Table 1: Variants of DRL Agents used

Methods	Batch-Constrained	Distributional
DQN	×	×
QRDQN	×	✓
IQN	×	✓
BCQ	✓	×
QRBCQ	✓	✓
BCD4Rec	✓	✓

A Details of Deep RL agents used

We elaborate on the features contrasting all the RL agents (RAs) as summarized in Table 1.

A.1 Deep Q-Networks (DQN)

Deep Q-network (DQN), parameterized by $\boldsymbol{\theta}$ is used as a function approximator to estimate the action-value function, i.e., $Q(s,a) \approx Q_{\boldsymbol{\theta}}(s,a)$, while encoding states and actions in terms of real-valued embedding vectors. We use double DQN (hereafter, DQN refers to the double DQN variant), which uses two networks $Q_{\boldsymbol{\theta}}$ and $Q_{\boldsymbol{\theta}'}$ to mitigate the overestimation bias of DQN by iteratively minimizing the following loss $\mathcal{L}_{DQN}(\boldsymbol{\theta})$ estimated over mini-batches of transitions (s,a,r,s') sampled from batch data $\mathcal{B}$:

$$\mathcal{L}_{DQN}(\boldsymbol{\theta}) = \mathbb{E}_{s,a,r,s'}[L_\kappa(r + \gamma \max_{a'} Q_{\boldsymbol{\theta}'}(s',a') - Q_{\boldsymbol{\theta}}(s,a))], \tag{1}$$

where L_κ is the Huber loss: $L_\kappa(\delta) = 0.5\delta^2$ if $\delta \leq \kappa$, and $\kappa(|\delta| - 0.5\kappa)$ otherwise; $Q_{\boldsymbol{\theta}'}$ is the target network with parameters $\boldsymbol{\theta}'$ fixed over multiple training steps or update iterations for $\boldsymbol{\theta}$, and $\boldsymbol{\theta}'$ is updated to $\boldsymbol{\theta}$ after a set number of training steps.

A.2 Distributional RL with Quantile Regression DQN (QRDQN)

In QRDQN, a set of K τ-quantiles of the value distribution, $\{\tau_i\}^K = \{\frac{i+0.5}{K}\}_{i=0}^{K-1}$ is estimated. Instead of estimating just the (expected) value for an action, a K-dimensional vector representing the K τ-quantiles is produced. So, the overall output of QR-DQN is of size $|\mathcal{A}| \times K$ instead of $|\mathcal{A}|$. The loss is computed over all pairs of quantiles as follows:

$$\mathcal{L}_{QRDQN}(\boldsymbol{\theta}) = \frac{1}{K^2}\mathbb{E}_{s,a,r,s'}\left[\sum_\tau \sum_{\tau'} l_\tau \left(r + \gamma \max_{a'} Q^{\tau'}_{\boldsymbol{\theta}'}(s',a') - Q^{\tau}_{\boldsymbol{\theta}}(s,a)\right)\right], \tag{2}$$

where l_τ is the quantile Huber loss $l_\tau(\delta) = |\tau - \mathbb{I}(\delta < 0)|L_\kappa(\delta)$. An estimate of the value can be recovered through the mean over the quantiles, and the policy π is defined by greedy selection over this value: $\pi(s) = \arg\max_a \frac{1}{K}\sum_\tau Q^\tau_{\boldsymbol{\theta}}(s,a)$.

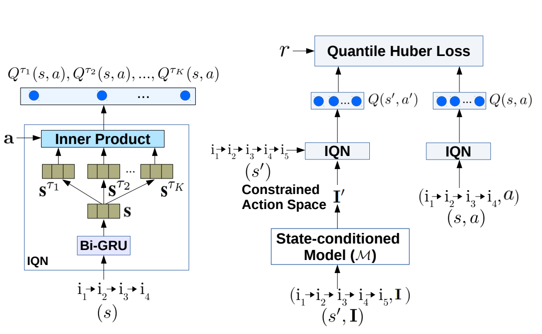
The batch-constrained variants of DQN and QRDQN, i.e. **BCQ** and **QRBCQ** are also trained using losses $\mathcal{L}_{DQN}$ and $\mathcal{L}_{QRDQN}$ respectively, but with the additional action constraining criteria for a':

$$a' = \operatorname*{arg\,max}_{a'|p_{\mathcal{M}}(a'|s')>\beta} \frac{1}{K}\sum_\tau Q^\tau_{\boldsymbol{\theta}}(s',a'), \tag{3}$$

which is same as that used for BCD4Rec.

