17: **Return** Learned expert policy parameter  $\theta_N$  (reward parameter  $\theta^*$ )

IBCB algorithm is designed to mimic specific evolution process made by experts. Therefore, instead of learning a new policy from scratch at each step, incremental IBCB learns an update to the previously inferred policy. The objective function  $1/2 \|\theta - \theta_{n-1}\|_2^2$  naturally models this by minimizing the squared L2-norm distance between the new policy parameters  $\theta$  and the parameters from the previous batch  $\theta_{n-1}$ . This incremental learning approach is central to IBCB's design.

**4.2.2 Analysis of time complexity of IBCB**

After using incremental implementation, overall time complexity of IBCB solution is reduced from  $O((NBM)^\beta d)$  (non-incremental IBCB algorithm) to  $O(N(BM)^\beta d)$  (incremental IBCB algorithm), where  $N$  is the total number of episodes,  $B$  is the total number of steps in each episode,  $M$  is the size of the action space, and  $d$  is the context space dimension.  $\beta$  is the adaptive complexity parameter corresponding to the QP problem of different constraints for OSQP. According to the experimental results and proof of OSQP,  $\beta \in [1, 2]$ . When the quantity of constraints is fixed,  $\beta$  will adjust adaptively. When there are moderate constraints ( $\leq 10^6$  constraints),  $\beta$  will tend to 1. Analysis can be found in the following part.

*Proof.* For any  $n \in N$ , i.e., in the  $n$ -th episode,

1. Solving matrix  $\phi_{n+1}$  in Algorithm 2 in Appendix A.1 requires matrix multiplication, with a time complexity of  $O(dBd) = O(Bd^2)$ .
2. Constructing  $B * (M - 1)$  constraints for each step in an episode with  $M$  actions requires a time complexity of  $O(BMd^3)$ .
3. Using the current  $B * (M - 1)$  constraints to solve with OSQP requires a time complexity of  $O((BM)^\beta d)$ , where  $\beta$  is the adaptive complexity parameter,  $\beta \in [1, 2]$ , close to 1 when constraints  $\leq 10^7$ . Therefore, in the incremental

update training of the IBCB algorithm, the time complexity of one episode is  $O(Bd^2 + BMd^3 + (BM)^\beta d)$ . Considering that  $d$  is usually less than 100 in practice, the basic unit time for matrix multiplication and inversion is very low. Therefore, the time complexity of one episode is  $O(BM + (BM)^\beta d) = O((BM)^\beta d)$ . So, in the setting of a total of  $N$  episodes, the training time complexity of the incremental IBCB is  $O(N(BM)^\beta d)$ , where  $\beta$  is the adaptive complexity parameter,  $\beta \in [1, 2]$ , close to 1 when constraints  $\leq 10^6$ .

4. If the incremental learning algorithm is not used to implement IBCB, i.e. non-incremental IBCB, OSQP needs to solve  $NB * (M - 1)$  constraints simultaneously, so the total time complexity of the non-incremental implementation is  $O((NBM)^\beta d)$ .

**4.2.3 Extension to Fairness-Aware Expert Limitation**

According to [11], a fairness-aware expert selects actions based on a nonlinear transformation of their selection probabilities, such as using Softmax, to ensure fairness by not only execute the best action. However, this can significantly reduce the average reward. To address this, we've enhanced the **top-k** contextual bandit recommendation algorithm from [50] with a compromise policy: choosing the final action from the **top-k** based on normalized probabilities, which balances reward and fairness. Our IBCB can be extended to imitate the above fairness-aware expert. Specifically, during each training episode, IBCB first trains for one round to obtain the initial reward parameters  $\theta_\tau$  in current episode  $n$ . IBCB then uses  $\theta_\tau$  to clean up the inequality constraints in Eq. (17) as a pre-process method, removing constraints that do not meet the criteria since expert may not select the best action, and subsequently retrains for one episode based on the remaining constraints.

Specifically, considering that IBCB has a corresponding incremental learning framework, combined with the update method of the BCB for each episode, we have made a new modification to IBCB after adding fairness. Details can be found in the following pseudo code in Algorithm 3.

**5 EXPERIMENTS**

**5.1 Baselines**

IBCB was compared with several existing original expert algorithms and imitation learning (IL) algorithms, including:

**5.1.1 Original Expert Algorithms with Rewards (SBUCB & SBTS)**

We used SBUCB and SBTS for original expert algorithms. These algorithms' performances are the oracle limit above all of other IL algorithms and IBCB since they have access to the rewards. SBUCB is a batched version of LinUCB [19], which updates the policy after receiving a batch of feedback data [17]. SBTS is a batched version of Linear Thompson Sampling (LinTS) [18].

**5.1.2 Behavior Cloning (BC) Algorithms**

We used LinearSVC [51] model (namely BaseSVC) for BC baseline, since the we assumed reward environment is linear in Section 3.

**Algorithm 3** IBCB’s adaptation to fairness-aware expert limitation

**INPUT:** Initialize  $\theta_0 = \mathbf{0}_d$ , number of episodes  $N$ , batch size  $B$ , data buffer  $\{\mathcal{D}_n\}_{n \in [N]}$ ,  $\lambda = \lambda_0$ ,  $\alpha = \alpha_0$ ,  $\Phi_1 = \mathbf{O}_d$ .

- 1: **for**  $n = 1$  **to**  $N$  **do**
- 2:   // Load previous data buffer
- 3:   Load last episode’s selected context vectors  $\mathbf{S}_n \in \mathbb{R}^{B \times d}$  from data buffer  $\mathcal{D}_n$ .
- 4:   Initialize current episode’s constraint set  $C_n = \emptyset$ .
- 5:   **for**  $b = 1$  **to**  $B$  **do**
- 6:     Load  $b$ -th step of  $n$ -th episode’s selected item  $I_{n,b}$ , other candidate items  $\bar{I}_{n,b}$  and all candidate context vectors  $\mathbf{s}$ .
- 7:      $\Phi_{n+1} \leftarrow \Phi_n + \mathbf{S}_n^\top \mathbf{S}_n$ ,
- 8:      $\Psi_{n+1} \leftarrow \lambda \mathbf{I}_d + \Phi_{n+1}$ .
- 9:     // Construct constraints
- 10:    Construct current step’s constraint element  $c$  as:  

$$c \leftarrow \left\{ \sum_{i \in [n]} \left\langle \theta, \mathbf{S}_i^\top \mathbf{S}_i \left[ \phi_n(\mathbf{s}_{\bar{I}_{n,b}}) - \phi_n(\mathbf{s}_{I_{n,b}}) \right] \right\rangle \leq \alpha \left[ H_n(\mathbf{s}_{I_{n,b}}) - H_n(\mathbf{s}_{\bar{I}_{n,b}}) \right] \right\}$$
 where  $\phi_n(\mathbf{s}) = \Psi_{n+1}^{-1} \mathbf{s}$ ,  $H_n(\mathbf{s}) = [\mathbf{s}^\top \phi_n(\mathbf{s})]^\frac{1}{2}$ .  
 Update:  $C_n \leftarrow C_n \cup c$ .
- 11:    **end for**
- 12:    // Solve the QP problem
- 13:    Solve the following optimization problem with current episode’s constraint set  $C_n$  through OSQP optimizer:  

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|\theta - \theta_{n-1}\|_2^2 \quad \text{s.t. } C_n$$
- 14:    // Incrementally update
- 15:    Use solved parameter  $\theta$  as this episode’s initial reward parameter  $\theta_r$ :  $\theta_r = \theta$ .
- 16:    // Use initial reward parameter -
- 17:    // - to clean up unsatisfied constraints
- 18:    Initialize current episode’s unsatisfied constraint set  $C_{n-} = \emptyset$ .
- 19:    **for**  $b = 1$  **to**  $B$  **do**
- 20:     **if**  $\sum_{i \in [n]} \left\langle \theta_r, \mathbf{S}_i^\top \mathbf{S}_i \left[ \phi_n(\mathbf{s}_{\bar{I}_{n,b}}) - \phi_n(\mathbf{s}_{I_{n,b}}) \right] \right\rangle > \alpha \left[ H_n(\mathbf{s}_{I_{n,b}}) - H_n(\mathbf{s}_{\bar{I}_{n,b}}) \right]$  **then**
- 21:      Construct current step’s unsatisfied constraint element  $c_-$  as:  

$$c_- \leftarrow \left\{ \sum_{i \in [n]} \left\langle \theta_r, \mathbf{S}_i^\top \mathbf{S}_i \left[ \phi_n(\mathbf{s}_{\bar{I}_{n,b}}) - \phi_n(\mathbf{s}_{I_{n,b}}) \right] \right\rangle > \alpha \left[ H_n(\mathbf{s}_{I_{n,b}}) - H_n(\mathbf{s}_{\bar{I}_{n,b}}) \right] \right\}$$
- 22:      Update unsatisfied constraint set:  $C_{n-} \leftarrow C_{n-} \cup c_-$ .
- 23:     **end if**
- 24:    **end for**
- 25:    Initialize current episode’s satisfied constraint set  $C'_n$  by removing unsatisfied constraints:  $C'_n \leftarrow C_n \setminus C_{n-}$ .
- 26:    // Solve QP problem -
- 27:    // - with satisfied constraints
- 28:    Solve the following optimization problem with current episode’s satisfied constraint set  $C'_n$  through OSQP optimizer:  

$$\min_{\theta' \in \mathbb{R}^d} \frac{1}{2} \|\theta' - \theta_{n-1}\|_2^2 \quad \text{s.t. } C'_n$$
- 29:    // Incrementally update
- 30:    Use solved parameter  $\theta'$  to update:  $\theta_n = \theta'$ .
- 31:    **end for**
- 32: **end for**
- 33: **Return** Learned expert policy parameter  $\theta_N$  (reward parameter  $\theta^*$ )

**5.1.3 Inverse Reinforcement Learning (IRL) Algorithms**

We used Bayesian Inversive Reinforcement Learning (B-IRL) [13], and current state-of-the-art Bayesian Inversive Contextual Bandits (B-ICB) [15] and One-Shot Inverse Reinforcement Learning Stochastic Linear Bandits (OSIRL-CB) [16] to serve as IRL baselines.

**5.2 Datasets**

We designed synthetic and real-world dataset for experiments. For both of them, we split the whole dataset into two subsets: the Online Learning (the OL-data) phase, and the Batch Testing (the BT-data) phase. Note that expert only update its policy parameters in OL-data, and do not update

parameter in BT-data phase so we could design experiments on estimating expert policy’s parameters on BT-data.