A.3 BCD4Rec

Here we provide additional details for training BCD4Rec in Algorithm 1, and a schematic of BCD4Rec contrasting it with vanilla DQN in Fig. 1. **IQN** without batch constraining is equivalent to BCD4Rec with $\beta = 0$.


$Q^{\tau_1}(s,a), Q^{\tau_2}(s,a), ..., Q^{\tau_K}(s,a)$
a
Inner Product
$\mathbf{s}^{\tau_1}$
$\mathbf{s}^{\tau_2}$
$\mathbf{s}^{\tau_K}$
s
Bi-GRU
IQN
$i_1 \to i_2 \to i_3 \to i_4$
(s)
r
Quantile Huber Loss
$Q(s', a')$
$Q(s, a)$
$i_1 \to i_2 \to i_3 \to i_4 \to i_5$
IQN
(s')
Constrained Action Space
$\mathbf{I}'$
State-conditioned Model ($\mathcal{M}$)
$(i_1 \to i_2 \to i_3 \to i_4 \to i_5, \mathbf{I})$
$(s', \mathbf{I})$
IQN
$(i_1 \to i_2 \to i_3 \to i_4, a)$
(s, a)
$Q(s,a)$
a
Inner Product
s
Bi-GRU
DQN
$i_1 \to i_2 \to i_3 \to i_4$
(s)
r
Huber Loss
$Q(s', a')$
$Q(s, a)$
DQN
DQN
$(i_1 \to i_2 \to i_3 \to i_4 \to i_5, \mathbf{I})$
$(i_1 \to i_2 \to i_3 \to i_4, a)$
$(s', \mathbf{I})$
(s, a)

(a) Proposed Approach BCD4Rec (b) Vanilla DQN

Figure 1: Handling of a tuple (s, a, r, s') in the proposed approach (BCD4Rec) in contrast to a traditional DQN. The IQN module in BCD4Rec estimates K quantiles of the value distribution while vanilla DQN only estimates the expected (mean) value. Furthermore, the state-conditioned model $\mathcal{M}$ restricts the action space for the BCD4Rec agent. While italic s and a denote the state and action, bold $\mathbf{s}$ and $\mathbf{a}$ denote the state and action embeddings, respectively. In the example considered, we assume the action a to consist of item i_5 that is clicked by the user, leading to updated state $s' = (i_1, i_2, i_3, i_4, i_5)$.

Algorithm 1 BCD4Rec

```
1: Input: Batch $\mathcal{B}$, number of iterations $T$, $targetUpdateRate$, mini-batch size $N$, $\mathcal{I}$.
2: Initialize the Q-network $Q^{\tau}_{\boldsymbol{\theta}}$ (initialize the item embeddings using pre-trained embeddings), conditional model $\mathcal{M}$ and target network $Q^{\tau'}_{\boldsymbol{\theta}'}$ with $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$.
3: for $t = 1, 2, \ldots T$ do
4:     Sample mini-batch $M$ of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$.
5:     $a' = \arg\max_{a'|p_{\mathcal{M}}(a'|s') > \beta} \frac{1}{K} \sum_{\tau} Q^{\tau}_{\boldsymbol{\theta}}(s', a')$
6:     $\boldsymbol{\theta} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{BCD}(\boldsymbol{\theta})$
7:     $\omega \leftarrow \arg\min_{\omega} -\sum_{(s,a)\in M} \log p_{\mathcal{M}}(a|s;\omega)$
8:     If $t$ mod $targetUpdateRate = 0$ : $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
9: end for
```