**5.2.1 Synthetic Dataset**

To ensure our experimental setup is robust and comparable to existing work, we adopted the simulation settings directly from the recent paper [52]: We set the synthetic dataset’s candidate context  $\mathbf{s}_I$  with context dimension  $d = 10$  as a single context (behavior) for any step of an episode. At step  $b$  of  $n$ -th episode in OL-data phase, candidate context set  $\mathcal{S}_{n,b} \subseteq \mathbb{R}^d$  was drawn from a Gaussian distribution  $\mathcal{N}(\mu_s \mathbf{1}_d, \sigma_s^2 \mathbf{I}_d)$ , where the means of candidate contexts were  $\mu_s \in [1 : -0.4 : -2.6]$ , and the standard deviation was  $\sigma_s^2 = 0.05^2$ . To simulate out-of-distribution(OOD) scenario, the mean of the first candidate context in the BT-data phase was set to 1.4. We set the observed reward of a given candidate context  $\mathbf{s}_{I_{n,b}} \in \mathcal{S}_{n,b}$  at step  $b$  in  $n$ -th episode as a sigmoid function  $\text{sigmoid}(\langle \mathbf{w}_{\text{reward}}, \mathbf{s}_{I_{n,b}} \rangle)$ , where each element of  $\mathbf{w}_{\text{reward}} \in \mathbb{R}^d$  was sampled from a Gaussian distribution as  $\mathcal{N}(0.1, 0.01^2)$ . To test the robustness of the model and simulate random selections of actions made by users, we also added a zero-mean Gaussian noise  $\text{std} \sim \mathcal{N}(0, \epsilon^2)$ ,  $\epsilon \in \{0.03, 0.07, 0.10\}$  in the reward, so final reward was set to be  $\text{sigmoid}(\langle \mathbf{w}_{\text{reward}}, \mathbf{s}_{I_{n,b}} \rangle) + \text{std}$ . We also made adjustments for the ablation study period. Contradictory data was introduced as *duplicated data*: we cloned the first  $k$  episodes’ candidate context sets for the second  $k$  episode’s candidate context sets, and design  $k$  with the name of *duplicated quantity*  $\text{dup} \in [0 : 1 : 9]$ . Total number of conflict executed actions (e.g. *expert select action 1 at step b of n-th episode, but select action 2 at step b of (n + k)-th episode.*) of the contradictory data was ascending when we increased  $\text{dup}$ . We also controlled the training rate for all baselines and IBCB: we set the end episode of the exposed expert’s behavioral evolution history logs with constraint end rate  $\text{ce\_rate} \in [0, 1]$  (e.g. if  $\text{ce\_rate} = 0.5$ . we used the first  $[0.5N]$  episodes for baselines and IBCB training).

In a nutshell, we generated the synthetic data for both OL-data and BT-data as follows: numbers of episodes  $N = 20$ , batch size  $B = 1000$ , number of candidate contexts (actions) is  $M = 10$  at any single step of an episode.

**5.2.2 Real-World Dataset**

We used MovieLens 100K <sup>1</sup> (ML-100K) dataset as our real-world experiment dataset. , which collected 5-star movie ratings from a movie recommendation service, including 943 unique users and 1682 unique movies (items). ML-100K dataset also provided various side information of both users and items. We split ML-100K dataset into OL-Data and BT-data according to the timestamps, where first half was treated as OL-data and the last half was BT-data. For reward setting, we treated the 4-5 star ratings as *positive feedback*(labeled with 1), and 1-3 star rating as *negative feedback* (labeled with 0). Through feature embedding [53] and principal component analysis (PCA), dimension of each [user, item] pair (candidate context, i.e., *behavior*) was reduced to 50, i.e.,  $d = 50$ . For the candidate context set, we retained the original item for each interaction and then randomly sampled 9 extra items from the entire item set

1. <https://grouplens.org/datasets/movielens/100k/>

(exclude the original item), so the number of candidates is  $M = 10$ . In real recommendation, user-item interaction is sparse, meaning that not all of the items can be shown to the user and have feedback [6], [7]. To overcome this issue, we trained two Matrix Factorization (MF) [54] models using OL-data (denotes MF-OL) and BT-data (denotes MF-BT) respectively for different baselines and IBCB model to serve as the simulated real environment. AUCs of the two MF models were both over 90%, which assures that the simulated environment can provide nearly realistic feedbacks of users. At step  $b$  of  $n$ -th episode of both OL-data and BT-data phase, the simulated environment (MF-OL/MF-BT) received a selected candidate context  $s_{I_{n,b}}$  and a context set  $S_{n,b}$  consisted of all candidate contexts in this step, getting all candidate context's reward and sort out the candidate context  $s_{I_{\max}}$  with max reward  $r_{I_{\max}}$ . If selected candidate context  $s_{I_{n,b}}$  do not have the biggest reward compared to  $s_{I_{\max}}$ , *i.e.*,  $r_{I_{n,b}} < r_{I_{\max}}$ , reward feedback would be directly set as 0, otherwise would be set as  $r_{I_{\max}}$ . If  $r_{I_{\max}} > 0.5$ , then the final reward feedback would be 1, otherwise would be 0. Adjustments for the ablation study period was also made similar with the synthetic dataset. Since duplicated data will not appeared in a large scale in real world, we just controlled the training rate similarly as that in synthetic dataset's setting.

Note that MF-OL is quite different compared to MF-BT (*i.e.*, BT-data's distribution is different to OL-data's distribution), so experiments made on BT-data can directly provide the results of model's generalization ability in out-of-distribution data.

Finally, for OL-data in real-world dataset, we set numbers of episodes  $N = 20$ , batch size  $B = 1000$ , and for BT-data, we set  $N = 50$ ,  $B = 1000$  for precise measurement, considering real-world data's noise. Number of candidate contexts (actions) is  $M = 10$  at any single step of an episode in both OL-data and BT-data phase.

### 5.3 Evaluation Protocol

#### 5.3.1 Online Train Log Fitness (OL-Fitness)

OL-Fitness measures the alignment of various algorithms' learned parameters with the true reward environment's parameters. It involves using the learned parameters from IL algorithms (including IBCB) or backbone bandit algorithms to simulate a reward environment, where an expert interacts to gather data for online learning. The actions taken by the expert in this simulated environment are compared against those taken in the true reward environment, calculated as  $\frac{1}{NB} \sum_{n=1}^N \sum_{b=1}^B \mathbb{I}(I_{n,b-\text{estimated}} = I_{n,b-\text{true}})$ . Here,  $I_{n,b-\text{estimated}}$  denotes the action index from the estimated parameters in step  $b$  of episode  $n$  during the OL-data phase, and  $I_{n,b-\text{true}}$  is the action index from the true reward parameters under the same conditions. OL-data's candidate context sets, sequences and reward parameters keep same during the comparison by using fixed seeds. Note that BC algorithms cannot obtain reward parameters, and thus were not included in this comparison scope. But in *dup-test* period, we used 'train log behaviors fitness' to evaluate whether BC algorithm suffers from contradictory data. A higher OL-Fitness indicates a closer match between the learned parameters and the true reward parameters.

#### 5.3.2 Batch Test Log Fitness (BT-Fitness)

BT-Fitness measures the alignment of parameters/models from various algorithms with those of the final expert policy. It is calculated by comparing the actions executed by IL algorithms against those of the expert policy during the BT-data phase, using the formula  $\frac{1}{NB} \sum_{n=1}^N \sum_{b=1}^B \mathbb{I}(I_{n,b-\text{IL}} = I_{n,b-\text{expert}})$ . Here,  $I_{n,b-\text{IL}}$  represents the action index executed by IL algorithms (including IBCB), while  $I_{n,b-\text{expert}}$  is the action index from the expert policy in step  $b$  of episode  $n$ . BT-data's candidate context sets, sequences and reward parameters keep same during the comparison by using fixed seeds, and expert do not update its policy parameters in BT-data phase. Higher BT-Fitness indicates greater similarity to the expert policy's parameters.

#### 5.3.3 Batch Test Average Reward (BT-AR)

measures the reward performance of IL algorithms (including IBCB) against expert algorithms during the BT-data phase. It calculates average reward from executed actions after interacting with the BT-data environment, denoted as  $\frac{1}{NB} \sum_{n=1}^N \sum_{b=1}^B R_{n,b}$ .  $R_{n,b}$  is binary reward for actions at step  $b$  of episode  $n$ . Higher BT-AR indicates more effective actions in the BT-data environment, suggesting superior reward performance by the algorithm.

#### 5.3.4 Cumulative Fairness Regret (CFR)

CFR measures the similarity in fairness between the policies of various IL algorithms (including IBCB) and the bandit policy after the final episode (*i.e.*, the experienced expert) with fairness optimization. Following definitions of fairness regret in [11], [12], we define CFR as  $\text{CFR} = \sum_{n=1}^N \sum_{b=1}^B (D_{\text{JS}}(\pi_{\text{IL}}(S_{n,b}) || \pi_{\text{expert}}(S_{n,b})) + |R_{n,b-\text{IL}} - R_{n,b-\text{expert}}|)$ ,  $R_{n,b-\text{IL}}, R_{n,b-\text{expert}} \in \{0, 1\}$ , where  $D_{\text{JS}}(P || Q)$  represents the Jensen-Shannon divergence [55] between distributions  $P$  and  $Q$ .  $\pi_{\text{IL}}(S_{n,b})$  refers to the policy obtained by various IL algorithms (including IBCB), and  $\pi_{\text{expert}}(S_{n,b})$  refers to the bandit policy after the final episode (*i.e.*, the experienced expert). Both of the above are the normalized probabilities of each context which can be selected in the candidate context list  $S_{n,b}$  at step  $b$  of episode  $n$  in BT-data phase.  $R_{n,b-\text{IL}}$  and  $R_{n,b-\text{expert}}$  are executed action's observed reward at step  $b$  of the  $n$ -th episode in BT-data phase from learned model and final expert respectively. It should be noted that CFR is only used in the BT-data phase, that is, when the parameters of all models have been fixed. A lower CFR indicates a closer similarity to the fairness of the final expert policy, that is, a more similar performance to the expert policy with fairness optimization.

### 5.4 Training Details

We run all of the experiments on the server equipped with 16 Intel Xeon E5-2630 v4@2.20GHz cores, and during training, all of the baselines and IBCB only uses the CPU. Note that all baselines and IBCB were training on expert behavioral evolution history of Online Learning. Detailed settings of baselines and IBCB are shown as follows.

### 5.4.1 Original Expert Policy Settings

**SBUCB:** We set the hyper parameter ( $\alpha$  in step 5 of Algorithm 1, *i.e.*, exploration parameter) of SBUCB to 0.4 for both synthetic and real-world ML-100K dataset experiments. Note that this parameter is only effective in Online Learning phase since expert (SBUCB) do not update its policy parameters (*i.e.*, explore) in Batch Test phase.

**SBTS:** We set the hyper parameter ( $\alpha$  in step 7 of Algorithm 1, *i.e.*, exploration parameter) of SBTS to 0.4 for synthetic dataset experiments and 0.05 for real-world ML-100K dataset experiments. Note that this parameter is only effective in Online Learning phase since expert (SBTS) do not update its policy parameters (*i.e.*, explore) in Batch Test phase.

### 5.4.2 Baseline Settings

**BaseSVC:** We followed the setting recommended in `sklearn.svm.LinearSVC` and set the `max_iteration` (*i.e.*, the maximum number of iterations) to 1,000,000 to ensure that the model can converge in the end.

**B-IRL:** Following [15], we have run Metropolis-Hastings algorithm for 10,000 iterations to obtain 1,000 samples on synthetic dataset (500 samples and 5,000 iterations for real-world ML-100K dataset) with intervals of 10 iterations between each sample after 10,000 burn-in iterations (5,000 for real-world ML-100K dataset). The final estimated parameters are formed by simply averaging all 1,000 samples. (500 for real-world ML-100K dataset).