B RecSim Simulation Environment

We consider the Interest Evolution environment of RecSim[1], where the goal is to evaluate RL algorithms to keep a user engaged for as long as possible (we consider the maximum episode length as 20) by showing relevant items that the user would be interested in. This environment consists of three main modules: i. *user model*, ii. *item model*, and iii. *user choice model*, as summarized in Algorithm 2. This environment has two user response types: click and skip. A user u is presented with a slate sl consisting of k items and a special skip item such that the effective slate size is $k + 1$. The interest or the relevance score $I(u, i)$ of an item i for the user u is defined as per line 10 of the algorithm, while s_u corresponds to a score for the skip item (in this work, we use the relevance score for the second-most relevant item for a user as the relevance score for the skip item). The relevance scores for the recommended items are used to get the probability of clicking an item from a given slate. For each item $i \in sl$, this probability is computed as per line 11, and the action by the user is drawn as per this probability distribution. If u clicks on i, the relevance score or the interest for the corresponding category is updated as per lines $13 - 16$. Lines $15 - 16$ ensure that $u's$ interests are reinforced as the episode progresses, i.e. if u clicks and consumes an item from a category where she had high interest to begin with, the interest in that category is likely to go up. We consider the default settings of this environment with $C = 20$ and y as 0.3.

We consider a random (exploration) policy as one of the behavior policy which is referred to as RecSim-1, and results in batch data with lowest CTR. We train an IQN agent (variant of BCD4Rec with $\beta = 0$) from scratch with ϵ-greedy exploration (where ϵ degrades linearly) in online manner. We

[1]`https://github.com/google-research/recsim`

Algorithm 2 RecSim: Interest Evolution Environment

```
1: Input: no. of categories, $C$
2: **User model:**
3: Interest vector of user $u$, $\mathbf{u} = [I_1, I_2, ..., I_C]$, where $I_c \sim U([-1, 1])$ and $I_c$ is user's interest in category $c$
4: **Item model:**
5: One-hot category vector of item $i$, $\mathbf{i} \in \{0, 1\}^C$
6: **User choice model:**
7: Given a slate ($sl$) of size, $k + 1$ (i.e. list of $k$ recommended items by agent and one skip item),
8: Position of item $i$ in slate $pos(i) \in \{0, 1, ..., k\}$
9: User $u$'s interest for item $i$,
10: $I(u, i) = \begin{cases} \mathbf{u}^T\mathbf{i}, & \text{if } pos(i) \in \{0, 1, ..., k-1\} \\ s_u, & \text{if } pos(i) = k \end{cases}$
11: $p(u, i) = \frac{I(u,i)}{\sum_{j \in sl} I(u,j)}$
12: **User interest updation:**
13: $I_c$: user $u$'s interest in category $c$ whose item is consumed
14: $\Delta(I_c) = (-y|I_c| + y).(1 - I_c)$ , $y \in [0, 1]$
15: $I_c \leftarrow I_c + \Delta(I_c)$ with probability $[I(u, i) + 1]/2$
16: $I_c \leftarrow I_c - \Delta(I_c)$ with probability $[1 - I(u, i)]/2$
```

Table 2: Details of the datasets used. Here: s:skip, c: click, b: buy.

Statistics	**Diginetica**	**RecSim**
#train sessions	4843	2000
#train tuples	70k	30k
#test sessions	1436	200
#items	6666	200
Response Types	{s,c,b}	{s,c}
Target Response	b	c

train this for $1k$ episodes and select behavior policies at different timesteps of training, referred as RecSim-2 and RecSim-3, respectively. We keep the user interest vector **u** latent (except for optimal policy) while training RL agents (online/offline) and non-RL baselines to mimic the SR scenario. For online policy, we consider **u** to be fully observable to an agent.

The behavior policies are then used to generate logs as batch data for evaluating various approaches in batch RL setting. During training, the default rewards of skip:0 and click:4 are used. The agents trained using the batch data are compared on 200 previously unseen new users.

C Evaluation Metrics

While all RAs are trained in offline fashion using batch data, we evaluate them under two scenarios: i. online, and ii. offline, as is common in literature.

Metrics for online testing: Depending upon the target response type that needs to be maximized by the RA, we compute the **CTR** (click through rate) or the **BR** (buy rate) as the percentage of responses corresponding to the target response type (i.e. buy for DN, click for RecSim) across all episodes or sessions. **C@X** (Coverage@X) denotes the percentage of items from $\mathcal{I}$ that are recommended at least once across test episodes by the agent within top-X items at any recommendation step.