**B-ICB:** We followed the setting used in [15]’s work. We changed the setting of `compute_rhox` period (*i.e.*, computing the reward parameters) in real-world ML-100K dataset experiments. Since that ML-100K’s user embedding used one-hot encoding, which may make the final matrix  $\beta_n$  to concatenate in `compute_rhox` be non-invertible, we replace the original `np.linalg.inv` method with `np.linalg.pinv` method to compute the pseudo inverse of  $\beta_n$ , making sure experiment can be done without computational errors. And due to that  $\beta_n$  may be non-invertible, we also dismiss the  $\det(\beta_n/\sigma^2)$  during iteration, where  $\det(\mathbf{A})$  means the determinant of matrix  $\mathbf{A}$ .

**OSIRL-CB:** We followed [16]’s setting in their work. Key parameters are set as follows: For the demonstration trajectory, the Phased Elimination error parameter  $\iota = 0.5$  and the failure probability  $\delta = 0.1$ . For OSIRL-CB algorithm parameters, the action dimension  $d$  is consistent with specific dataset, the action set smoothness  $\omega = 3.0$ , and the projection distance threshold  $\gamma = 1e - 4$  (for action subset selection). No MCMC sampling is used, and  $\hat{\theta}$  is directly solved by the least-squares method in a single-round trajectory processing.

**Data Sampling Rate for B-ICB** denotes the proportion of training data B-ICB used for training since B-ICB cannot be run with large-scale training data. When the data sampling rate is  $k\%$ , we uniformly randomly sample  $k\%$  of the original expert’s training data as the training data (behavioral evolution history) for B-ICB. Data sampling rate was set to be 1% to make sure that B-ICB can be trained in 100s on synthetic dataset experiments, and 0.5% to make sure that B-ICB can be trained in 150s on real-world ML-100K dataset experiments. Also, in ablation study section, we used this setting for B-ICB.

### 5.4.3 IBCB Settings

We used OSQP solver to solve Eq.(17)’s quadratic problem(QP) as mentioned. Tolerance levels of OSQP was set to be  $\epsilon_{abs} = 8 \times 10^{-2}$ ,  $\epsilon_{rel} = 8 \times 10^{-2}$ . If QP with the former setting cannot convergent, then setting would be modified as  $\epsilon_{abs} = 8 \times 10^{-1}$ ,  $\epsilon_{rel} = 8 \times 10^{-1}$ . For real-world ML-100K experiments with SBTS backbone, setting was  $\epsilon_{abs} = 3 \times 10^{-1}$ ,  $\epsilon_{rel} = 3 \times 10^{-1}$ . Also, when expert’s policy is SBTS, right-hand side of Eq.(17) should be set to a constant  $\epsilon$  slightly greater than 0 to prevent the parameters learned by IBCB from being too close to the origin ( $\mathbf{0}_d$ ). Therefore, we set IBCB’s  $\epsilon$  to 0.01 in the experimental settings of the synthetic dataset and 0.001 in real-world ML-100K dataset when we assume expert’s policy is SBTS.

## 5.5 Experimental Results & Analyses

In this section, we compared several indicators between IBCB with other baselines. We used data sampling method to train B-ICB (namely B-ICB-S, where ‘S’ means ‘used sampled data’) since B-ICB has not yet been adapted to batched version, so B-ICB can only be trained in environment that quantity of train data is not so large.

All results were averaged over 5 different runs, and sampled the data over 5 different random seeds for each train data log (behavioral evolution history log) of expert on both synthetic and real-world ML-100K dataset. The source code has been available at <https://anonymous.4open.science/r/IBCB-F25F>.

### 5.5.1 Experiments on Synthetic Dataset

Table 1 displays OL-Fitness and training time for experts (SBUCB & SBTS), baselines, and IBCB. IBCB outperforms others with higher OL-Fitness and faster training, thanks to its optimal reward parameter estimation and OSQP solver’s efficiency in managing large-scale QP problems with nearly linear complexity. BaseSVC trains slower due to its complexity near to quadratic, while B-IRL requires many Metropolis-Hastings iterations for better performance. OSIRL-CB slightly outperforms B-IRL through least-square estimation, but need more time to convergent. B-ICB’s quadratic complexity slows its training, despite sampling data.

Table 2 show that IBCB also achieves higher BT-Fitness and BT-AR, highlighting the effectiveness of its reward imputation method for learning from the expert’s behavioral evolution history. This confirms IBCB’s ability to learn the expert policy’s parameters effectively and efficiently, even in out-of-distribution (OOD) scenarios as seen in Table 7 in Section 5.6.1, demonstrating its robustness to distribution shifts.

### 5.5.2 Experiments on Real-World Dataset (ML-100K Dataset)

Results on ML-100K dataset of different indicators are reported in Table 3. Similar to the results on synthetic dataset, IBCB achieved significantly higher performance than other baselines in all indicators. From the outstanding results of training speed and average reward, we concluded that IBCB can also adapt to learning real-world behavioral evolution history generated by expert with great efficiency.

TABLE 1

Reward parameters' estimation (**Upper Subtable**) and Train time (**Lower Subtable**) comparison on synthetic dataset, where std denotes the standard deviation of zero-mean Gaussian noise added to the final reward. Experts (SBUCB & SBTS) do not need to train since they produce the train logs (behavioral evolution history logs) for baselines and IBCB.

Online Train Log Fitness (OL-Fitness)					Online Train Log Fitness (OL-Fitness)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	0.883±0.007	0.879±0.016	0.878±0.018	0.874±0.019	SBTS	0.857±0.017	0.876±0.012	0.872±0.014	0.871±0.008
B-IRL	0.827±0.012	0.808±0.039	0.806±0.037	0.802±0.036	B-IRL	0.457±0.020	0.468±0.029	0.480±0.023	0.476±0.039
B-ICB-S	0.805±0.020	0.771±0.031	0.780±0.034	0.771±0.024	B-ICB-S	0.424±0.020	0.451±0.036	0.457±0.016	0.454±0.024
OSIRL-CB	0.822±0.013	0.801±0.037	0.799±0.038	0.795±0.025	OSIRL-CB	0.453±0.026	0.461±0.026	0.477±0.034	0.459±0.042
<b>IBCB (Ours)</b>	<b>0.868±0.008</b>	<b>0.858±0.036</b>	<b>0.859±0.038</b>	<b>0.861±0.029</b>	<b>IBCB (Ours)</b>	<b>0.835±0.014</b>	<b>0.856±0.009</b>	<b>0.849±0.014</b>	<b>0.849±0.009</b>

Train Time (sec.)					Train Time (sec.)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	/	/	/	/	SBTS	/	/	/	/
B-IRL	36.997±0.803	37.67±1.328	39.91±3.419	37.424±0.563	B-IRL	38.46±3.550	36.419±0.187	37.843±3.486	43.469±6.228
BaseSVC	116.86±20.63	116.39±26.58	132.18±19.86	117.25±19.47	BaseSVC	96.603±9.619	87.310±8.435	89.240±6.566	90.472±8.494
B-ICB-S	86.365±0.827	87.599±4.489	86.85±1.162	85.713±0.676	B-ICB-S	91.166±1.164	89.193±1.477	89.892±1.162	89.483±1.585
OSIRL-CB	56.224±1.158	57.713±1.425	53.606±1.373	53.712±1.277	OSIRL-CB	54.182±2.565	54.131±2.121	57.739±4.377	59.981±6.361
<b>IBCB (Ours)</b>	<b>0.815±0.045</b>	<b>0.999±0.128</b>	<b>0.866±0.137</b>	<b>1.016±0.109</b>	<b>IBCB (Ours)</b>	<b>0.778±0.035</b>	<b>0.809±0.077</b>	<b>0.772±0.034</b>	<b>0.871±0.216</b>

TABLE 2

Expert policy (SBUCB) parameters' estimation (**Upper Subtable**), Expert policy (SBTS) parameters' estimation (**Lower Subtable**) on synthetic dataset. Expert policy produces same actions compared with itself, so Batch Test Log Fitness (BT-Fitness) of expert policy is 1.

Batch Test Log Fitness (BT-Fitness)					Batch Test Average Reward (BT-AR)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1	SBUCB	0.641±0.010	0.629±0.028	0.630±0.024	0.622±0.024
B-IRL	0.828±0.005	0.820±0.013	0.820±0.014	0.818±0.016	B-IRL	0.588±0.015	0.571±0.037	0.572±0.034	0.561±0.034
BaseSVC	0.791±0.008	0.781±0.020	0.781±0.020	0.777±0.020	BaseSVC	0.577±0.017	0.559±0.042	0.559±0.037	0.547±0.037
B-ICB-S	0.800±0.032	0.767±0.055	0.780±0.045	0.776±0.022	B-ICB-S	0.581±0.006	0.553±0.024	0.560±0.025	0.552±0.025
OSIRL-CB	0.799±0.005	0.791±0.014	0.792±0.013	0.790±0.015	OSIRL-CB	0.585±0.016	0.569±0.038	0.571±0.033	0.557±0.033
<b>IBCB (Ours)</b>	<b>0.972±0.011</b>	<b>0.963±0.031</b>	<b>0.964±0.028</b>	<b>0.972±0.016</b>	<b>IBCB (Ours)</b>	<b>0.647±0.012</b>	<b>0.638±0.026</b>	<b>0.639±0.019</b>	<b>0.628±0.022</b>

Batch Test Log Fitness (BT-Fitness)					Batch Test Average Reward (BT-AR)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBTS	1	1	1	1	SBTS	0.650±0.020	0.662±0.025	0.661±0.025	0.656±0.024
B-IRL	0.816±0.007	0.820±0.015	0.816±0.008	0.816±0.006	B-IRL	0.595±0.026	0.615±0.028	0.611±0.031	0.605±0.030
BaseSVC	0.789±0.011	0.807±0.009	0.801±0.015	0.798±0.012	BaseSVC	0.586±0.032	0.611±0.035	0.607±0.037	0.600±0.034
B-ICB-S	0.740±0.059	0.742±0.089	0.741±0.074	0.752±0.076	B-ICB-S	0.571±0.021	0.589±0.022	0.588±0.021	0.584±0.023
OSIRL-CB	0.791±0.005	0.793±0.010	0.792±0.006	0.794±0.005	OSIRL-CB	0.591±0.028	0.613±0.029	0.608±0.034	0.602±0.037
<b>IBCB (Ours)</b>	<b>0.963±0.012</b>	<b>0.958±0.010</b>	<b>0.958±0.006</b>	<b>0.958±0.005</b>	<b>IBCB (Ours)</b>	<b>0.651±0.027</b>	<b>0.667±0.027</b>	<b>0.666±0.026</b>	<b>0.658±0.029</b>

TABLE 3

Reward parameters' estimation, Train time comparison and Expert policy parameters' estimation on ML-100K dataset. Experts do not need to train since they produce train logs (behavioral evolution history logs) for baselines and IBCB. BaseSVC cannot obtain reward parameters, so it is excluded from OL-Fitness comparison. Expert policy produces same actions compared with itself, so BT-Fitness of expert policy is 1.

Algorithm	OL-Fitness	Train Time (sec.)	BT-Fitness	BT-AR	Algorithm	OL-Fitness	Train Time (sec.)	BT-Fitness	BT-AR
SBUCB	0.891±0.006	/	1	0.193±0.005	SBTS	0.868±0.015	/	1	0.200±0.005
B-IRL	0.830±0.010	43.669±1.282	0.864±0.008	0.189±0.005	B-IRL	0.851±0.015	47.005±4.311	0.870±0.030	0.199±0.007
BaseSVC	/	162.767±12.135	0.804±0.012	0.189±0.005	BaseSVC	/	192.83±33.199	0.801±0.019	0.198±0.007
B-ICB-S	0.746±0.032	147.474±4.633	0.767±0.028	0.187±0.007	B-ICB-S	0.721±0.030	145.441±5.916	0.759±0.035	0.196±0.004
OSIRL-CB	0.838±0.011	115.720±8.137	0.874±0.004	0.189±0.008	OSIRL-CB	0.846±0.013	103.850±7.752	0.889±0.022	0.200±0.006
<b>IBCB (Ours)</b>	<b>0.862±0.023</b>	<b>4.036±0.111</b>	<b>0.925±0.019</b>	<b>0.192±0.005</b>	<b>IBCB (Ours)</b>	<b>0.869±0.021</b>	<b>8.830±1.825</b>	<b>0.956±0.006</b>	<b>0.201±0.005</b>

### 5.5.3 Memory Usage Comparison on both Datasets

Table 4 shows the memory usage comparison between IBCB and all baselines. As the results indicate, the memory footprint of IBCB is comparable to that of the baseline methods. This characteristic further substantiates the spatial efficiency of our IBCB framework.

### 5.5.4 Experiments on Fairness-Aware Expert Limitation

In the original fairness-aware experiment, we set the **top-k** with  $k = 2$ , which means we choose the final action from the **top-2** based on normalized probabilities.

TABLE 4

**Memory usage comparison** of different baselines and IBCB on synthetic and ML-100K dataset. All results were averaged over 5 different runs, and standard deviations of each result are provided. Memory usage is evaluated with MB size.

Dataset	BaseSVC	B-IRL	B-ICB-S	OSIRL-CB	IBCB
Synthetic	48.077±0.863	99.349±1.270	201.919±1.752	165.558±1.346	83.151±0.945
ML-100K	56.195±1.228	175.652±1.376	256.723±2.119	215.832±1.304	88.316±1.271

The experimental results in Table 5 and Table 6 demonstrate that IBCB effectively learns expert parameters in

TABLE 5

Fairness comparison (**Upper Subtable**), Fairness-aware train time comparison (**Lower Subtable**) on synthetic dataset (CFR definition can be found in Section 5.3.4). Expert selects action from **top-2** actions with proportion of its learned probability distribution through all actions. Expert is defined as the fairest policy from the CFR metric, so its CFR is always 0. Fairness-aware experts (SBUCB & SBTS) do not need to train since they produce the train logs (behavioral evolution history logs) for baselines and IBCB.

Algorithm	Cumulative Fairness Regret (CFR)				Algorithm	Cumulative Fairness Regret (CFR)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	0	0	0	0	SBTS	0	0	0	0
B-IRL	2175.0±159.1	2330.0±235.8	2347.2±305.4	2449.0±344.1	B-IRL	2106.5±136.6	2080.7±236.3	2102.9±227.4	2136.9±192.1
BaseSVC	2027.5±184.2	2218.2±342.6	2248.0±402.2	2367.8±449.9	BaseSVC	1848.8±232.4	1776.9±427.0	1763.8±391.5	1841.2±344.7
B-ICB-S	2162.3±205.5	2661.7±169.6	2507.1±240.8	2437.6±175.9	B-ICB-S	2679.5±405.6	2922.5±451.3	2660.4±305.1	2800.2±197.7
OSIRL-CB	2180.2±159.2	2338.1±148.5	2371.0±308.4	2487.6±341.8	OSIRL-CB	2123.4±137.5	2090.4±236.7	2125.6±229.6	2229.5±201.4
<b>IBCB (ours)</b>	<b>1609.6±92.6</b>	<b>1475.8±127.2</b>	<b>1353.9±132.3</b>	<b>1409.8±205.8</b>	<b>IBCB (ours)</b>	<b>1263.2±104.3</b>	<b>1262.1±198.3</b>	<b>1287.0±183.3</b>	<b>1257.8±117.3</b>

Algorithm	Train Time (sec.)				Algorithm	Train Time (sec.)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	/	/	/	/	SBTS	/	/	/	/
B-IRL	41.466±2.898	42.802±10.953	40.348±2.888	42.361±4.737	B-IRL	38.332±4.622	35.703±0.101	35.867±0.287	39.226±4.344
BaseSVC	210.83±37.72	187.31±19.48	181.46±21.53	193.40±14.25	BaseSVC	152.30±13.81	147.73±15.27	146.71±8.414	145.46±7.936
B-ICB-S	90.922±4.373	89.637±4.892	88.908±2.650	90.525±2.571	B-ICB-S	85.615±1.072	85.511±2.658	84.919±1.306	86.802±1.530
OSIRL-CB	51.642±1.382	57.146±10.549	54.592±6.787	53.012±2.877	OSIRL-CB	54.020±3.691	54.914±7.961	50.914±2.019	51.093±1.364
<b>IBCB (Ours)</b>	<b>1.370±0.238</b>	<b>1.308±0.198</b>	<b>1.403±0.243</b>	<b>1.191±0.175</b>	<b>IBCB (Ours)</b>	<b>0.889±0.015</b>	<b>0.916±0.028</b>	<b>0.898±0.020</b>	<b>0.937±0.079</b>

TABLE 6

Fairness-aware expert policy (SBUCB) parameters' estimation (**Upper Subtable**) and Fairness-aware expert policy (SBTS) parameters' estimation (**Lower Subtable**) on synthetic dataset. Expert selects action from **top-2** actions with proportion of its learned probability distribution through all actions. Expert policy produces same actions compared with itself, so Batch Test Log Fitness of expert policy is 1.

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1	SBUCB	0.589±0.008	0.583±0.030	0.579±0.029	0.568±0.032
B-IRL	0.445±0.003	0.431±0.007	0.432±0.010	0.430±0.014	B-IRL	0.536±0.012	0.524±0.039	0.519±0.040	0.504±0.045
BaseSVC	0.446±0.004	0.439±0.009	0.439±0.014	0.434±0.015	BaseSVC	0.530±0.013	0.516±0.041	0.511±0.042	0.495±0.047
B-ICB-S	0.431±0.014	0.417±0.008	0.419±0.012	0.421±0.007	B-ICB-S	0.543±0.005	0.512±0.025	0.515±0.024	0.508±0.027
OSIRL-CB	0.447±0.007	0.440±0.008	0.435±0.018	0.425±0.020	OSIRL-CB	0.570±0.013	0.555±0.045	0.549±0.047	0.533±0.052
<b>IBCB (Ours)</b>	<b>0.455±0.003</b>	<b>0.461±0.008</b>	<b>0.464±0.008</b>	<b>0.464±0.012</b>	<b>IBCB (Ours)</b>	<b>0.609±0.011</b>	<b>0.600±0.025</b>	<b>0.591±0.030</b>	<b>0.585±0.033</b>

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBTS	1	1	1	1	SBTS	0.608±0.023	0.614±0.033	0.612±0.029	0.606±0.027
B-IRL	0.448±0.005	0.444±0.009	0.444±0.006	0.442±0.007	B-IRL	0.558±0.029	0.565±0.044	0.563±0.039	0.555±0.035
BaseSVC	0.451±0.007	0.452±0.007	0.449±0.005	0.449±0.008	BaseSVC	0.553±0.030	0.562±0.047	0.560±0.042	0.552±0.037
B-ICB-S	0.409±0.023	0.396±0.027	0.406±0.017	0.402±0.013	B-ICB-S	0.536±0.022	0.530±0.018	0.541±0.016	0.529±0.022
OSIRL-CB	0.450±0.007	0.448±0.009	0.447±0.006	0.445±0.009	OSIRL-CB	0.596±0.033	0.603±0.048	0.602±0.044	0.591±0.040
<b>IBCB (Ours)</b>	<b>0.472±0.006</b>	<b>0.473±0.008</b>	<b>0.473±0.007</b>	<b>0.476±0.003</b>	<b>IBCB (Ours)</b>	<b>0.617±0.024</b>	<b>0.622±0.035</b>	<b>0.621±0.033</b>	<b>0.616±0.028</b>

complex, fairness-oriented recommendation settings. With a significantly lower CFR compared to the baseline and well-managed training time, IBCB achieves superior BT-Fitness and BT-AR, showing its capacity to uphold high recommendation quality while ensuring fairness. IBCB’s exceptional performance indicates its robust generalization in learning fair expert parameters and its framework’s adaptability in addressing BCB-related challenges.

**5.6 Ablation Study**

In this section, we explore extreme scenarios faced by imitation learning (IL) algorithms and IBCB, including delayed updates to training logs, which provides only partial access to the expert’s behavioral history (e.g., the first half of the data), and the emergence of out-of-distribution or contradictory data. Since hyperparameter  $\alpha$  of the original expert policy is inaccessible to IBCB, we also examine the impact of different  $\alpha$  settings in IBCB. Ablation Study’s results on synthetic dataset are detailed in Section 5.6.1–Section 5.6.5. Also, comprehensive results and analysis of

IBCB’s hyperparameter tuning and ablation study on ML-100K dataset are detailed in Section 5.6.6 and Section 5.6.7.

**5.6.1 Out-of-distribution Data Experiments on Synthetic Dataset**

As shown in Table 7, IBCB still outperforms other IL baselines in expert policy’s parameter estimation on out-of-distribution (OOD) data.

**5.6.2 Limited Training logs of Experts on Synthetic Dataset**

From the results in Table 8 and Table 9 we find that, when all IL baselines and IBCB only have the access to the first half of expert’s behavioral evolution history logs, i.e., *novice expert’s evolution history*, IBCB can have better performance, compared with the degree of decline in all algorithms from Table 1 and Table 2.

TABLE 7

Expert policy (SBUCB) parameters' estimation (**Upper Subtable**) and Expert policy (SBTS) parameters' estimation (**Lower Subtable**) for **Out-of-distribution data** on synthetic dataset. Expert policy produces same actions compared with itself, so BT-Fitness of expert policy is 1.

Batch Test Log Fitness (OOD)					Batch Test Average Reward (OOD)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1	SBUCB	0.709±0.015	0.695±0.036	0.694±0.031	0.684±0.030
B-IRL	0.802±0.005	0.793±0.013	0.795±0.015	0.793±0.016	B-IRL	0.642±0.019	0.622±0.047	0.622±0.042	0.608±0.041
BaseSVC	0.764±0.009	0.756±0.021	0.756±0.022	0.750±0.023	BaseSVC	0.627±0.021	0.607±0.052	0.606±0.046	0.591±0.046
B-ICB-S	0.775±0.039	0.738±0.065	0.755±0.050	0.754±0.025	B-ICB-S	0.632±0.008	0.600±0.030	0.607±0.032	0.596±0.032
<b>ICB (Ours)</b>	<b>0.970±0.012</b>	<b>0.960±0.036</b>	<b>0.959±0.034</b>	<b>0.969±0.02</b>	<b>ICB (Ours)</b>	<b>0.717±0.017</b>	<b>0.707±0.034</b>	<b>0.706±0.024</b>	<b>0.693±0.028</b>

Batch Test Log Fitness (OOD)					Batch Test Average Reward (OOD)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10	Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBTS	1	1	1	1	SBTS	0.719±0.028	0.739±0.032	0.735±0.032	0.729±0.032
B-IRL	0.792±0.005	0.795±0.010	0.793±0.006	0.793±0.005	B-IRL	0.650±0.034	0.676±0.037	0.671±0.039	0.663±0.037
BaseSVC	0.767±0.015	0.782±0.017	0.780±0.019	0.776±0.014	BaseSVC	0.640±0.041	0.670±0.045	0.666±0.047	0.658±0.044
B-ICB-S	0.714±0.065	0.710±0.096	0.712±0.081	0.724±0.086	B-ICB-S	0.620±0.028	0.643±0.028	0.642±0.026	0.637±0.027
<b>ICB (Ours)</b>	<b>0.965±0.013</b>	<b>0.962±0.009</b>	<b>0.961±0.003</b>	<b>0.962±0.005</b>	<b>ICB (Ours)</b>	<b>0.722±0.035</b>	<b>0.744±0.034</b>	<b>0.743±0.033</b>	<b>0.732±0.038</b>

TABLE 8

Reward parameters' estimation on synthetic dataset (using **first half** of expert's behavior evolution history logs). BaseSVC cannot obtain reward parameters, so it was excluded from OL-Fitness comparison.

Online Train Log Fitness (Half Train)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	0.883±0.007	0.879±0.016	0.878±0.018	0.874±0.019
B-IRL	0.752±0.018	0.729±0.051	0.728±0.050	0.721±0.047
B-ICB-S	0.730±0.039	0.727±0.045	0.721±0.063	0.717±0.041
<b>ICB (Ours)</b>	<b>0.895±0.011</b>	<b>0.894±0.016</b>	<b>0.892±0.024</b>	<b>0.886±0.017</b>

TABLE 9

Expert policy parameters' estimation on synthetic dataset (using **first half** of expert's behavioral evolution history logs). Expert policy produces same actions compared with itself, so BT-Fitness of expert policy is 1.

Batch Test Log Fitness (Half Train)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1
B-IRL	0.718±0.012	0.699±0.031	0.702±0.035	0.694±0.037
BaseSVC	0.656±0.019	0.638±0.039	0.638±0.043	0.627±0.045
B-ICB-S	0.695±0.054	0.707±0.069	0.695±0.084	0.688±0.043
<b>ICB (Ours)</b>	<b>0.891±0.019</b>	<b>0.919±0.030</b>	<b>0.890±0.031</b>	<b>0.890±0.023</b>

Batch Test Average Reward (Half Train)				
Algorithm	std=0	std=0.03	std=0.07	std=0.10
SBUCB	0.641±0.010	0.629±0.028	0.630±0.024	0.622±0.024
B-IRL	0.545±0.020	0.522±0.047	0.522±0.046	0.510±0.046
BaseSVC	0.520±0.023	0.497±0.053	0.497±0.052	0.482±0.051
B-ICB-S	0.549±0.019	0.538±0.024	0.540±0.030	0.529±0.033
<b>ICB (Ours)</b>	<b>0.610±0.015</b>	<b>0.608±0.025</b>	<b>0.597±0.036</b>	<b>0.588±0.030</b>

### 5.6.3 Extra Fairness-aware Expert Experiments on Synthetic Dataset

We also conducted fairness-aware expert experiments with different fairness settings on synthetic dataset. Specifically, in the original experiment, we set top- $k$  fairness with  $k = 2$  (Table 5-Table 6). We also make experiments with  $k = 3, 4$  to discover the impact of changing fairness parameter (Table 10-Table 13). It can be observed from the results that when  $k$  increases, BC methods' training time significantly increase, indicating that BC methods tend to be confused because observed actions made by the expert are drawn from more possible candidates, i.e. the top- $k$  set. When  $k$  increased, the BT-Fitness performance of all models decreased

significantly, and the CFR also increased rapidly, which is related to the more random action selection of experts. However, ICB can still quickly learn expert parameters and get the best performance in this situation, which also shows that ICB still has a very high generalization ability in complex scenarios of fairness.

### 5.6.4 Detailed Results of Contradictory Data Testing on Synthetic Dataset

Note that BaseSVC cannot obtain reward parameters, and thus were not included in OL-Fitness. We used 'Train Log Behaviors Fitness' to evaluate whether BaseSVC suffers from contradictory data and make comparison with ICB and other baselines. As shown in Table 14, contradictory data (same context in different time periods) undermines BaseSVC's train log behaviors fitness, but ICB stayed stable in OL-Fitness. That is, Contradictory data significantly affects BaseSVC's fitness but does not impact models that learn reward parameters like ICB, highlighting ICB's robustness in extreme conditions.

### 5.6.5 ICB's Parameter Test on Synthetic Dataset

'ICB- $\alpha$ - $k$ ' denotes ICB's hyper parameter (in Eq.(17)) is  $k$ . As shown in Table 16 and Table 17, ICB's hyper parameter  $\alpha$  will influence several indicators, especially the train time, due to the OSQP tolerance level will be relaxed if  $\alpha$  is above 1.5. So for ICB of large  $\alpha$  with more relaxed tolerance, i.e., lower accuracy, training will be faster. Meanwhile, the performance of expert policy's parameter estimation is similar according to BT-Fitness and BT-AR. By examining the parameters learned by ICB with different  $\alpha$  values, we found that after normalization (by dividing each parameter's  $l_2$ -norm), the sum of absolute differences in every dimension of the parameters from each pair of two distinct parameters is tightly small. This suggests that one of these parameters can be a result of scaling up or down from another. During BT-data testing, items are recommended based on the parameter under BCB setting without exploration. Therefore, scaling up or down does not influence the recommended item since  $\arg \max_{I \in \mathcal{I}} (\hat{\theta}, s_I)$  is only determined by the item's content  $s_I$  when  $\hat{\theta}$  is fixed with a scaling factor. On the synthetic dataset, the item's

TABLE 10

Cumulative Fairness Regret (CFR) comparison (**Upper Subtable**) and Train time comparison (**Lower Subtable**) on synthetic dataset. Expert selects action from **top-3** actions with proportion of its learned probability distribution through all actions. Expert is defined as the fairest policy from the CFR metric, so its CFR is always 0. Fairness-aware experts (SBUCB & SBTS) do not need to train since they produce the train logs (behavioral evolution history logs) for baselines and IBCB.

Algorithm	Cumulative Fairness Regret				Algorithm	Cumulative Fairness Regret			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	0	0	0	0	SBTS	0	0	0	0
B-IRL	2713.9±230.2	2882.3±356.2	2841.7±356.4	2982.029±370.4	B-IRL	2704.2±300.9	2433.9±285.2	2498.8±248.0	2488.4±226.6
BaseSVC	2465.2±257.4	2653.5±412.5	2608.6±447.9	2758.4±421.7	BaseSVC	2395.3±346.8	2131.7±376.1	2166.1±291.1	2170.2±299.2
B-ICB-S	2637.6±168.5	2747.3±260.1	2805.3±398.6	2631.2±485.6	B-ICB-S	2856.4±439.8	3281.5±162.1	3136.9±187.3	2980.8±350.3
<b>IBCB (ours)</b>	<b>2232.6±148.6</b>	<b>2071.0±264.4</b>	<b>2161.9±124.4</b>	<b>2192.0±110.0</b>	<b>IBCB (ours)</b>	<b>1596.7±172.3</b>	<b>1704.7±158.6</b>	<b>1584.4±124.1</b>	<b>1705.5±176.8</b>

Algorithm	Train Time (sec.)				Algorithm	Train Time (sec.)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	/	/	/	/	SBTS	/	/	/	/
B-IRL	40.758±4.040	35.796±0.352	38.59±4.594	41.948±7.858	B-IRL	35.664±0.208	37.07±2.135	39.162±6.970	35.662±0.081
BaseSVC	234.56±13.51	228.80±19.52	231.25±31.15	249.73±31.44	BaseSVC	209.01±10.75	221.06±18.02	215.58±18.12	230.40±25.84
B-ICB-S	85.977±1.975	87.631±4.364	86.164±1.884	84.881±0.493	B-ICB-S	86.726±1.870	87.447±1.977	86.796±1.808	85.974±1.826
<b>IBCB (ours)</b>	<b>1.169±0.225</b>	<b>1.102±0.084</b>	<b>1.075±0.048</b>	<b>1.055±0.031</b>	<b>IBCB (ours)</b>	<b>0.798±0.020</b>	<b>0.786±0.012</b>	<b>0.791±0.013</b>	<b>0.849±0.116</b>

TABLE 11

Fairness-aware expert policy (SBUCB) parameters' estimation (**Upper Subtable**) and Fairness-aware expert policy (SBTS) parameters' estimation (**Lower Subtable**) on synthetic dataset. Expert selects action from **top-3** actions with proportion of its learned probability distribution through all actions. Expert policy produces same actions compared with itself, so Batch Test Log Fitness of expert policy is 1.

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1	SBUCB	0.538±0.012	0.529±0.030	0.528±0.031	0.516±0.031
B-IRL	0.307±0.005	0.297±0.008	0.300±0.008	0.296±0.007	B-IRL	0.486±0.018	0.472±0.039	0.472±0.041	0.457±0.042
BaseSVC	0.306±0.006	0.302±0.01	0.305±0.009	0.303±0.009	BaseSVC	0.481±0.021	0.467±0.041	0.467±0.044	0.451±0.043
B-ICB-S	0.300±0.006	0.296±0.006	0.295±0.008	0.296±0.011	B-ICB-S	0.500±0.004	0.484±0.025	0.482±0.029	0.475±0.026
<b>IBCB (ours)</b>	<b>0.307±0.004</b>	<b>0.310±0.006</b>	<b>0.309±0.004</b>	<b>0.309±0.004</b>	<b>IBCB (ours)</b>	<b>0.558±0.010</b>	<b>0.543±0.029</b>	<b>0.545±0.027</b>	<b>0.535±0.029</b>

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBTS	1	1	1	1	SBTS	0.548±0.028	0.566±0.030	0.560±0.025	0.556±0.024
B-IRL	0.302±0.006	0.309±0.003	0.305±0.002	0.307±0.002	B-IRL	0.495±0.039	0.521±0.037	0.512±0.030	0.508±0.028
BaseSVC	0.308±0.007	0.311±0.005	0.312±0.006	0.312±0.004	BaseSVC	0.492±0.039	0.518±0.039	0.510±0.032	0.507±0.031
B-ICB-S	0.292±0.011	0.281±0.008	0.287±0.004	0.292±0.009	B-ICB-S	0.501±0.014	0.493±0.021	0.496±0.016	0.496±0.017
<b>IBCB (ours)</b>	<b>0.321±0.004</b>	<b>0.319±0.004</b>	<b>0.322±0.003</b>	<b>0.318±0.004</b>	<b>IBCB (ours)</b>	<b>0.547±0.031</b>	<b>0.568±0.031</b>	<b>0.559±0.026</b>	<b>0.559±0.022</b>

TABLE 12

Cumulative Fairness Regret (CFR) comparison (**Upper Subtable**) and Train time comparison (**Lower Subtable**) on synthetic dataset. Expert selects action from **top-4** actions with proportion of its learned probability distribution through all actions. Expert is defined as the fairest policy from the CFR metric, so its CFR is always 0. Fairness-aware experts (SBUCB & SBTS) do not need to train since they produce the train logs (behavioral evolution history logs) for baselines and IBCB.

Algorithm	Cumulative Fairness Regret				Algorithm	Cumulative Fairness Regret			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	0	0	0	0	SBTS	0	0	0	0
B-IRL	3063.9±258.1	3279.7±364.6	3214.4±381.0	3263.6±286.0	B-IRL	2935.1±407.0	2812.5±254.6	2790.2±260.6	2794.1±319.6
BaseSVC	<b>2893.6±240.7</b>	3086.1±401.7	3042.2±399.8	3121.7±302.5	BaseSVC	2752.0±379.1	2597.6±225.7	2617.2±273.2	2588.8±261.7
B-ICB-S	2998.0±153.3	3109.8±364.9	3123.4±287.4	3018.3±165.1	B-ICB-S	3295.3±357.9	3092.2±240.2	3369.5±59.5	3093.9±388.0
<b>IBCB (ours)</b>	<b>3130.6±194.2</b>	<b>2794.3±282.2</b>	<b>2966.9±205.4</b>	<b>2882.1±238.9</b>	<b>IBCB (ours)</b>	<b>2162.4±129.4</b>	<b>2117.3±43.8</b>	<b>1918.0±128.6</b>	<b>1990.1±232.6</b>

Algorithm	Train Time (sec.)				Algorithm	Train Time (sec.)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	/	/	/	/	SBTS	/	/	/	/
B-IRL	36.867±0.571	43.7±6.185	40.665±6.652	37.241±0.991	B-IRL	36.697±0.373	41.568±4.424	39.308±5.713	36.270±0.706
BaseSVC	330.68±22.03	343.56±37.87	349.05±37.32	365.68±7.47	BaseSVC	295.92±23.73	327.44±37.10	314.77±19.73	309.99±14.98
B-ICB-S	86.362±1.885	84.884±0.443	86.48±0.594	86.488±1.541	B-ICB-S	88.003±2.241	86.189±2.22	86.318±1.266	87.87±1.607
<b>IBCB (ours)</b>	<b>1.052±0.052</b>	<b>1.016±0.037</b>	<b>1.135±0.211</b>	<b>1.034±0.032</b>	<b>IBCB (ours)</b>	<b>0.950±0.175</b>	<b>0.737±0.013</b>	<b>0.838±0.148</b>	<b>0.820±0.112</b>

TABLE 13

Fairness-aware expert policy (SBUCB) parameters' estimation (**Upper Subtable**) and Fairness-aware expert policy (SBTS) parameters' estimation (**Lower Subtable**) on synthetic dataset. Expert selects action from **top-4** actions with proportion of its learned probability distribution through all actions. Expert policy produces same actions compared with itself, so Batch Test Log Fitness of expert policy is 1.

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBUCB	1	1	1	1	SBUCB	0.490±0.01	0.480±0.029	0.478±0.029	0.471±0.028
B-IRL	<b>0.236±0.003</b>	0.229±0.004	0.229±0.005	0.230±0.005	B-IRL	0.444±0.014	0.429±0.035	0.428±0.038	0.421±0.031
BaseSVC	0.235±0.003	<b>0.232±0.007</b>	<b>0.232±0.005</b>	<b>0.233±0.006</b>	BaseSVC	0.441±0.016	0.424±0.038	0.423±0.038	0.414±0.032
B-ICB-S	0.228±0.003	0.226±0.007	0.227±0.007	0.227±0.004	B-ICB-S	0.456±0.007	0.450±0.024	0.443±0.024	0.442±0.022
<b>ICB (ours)</b>	0.227±0.003	0.231±0.007	0.230±0.003	0.230±0.004	<b>ICB (ours)</b>	<b>0.523±0.005</b>	<b>0.499±0.024</b>	<b>0.502±0.024</b>	<b>0.495±0.024</b>

Algorithm	Batch Test Log Fitness				Algorithm	Batch Test Average Reward			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
SBTS	1	1	1	1	SBTS	0.500±0.024	0.510±0.021	0.510±0.022	0.503±0.026
B-IRL	0.234±0.006	0.234±0.003	0.235±0.002	0.233±0.002	B-IRL	0.455±0.033	0.466±0.026	0.465±0.026	0.460±0.032
BaseSVC	0.235±0.004	0.238±0.005	0.235±0.003	0.235±0.004	BaseSVC	0.453±0.034	0.466±0.027	0.464±0.028	0.458±0.032
B-ICB-S	0.226±0.006	0.229±0.007	0.222±0.007	0.228±0.009	B-ICB-S	0.451±0.016	0.465±0.021	0.462±0.016	0.458±0.018
<b>ICB (ours)</b>	<b>0.239±0.004</b>	<b>0.241±0.002</b>	<b>0.242±0.004</b>	<b>0.241±0.003</b>	<b>ICB (ours)</b>	<b>0.496±0.024</b>	<b>0.507±0.020</b>	<b>0.500±0.016</b>	<b>0.499±0.032</b>

TABLE 14

Contradictory (duplicated) data (appeared at expert's OL-data behavioral evolution history logs) on synthetic dataset testing. dup denotes the quantity of duplicated episodes in OL-data phase. This table shows the OL-Fitness results.

Algorithm	Online Train Log Fitness									
	dup=0	dup=1	dup=2	dup=3	dup=4	dup=5	dup=6	dup=7	dup=8	dup=9
SBUCB	0.879±0.016	0.880±0.015	0.880±0.015	0.879±0.015	0.880±0.018	0.879±0.015	0.880±0.016	0.879±0.015	0.879±0.015	0.878±0.017
B-IRL	0.808±0.039	0.808±0.037	0.808±0.038	0.806±0.038	0.807±0.038	0.807±0.037	0.809±0.037	0.810±0.037	0.809±0.037	0.808±0.037
<b>ICB (Ours)</b>	<b>0.858±0.034</b>	<b>0.860±0.021</b>	<b>0.855±0.034</b>	<b>0.865±0.021</b>	<b>0.868±0.005</b>	<b>0.874±0.003</b>	<b>0.875±0.006</b>	<b>0.848±0.038</b>	<b>0.871±0.006</b>	<b>0.867±0.008</b>

Algorithm	Train Log Behaviors Fitness (BC)									
	dup=0	dup=1	dup=2	dup=3	dup=4	dup=5	dup=6	dup=7	dup=8	dup=9
BaseSVC	0.823±0.026	0.825±0.025	0.824±0.026	0.821±0.026	0.820±0.026	0.819±0.025	0.819±0.025	0.819±0.024	0.817±0.023	0.816±0.023

TABLE 15

Contradictory (duplicated) data (appeared at expert's OL-data behavioral evolution history logs) on synthetic dataset testing. dup denotes the quantity of duplicated episodes in OL-data phase. This table shows the BT-Fitness results.

Algorithm	Batch Test Log Fitness									
	dup=0	dup=1	dup=2	dup=3	dup=4	dup=5	dup=6	dup=7	dup=8	dup=9
SBUCB	1	1	1	1	1	1	1	1	1	1
B-IRL	0.820±0.013	0.820±0.012	0.820±0.013	0.820±0.013	0.819±0.014	0.818±0.014	0.819±0.014	0.818±0.011	0.818±0.011	0.819±0.014
BaseSVC	0.781±0.020	0.783±0.019	0.782±0.020	0.781±0.018	0.780±0.021	0.780±0.020	0.779±0.020	0.778±0.018	0.776±0.018	0.776±0.021
<b>ICB (Ours)</b>	<b>0.963±0.029</b>	<b>0.964±0.015</b>	<b>0.960±0.027</b>	<b>0.967±0.017</b>	<b>0.967±0.013</b>	<b>0.972±0.015</b>	<b>0.975±0.014</b>	<b>0.953±0.034</b>	<b>0.973±0.012</b>	<b>0.973±0.013</b>

TABLE 16

Reward parameters' estimation (Left) and training time (Right) comparison on synthetic dataset (ICB's parameter test). Experts do not need train since they produce train logs (behavioral evolution history logs) for baselines and ICB. BaseSVC cannot obtain reward parameters, so it was excluded from OL-Fitness comparison. Original Expert is SBUCB.

Algorithm	Online Train Log Fitness (Param Test)				Algorithm	Train Time(sec.) (Param Test)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
ICB- $\alpha$ -1	0.868±0.008	0.858±0.036	0.859±0.038	0.861±0.029	ICB- $\alpha$ -1	0.815±0.045	0.999±0.128	0.866±0.137	1.016±0.109
ICB- $\alpha$ -0.6	<b>0.880±0.022</b>	0.876±0.006	0.866±0.012	0.848±0.050	ICB- $\alpha$ -0.6	0.907±0.109	0.971±0.092	0.904±0.106	0.867±0.121
ICB- $\alpha$ -0.8	<b>0.880±0.022</b>	0.876±0.006	0.866±0.012	0.848±0.050	ICB- $\alpha$ -0.8	0.958±0.134	0.880±0.097	0.868±0.086	0.961±0.119
ICB- $\alpha$ -1.2	0.867±0.005	0.870±0.007	0.869±0.010	0.862±0.009	ICB- $\alpha$ -1.2	0.725±0.037	0.776±0.036	0.730±0.025	0.772±0.041
ICB- $\alpha$ -1.5	0.879±0.012	<b>0.880±0.013</b>	<b>0.870±0.018</b>	<b>0.874±0.014</b>	ICB- $\alpha$ -1.5	<b>0.462±0.052</b>	<b>0.467±0.054</b>	<b>0.469±0.044</b>	<b>0.468±0.046</b>
ICB- $\alpha$ -2	0.865±0.007	0.865±0.012	0.864±0.016	0.861±0.008	ICB- $\alpha$ -2	0.574±0.007	0.582±0.017	0.557±0.052	0.538±0.070

content  $s_I$  is very easy to separate (interval difference of each dimension of  $s_I$  is very obvious through the setting in Section 5.2.1), so two distinct parameters with very small sum of absolute differences may share similar BT-Fitness and BT-AR.

### 5.6.6 Ablation Study on ML-100K Dataset

For ML-100K dataset, Table 18 and Table 19 reports the result of different ablation studies. Results are similar to those on synthetic dataset in most indicators: ICB can learn from novice expert's evolution history with better performance compared with other baselines. Note that mean and variance of train time for ICB- $\alpha$ -1.2 is larger than any other  $\alpha$  settings. From the OSQP train record, we analysed that

TABLE 17

Expert policy parameters' estimation on synthetic dataset (ICB's parameter test). Expert policy produces same actions compared with itself, so BT-Fitness of expert policy is 1. Original Expert is SBUCB.

Algorithm	Batch Test Log Fitness (Param Test)				Algorithm	Batch Test Average Reward (Param Test)			
	std=0	std=0.03	std=0.07	std=0.10		std=0	std=0.03	std=0.07	std=0.10
ICB- $\alpha$ -1	0.970±0.012	0.960±0.036	0.959±0.034	<b>0.969±0.020</b>	ICB- $\alpha$ -1	0.647±0.012	0.638±0.026	0.639±0.019	0.628±0.022
ICB- $\alpha$ -0.6	0.964±0.022	0.966±0.017	0.950±0.015	0.933±0.047	ICB- $\alpha$ -0.6	0.644±0.013	0.634±0.032	0.639±0.03	<b>0.637±0.018</b>
ICB- $\alpha$ -0.8	0.969±0.017	0.954±0.032	0.937±0.042	0.930±0.041	ICB- $\alpha$ -0.8	0.647±0.012	<b>0.640±0.025</b>	<b>0.644±0.02</b>	0.635±0.020
ICB- $\alpha$ -1.2	<b>0.972±0.006</b>	0.971±0.012	<b>0.970±0.005</b>	0.967±0.013	ICB- $\alpha$ -1.2	0.647±0.012	0.632±0.032	0.633±0.029	0.625±0.030
ICB- $\alpha$ -1.5	0.959±0.011	0.960±0.012	0.943±0.015	0.956±0.020	ICB- $\alpha$ -1.5	<b>0.650±0.010</b>	0.637±0.029	0.643±0.020	0.631±0.025
ICB- $\alpha$ -2	0.970±0.006	<b>0.973±0.017</b>	0.962±0.018	0.960±0.021	ICB- $\alpha$ -2	0.647±0.011	0.633±0.031	0.638±0.020	0.629±0.025

TABLE 18

Reward & policy parameters' estimation on ML-100K dataset (using first half of expert's behavioral evolution history logs). Expert policy produces same actions compared with itself, so BT-Fitness of expert policy is 1. BaseSVC cannot obtain reward parameters, so it was excluded from OL-Fitness comparison.

Algorithm	OL-Fitness	BT-Fitness	BT-AR
SBUCB	0.891±0.006	1	0.193±0.005
B-IRL	0.789±0.017	0.784±0.022	0.189±0.004
BaseSVC	/	0.701±0.022	0.186±0.004
B-ICB-S	0.399±0.110	0.325±0.123	0.140±0.025
<b>ICB (Ours)</b>	<b>0.861±0.002</b>	<b>0.829±0.018</b>	<b>0.189±0.006</b>

TABLE 19

ICB's parameter test on ML-100K dataset. Experts do not need to train since they produce train logs (behavioral evolution history logs) for baselines and ICB.

Algorithm	OL-Fitness	BT-Fitness	BT-AR	Train Time (sec.)
ICB- $\alpha$ -1	<b>0.862±0.023</b>	0.925±0.019	0.192±0.005	4.036±0.111
ICB- $\alpha$ -0.6	0.775±0.007	0.861±0.016	0.192±0.005	3.648±0.139
ICB- $\alpha$ -0.8	0.860±0.029	0.921±0.021	0.191±0.006	4.026±0.194
ICB- $\alpha$ -1.2	0.861±0.013	<b>0.929±0.007</b>	0.191±0.006	4.210±0.121
ICB- $\alpha$ -1.5	0.858±0.014	0.855±0.012	<b>0.194±0.005</b>	<b>2.501±0.075</b>
ICB- $\alpha$ -2	0.845±0.017	0.855±0.012	<b>0.194±0.005</b>	2.663±0.221

this happened due to OSQP failed to convergent for some of the OL-data train logs, so for large  $\alpha$ , it is recommended to use more relaxed tolerance level in OSQP setting. Note that B-ICB-S's performance significantly decline when using first half of the train logs in Table 18, this is because when in first half of the real-world train logs, *novice expert* is still exploring with large steps in a batched manner rather than updating after each step or staying stable with little exploration, which B-ICB-S assumed.

5.6.7 Extra Ablation Study Experiments of Batch Test period on ML-100K Dataset

Furthermore, we conducted some extra ablation study experiments on ML-100K to test the real-world task's performance (e.g. train time, average reward in Batch Test period) improvements against other baselines.

Firstly, since we assume that training period can be described as a period containing  $N$  episodes' training data, we want to know how this  $N$ 's setting influence the performance of baselines. Specifically, we divided the total  $N$  episodes to individually train corresponding IRL or BC models. Finally, we averaged the parameters of the  $N$  models obtained from each episode's training to form the final test model. In this setup, the training time equates to the cumulative time for training  $N$  models (disregarding multi-threading, assuming that each time we collect data from

TABLE 20

Comparison made between original algorithms and baselines using average method when training on ML-100K dataset. **Upper Subtable** is the performance made by original ones, which use **whole** data to train. **Lower Subtable** is the performance made by algorithms using average method (only baselines, excluding SBUCB and ICB), which use every episode data training distinct models, and final model uses **simple average** method to average these models' parameters.

Algorithm	OL-Fitness	BT-Fitness	BT-AR	Train Time (sec.)
SBUCB	0.891±0.006	1	0.193±0.005	/
B-IRL	0.830±0.010	0.864±0.008	0.189±0.005	43.669±1.282
BaseSVC	/	0.804±0.012	0.189±0.005	162.767±12.135
B-ICB-S	0.746±0.032	0.767±0.028	0.187±0.007	147.474±4.633
<b>ICB (Ours)</b>	<b>0.862±0.023</b>	<b>0.925±0.019</b>	<b>0.192±0.005</b>	<b>4.036±0.111</b>

Algorithm	OL-Fitness	BT-Fitness	BT-AR	Train Time (sec.)
SBUCB	0.891±0.006	1	0.193±0.005	/
B-IRL	0.835±0.007	0.868±0.008	0.190±0.005	85.352±9.192
BaseSVC	/	0.748±0.009	0.189±0.004	20.619±0.377
B-ICB-S	0.336±0.076	0.262±0.064	0.129±0.019	214.288±2.558
<b>ICB (Ours)</b>	<b>0.862±0.023</b>	<b>0.925±0.019</b>	<b>0.192±0.005</b>	<b>4.036±0.111</b>

TABLE 21

Comparison made between larger batch size  $B$  ( $B = 2000$ ) and smaller batch size  $B$  ( $B = 500$ ) when training & testing on ML-100K dataset. **Upper Subtable** is the experiment results obtained when  $B = 2000$ . **Lower Subtable** is the experiment results obtained when  $B = 500$ .

Algorithm	OL-Fitness	BT-Fitness	BT-AR	Train Time (sec.)
SBUCB	0.896±0.001	1	0.194±0.005	/
B-IRL	0.816±0.023	0.866±0.014	0.192±0.005	52.485±1.241
BaseSVC	/	0.807±0.014	0.191±0.004	230.26±12.606
B-ICB-S	0.685±0.028	0.707±0.052	0.188±0.006	144.697±7.972
<b>ICB (Ours)</b>	<b>0.861±0.010</b>	<b>0.886±0.014</b>	<b>0.192±0.004</b>	<b>2.907±0.261</b>

Algorithm	OL-Fitness	BT-Fitness	BT-AR	Train Time (sec.)
SBUCB	0.890±0.012	1	0.193±0.006	/
B-IRL	<b>0.845±0.007</b>	<b>0.877±0.009</b>	0.190±0.006	65.646±1.378
BaseSVC	/	0.815±0.006	0.189±0.005	266.538±12.153
B-ICB-S	0.716±0.039	0.723±0.049	0.187±0.007	138.616±8.982
<b>ICB (Ours)</b>	0.835±0.023	0.833±0.019	<b>0.194±0.006</b>	<b>4.296±0.128</b>

an episode, we train a model sequentially). The experiment was conducted using the ML-100K dataset, as we aimed to investigate the practical feasibility. Detailed experimental results can be found in Table 20. From the experimental results, it is evident that training IRL or BC models separately for each episode does yield certain differences in comparison to the unified training approach. The B-IRL model demonstrates a slight improvement in performance (e.g. OL-Fitness), albeit with a significant increase in training time. This could be attributed to the necessity of restarting training each time due to the inability to iteratively utilize previously acquired parameters when new data is collected.

Of particular interest is the significant decline in performance for B-ICB. This discrepancy is likely linked to the inconsistency in retaining the original expert model’s (i.e. SBUCB) parameters within each episode, which contradicts B-ICB’s assumption of evolving data within every single step. Additionally, due to the necessity for repeated restarts during training, the overall training time becomes extended. As for BaseSVC, the reduced training sample size in each episode leads to notably shorter training time for individual runs (attributed to its near-quadratic time complexity). While this reduces overall training time, the efficacy is comparatively diminished compared to the direct training using the entire dataset. Ultimately, even in comparison with the models subjected to parameter averaging, the performance of IBCB remains superior overall. This underscores the effectiveness of using episodes as evolutionary segments in IBCB’s training process in real-world datasets.

Secondly, we wanted to work out the influence of batch size  $B$ ’s setting. Throughout this ablation study experiment, we controlled the total number of training and testing samples and fixed the context, ensuring that the product of the total number of episodes  $N$  and the batch size  $B$  within each episode remains constant. Based on this foundation, we adjusted the batch size  $B$ . Specifically, the original values were  $N = 20$  and  $B = 1000$ . Extra experiments included scenarios with larger  $B$ , denoted as  $N = 10$  and  $B = 2000$ , with corresponding results displayed in upper subtable of Table 21. Likewise, there were scenarios with smaller  $B$ , represented as  $N = 40$  and  $B = 500$ , with results exhibited in lower subtable of Table 21. From the results of these additional experiments and comparisons with the original experiments, we deduced that, while controlling the total number of training and testing samples and context, a larger  $B$  yields shorter overall training time and better fitness on both environment and expert algorithm’s parameters. Therefore, the fitting to the environment and expert is reduced due to the increased value of  $N$ , implying that more data for training comprises novice expert’s evolutionary history, hence fostering more exploration. With reduced  $N$  and increased  $B$ , the frequency of evolution decreases, making adaptation to the original training data easier, thereby shortening training time. We found that the smaller  $B$ , the larger the total number of episodes  $N$ , but the extent of subsequent episode exploration gradually becomes smaller, so IBCB may tend to overfit. On the contrary, when the  $B$  is larger, the smaller the total number of episodes  $N$ , so the environment will encourage more exploration, and IBCB can fit better environment and expert algorithm’s parameters. In such cases, IBCB shows preciser performance against other baselines under larger  $B$ . Appropriately crafting exploration items can result in higher rewards.

5.6.8 Extra Experiments for Fully Online Scenarios

We have conducted new experiments on the synthetic dataset to evaluate IBCB and all baselines in a fully online setting, where the batch size  $B = 1$ . In this scenario, the expert updates their policy after every single episode ( $N = 2000, B = 1$ ). We compared IBCB against all existing baselines under this condition, and the results are shown in Table 22.

TABLE 22

**Fully online update** policy experiment on ML-100K dataset. Expert updates its policy after single decision. Episodes is set to be  $N = 2000$ , and batch size is set to be  $B = 1$ . All results were averaged over 5 different runs, and standard deviations of each result are provided. ‘↑’: larger is better; ‘↓’: smaller is better. bold: best results except for the base models. Note that in this setting, sampling strategy is not applied to B-ICB because its assumption that expert update after each state is satisfied, so training B-ICB cost extensive time.

Algorithm	OL-Fitness ↑	BT-Fitness ↑	BT-AR ↑	Train Time (sec.) ↓
SBUCB	0.731±0.036	1	0.183±0.017	/
B-IRL	0.722±0.023	0.757±0.056	0.175±0.021	6.669±0.138
BaseSVC	/	0.476±0.026	0.154±0.015	6.366±0.095
B-ICB	0.726±0.030	0.804±0.022	0.178±0.033	58704.142±1772.335
OSIRL-CB	0.724±0.025	0.763±0.044	0.176±0.025	8.127±0.281
<b>IBCB (Ours)</b>	<b>0.731±0.038</b>	<b>0.888±0.021</b>	<b>0.182±0.017</b>	<b>0.959±0.025</b>

The results show that IBCB continues to deliver the best performance, even in this fully online scenario. This confirms that IBCB is not limited to batch-updating experts and can effectively handle online policy updates, which further validates the robustness of our incremental update mechanism. We also noted that the performance of B-ICB improved significantly, which is expected, as its assumption of a policy update after every instance aligns perfectly with the fully online setting. However, training B-ICB cost extensive time to coverage, which also indicates its low efficiency for parameter recovery.

5.6.9 Scalability Experiments for IBCB on Synthetic Dataset

Furthermore, to address the concern about scalability, we have evaluated the solving time of IBCB with varying constraint sizes, as shown in Table 23.

TABLE 23

**Constraint comparison with same parameters mentioned in Section 5.4.3 of original manuscript.**  $N$  represents for the total episodes,  $B$  represents for the batch size of each episode, and  $M$  represents the action space size of candidates.  $d$  stands for the the number of candidate’s feature dimension.

Dataset	$N$	$B$	$M$	$d$	Constraint Size Level ( $m = N * B * M$ )	Train Time (sec.)
Synthetic	20	10	10	10	$\sim 10^3$	0.019±0.001
Synthetic	20	100	10	10	$\sim 10^4$	0.075±0.006
Synthetic	20	1000	10	10	$\sim 10^5$	0.815±0.045
Synthetic	10	2000	10	10	$\sim 10^5$	0.773±0.041
Synthetic	40	500	10	10	$\sim 10^5$	1.284±0.078
Synthetic	20	10000	10	10	$\sim 10^6$	12.973±1.639

We observe that the solving time of IBCB exhibits a sub-quadratic growth rate with respect to the problem size. Note that since the QP solver requires more time to polish the results when constraint size become larger, the increase in standard deviation is greater compared to the increase in mean value. This demonstrates that our method is capable of efficiently solving large-scale problems.

6 CONCLUSION

This paper aims to address the problem of efficiently learning behaviours from the expert’s evolution history. Specifically, we propose an inverse batched contextual bandit model called IBCB. The proposed IBCB gives a unified framework for both deterministic and randomized bandit

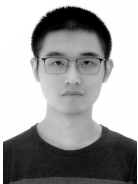
policies, and solves the problem of learning from evolution history with inaccessible rewards through a simple quadratic programming problem. Experimental results demonstrate the effectiveness and efficient training of IBCB on both synthetic and real-world scenarios. Besides, IBCB can also be generalized to out-of-distribution and contradictory data scenarios with great robustness. When it comes to limitations, IBCB needs to be further explored in experiments related to fairness, such as more random action selection methods, fairness assumptions based on specific distributions and other related settings. In addition, this article mainly discusses the UCB-like gambling machine framework. In the future work, we will focus on better random optimization of IBCB.

## REFERENCES

- [1] M. Bain and C. Sammut, "A framework for behavioural cloning," in *Machine Intelligence* 15, 1995, pp. 103–129.
- [2] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 101–103.
- [3] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–16, 2022.
- [4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.
- [5] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.
- [6] X. Zhang, H. Jia, H. Su, W. Wang, J. Xu, and J. Wen, "Counterfactual reward modification for streaming recommendation with delayed feedback," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 41–50.
- [7] X. Zhang, S. Dai, J. Xu, Z. Dong, Q. Dai, and J.-R. Wen, "Counteracting user attention bias in music streaming recommendation via reward modification," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2504–2514.
- [8] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel, "Streamrec: a real-time recommender system," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011, pp. 1243–1246.
- [9] S. Chang, Y. Zhang, J. Tang, D. Yin, Y. Chang, M. A. Hasegawa-Johnson, and T. S. Huang, "Streaming recommender systems," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 381–389.
- [10] M. Jakomin, Z. Bosnić, and T. Curk, "Simultaneous incremental matrix factorization for streaming recommender systems," *Expert Systems with Applications*, vol. 160, p. 113685, 2020.
- [11] L. Wang, Y. Bai, W. Sun, and T. Joachims, "Fairness of exposure in stochastic bandits," in *International Conference on Machine Learning*. PMLR, 2021, pp. 10 686–10 696.
- [12] S. Gillen, C. Jung, M. Kearns, and A. Roth, "Online learning with an unknown fairness metric," *Advances in neural information processing systems*, vol. 31, 2018.
- [13] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *IJCAI*, vol. 7, 2007, pp. 2586–2591.
- [14] J. Choi and K.-E. Kim, "Map inference for bayesian inverse reinforcement learning," *Advances in neural information processing systems*, vol. 24, 2011.
- [15] A. Hüyük, D. Jarrett, and M. van der Schaar, "Inverse contextual bandits: Learning how behavior evolves over time," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9506–9524.
- [16] E. K. Guha, J. T. James, K. Acharya, V. Muthukumar, and A. Pananjady, "One shot inverse reinforcement learning for stochastic linear bandits," in *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.
- [17] Y. Han, Z. Zhou, Z. Zhou, J. H. Blanchet, P. W. Glynn, and Y. Ye, "Sequential batch learning in finite-action linear contextual bandits," *CoRR*, vol. abs/2004.06321, 2020.
- [18] M. Dimakopoulou, Z. Zhou, S. Athey, and G. Imbens, "Balanced linear contextual bandits," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2019, pp. 3445–3453.
- [19] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 661–670.
- [20] A. S. Lan and R. G. Baraniuk, "A contextual bandits framework for personalized learning action selection," in *Proceedings of the 9th International Conference on Educational Data Mining*, 2016, pp. 424–429.
- [21] J. Yang, W. Hu, J. D. Lee, and S. S. Du, "Impact of representation learning in linear bandits," in *Proceedings of the 9th International Conference on Learning Representations*, 2021.
- [22] Z. Ren and Z. Zhou, "Dynamic batch learning in high-dimensional sparse linear contextual bandits," *arXiv preprint arXiv:2008.11918*, 2020.
- [23] Q. Gu, A. Karbasi, K. Khosravi, V. Mirrokni, and D. Zhou, "Batched neural bandits," *arXiv preprint arXiv:2102.13028*, 2021.
- [24] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.
- [25] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.
- [26] T. Osa, N. Sugita, and M. Mitsuishi, "Online trajectory planning and force control for automation of surgical tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 675–691, 2017.
- [27] E. Liu, M. Hashemi, K. Swersky, P. Ranganathan, and J. Ahn, "An imitation learning approach for cache replacement," in *International Conference on Machine Learning*. PMLR, 2020, pp. 6237–6247.
- [28] A. Balakrishna, B. Thananjeyan, J. Lee, F. Li, A. Zahed, J. E. Gonzalez, and K. Goldberg, "On-policy robot imitation learning from a converging supervisor," in *Conference on Robot Learning*. PMLR, 2020, pp. 24–41.
- [29] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.
- [30] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," in *Proceedings of the 6th International Conference on Learning Representations*, 2018.
- [31] A. H. Qureshi, B. Boots, and M. C. Yip, "Adversarial imitation via variational inverse reinforcement learning," in *Proceedings of the 7th International Conference on Learning Representations*, 2019.
- [32] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in Neural Information Processing Systems*, vol. 29, 2016.
- [33] X. Chen, L. Yao, A. Sun, X. Wang, X. Xu, and L. Zhu, "Generative inverse deep reinforcement learning for online recommendation," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 201–210.
- [34] D. S. Brown and S. Niekum, "Deep bayesian reward learning from preferences," *Workshop on Safety and Robustness in Decision-Making at the 33rd Conference on Neural Information Processing Systems (NeurIPS) 2019*, 2019.
- [35] R. F. Prudencio, M. R. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- [36] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.
- [37] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [38] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*. PMLR, 2019, pp. 2052–2062.
- [39] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.
- [40] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum, "Offline reinforcement learning with fisher divergence critic regularization,"

in *International Conference on Machine Learning*. PMLR, 2021, pp. 5774–5783.

- [41] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, “Advantage-weighted regression: Simple and scalable off-policy reinforcement learning,” *arXiv preprint arXiv:1910.00177*, 2019.
- [42] X. Chen, Z. Zhou, Z. Wang, C. Wang, Y. Wu, and K. Ross, “BAIL: Best-action imitation learning for batch deep reinforcement learning,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 353–18 363, 2020.
- [43] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, “MoReL: Model-based offline reinforcement learning,” *Advances in neural information processing systems*, vol. 33, pp. 21 810–21 823, 2020.
- [44] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, “COMBO: Conservative offline model-based policy optimization,” *Advances in neural information processing systems*, vol. 34, pp. 28 954–28 967, 2021.
- [45] E. D. Andersen and K. D. Andersen, “The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm,” in *High performance optimization*. Springer, 2000, pp. 197–232.
- [46] A. Domahidi, E. Chu, and S. Boyd, “Ecos: An socp solver for embedded systems,” in *2013 European control conference (ECC)*. IEEE, 2013, pp. 3071–3076.
- [47] E. M. Gertz and S. J. Wright, “Object-oriented software for quadratic programming,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 1, pp. 58–81, 2003.
- [48] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Mathematical Programming Computation*, vol. 6, pp. 327–363, 2014.
- [49] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, “OSQP: An operator splitting solver for quadratic programs,” *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.
- [50] M. Mansoury, B. Mobasher, and H. van Hoof, “Exposure-aware recommendation using contextual bandits,” *arXiv preprint arXiv:2209.01665*, 2022.
- [51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [52] X. Zhang, S. Dai, J. Xu, Y. Liu, and Z. Dong, “Adao2b: Adaptive online to batch conversion for out-of-distribution generalization,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 21, pp. 22 596–22 604, Apr. 2025. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/34418>
- [53] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, “Deep graph library: A graph-centric, highly-performant package for graph neural networks,” *arXiv preprint arXiv:1909.01315*, 2019.
- [54] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
- [55] M. Menéndez, J. Pardo, L. Pardo, and M. Pardo, “The jensen-shannon divergence,” *Journal of the Franklin Institute*, vol. 334, no. 2, pp. 307–318, 1997.



**Yi Xu** is currently pursuing his M.S. degree of Artificial Intelligence at Gaoling School of Artificial Intelligence, Renmin University of China (RUC). His current research interests lie at reinforcement learning, personalize search and recommender system via Large Language Models' reasoning.



**Weiran Shen** is a tenure-track associate professor at Gaoling School of Artificial Intelligence, Renmin University of China. His research interest includes multi-agent system, game theory, mechanism design, and machine learning. He published over 30 papers in top-tier conferences and journals in these research areas. He was PC, SPC, AC members of multiple international conferences including AAAI, IJCAI, ICML, NeurIPS, WWW, AAMAS.



**Xiao Zhang** is a tenure-track associate professor at Gaoling School of Artificial Intelligence, Renmin University of China. His research interests include online learning, trustworthy machine learning, and information retrieval. He has published over 40 papers on top-tier conferences and journals in artificial intelligence, e.g., NeurIPS, ICML, KDD, SIGIR, AAAI, IJCAI, ICDE, WWW, VLDB, etc.



**Jun Xu** is a professor with the Gaoling School of Artificial Intelligence, Renmin University of China. His research interests focus on learning to rank and semantic matching in web search. He served or is serving as SPC for SIGIR, WWW, and AAAI, editorial board member for Journal of the Association for Information Science and Technology, and associate editor for ACM TIST. He has won the Test of Time Award Honorable Mention in SIGIR (2019), Best Paper Award in AIRS (2010) and Best Paper



**Ji-Rong Wen** is a professor of the Renmin University of China (RUC). He is also the dean of the School of Information and executive dean of the Gaoling School of Artificial Intelligence with RUC. His main research interests include information retrieval, data mining, and machine learning. He was a senior researcher and group manager of the Web Search and Mining Group with Microsoft Research Asia (MSRA).