Metrics for Offline Evaluation: **i) R@X (Recall@X):** Given the initial interactions from a test session, the task is to re-rank the future interacted ground truth items from the session. We compute the standard $R@X$ metric as the percentage of times the eventually clicked or bought items appear in the top-X items in the re-ranked list. **ii) $\bar{\mathbf{Q}}$:** Average over the q-values of the evaluation policy for the given state distribution $\mathbb{E}_{s \sim \mathcal{B}}[Q_\theta(s, a)]$, i.e. the average Q across all states for the action a chosen as per the RA policy.

Table 3: Hyperparameters considered and the best hyperparameters obtained using $\bar{\mathbf{Q}}$.

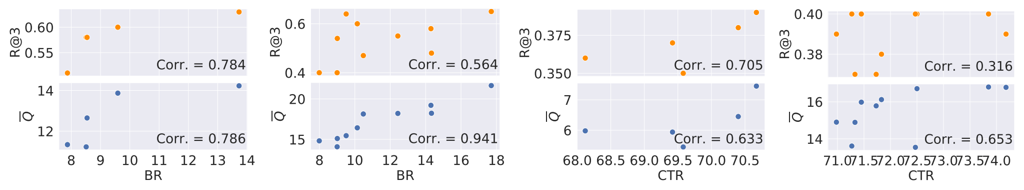
Hyper-parameter (Algorithm)	**Range Tried**	**Selected-DN**	**Selected-RecSim**
Quantiles K (DQN/BCQ)	1	1	1
Quantiles K ($QRDQN/QRBCQ$)	$5, 7, 10$	5	5
Quantiles K ($IQN/BCD4Rec$)	$5, 7, 10$	5	10
Cosines Number n ($IQN/BCD4Rec$)	$32, 64, 128$	64	128
BC Threshold β ($BCD4Rec/QRBCQ/BCQ$)	$0.1, 0.3, 0.5, 0.7, 0.9$	0.3	0.5
Learning Rate (All)	$0.0003, 0.001, 0.003$	0.003	0.003
HiddenSize d (All)	100	100	100
Bi-GRU Layers (All)	$2, 3$	2	2
Bi-GRU Hidden Units (All)	$d/2 \times K$	$d/2 \times K$	$d/2 \times K$
Discounted Factor γ (All)	0.9	0.9	0.9
Optimizer	$ADAM$	$ADAM$	$ADAM$
Recent positive interactions L	10	10	10
Mini-Batch Size	64	64	64

Table 4: Pearson correlation coefficient (PCC) between online and offline evaluation metrics for various recommender agents. PCC is computed over the original values for the metrics (value-based), and by ranking the values and computing correlation over the ranks (rank-based). We observe that $\bar{\mathbf{Q}}$ has higher rank-based as well as value-based correlation with online metrics.

	Diginetica				**RecSim-2**			
	Value-based		**Rank-based**		**Value-based**		**Rank-based**	
Algorithm	**R@3**	**Q**	**R@3**	**Q**	**R@3**	**Q**	**R@3**	**Q**
BCQ	0.784	**0.786**	**0.975**	0.900	**0.705**	0.633	**0.700**	0.600
QRDQN	0.564	**0.941**	0.627	**0.958**	0.316	**0.653**	0.174	**0.768**
BCD4Rec	0.172	**0.943**	0.227	**0.961**	0.531	**0.896**	0.204	**0.898**

D Pre-processing and Hyperparameters Selection

We pre-process the batch data to obtain incremental sessions. Each incremental session results in a tuple of (s, a, r, s'). The final data related statistics are available in Table 2. We use 20% of the train set as hold-out validation set for tuning the hyperparameters in completely offline manner. Selecting the best hyperparameters using only offline logs and metrics while optimizing for the online evaluation metrics is a non-trivial task. This demands for the reliable offline evaluation metrics, preferably to be highly correlated with the online ones. We consider $R@3$ and $\bar{Q}$ as offline evaluation metrics (described in Section C). We find $\bar{Q}$ to be a better metric in comparison to $R@3$. The performance of $\bar{Q}$ is inline with the performance of online metrics as it is highly correlated with the online metrics, as shown in Table 4. We also compare performance of BCQ and QRBCQ in terms of online and offline evaluation metrics while considering varying hyperparameters set, as shown in Fig 2. Results indicate the reliability in considering $\bar{Q}$ over $R@3$ for selecting best hyperparameters set. The finally selected best hyperparameters are summarized in Table 3. Further, the test environment for Diginetica (DN) is a bi-directional GRU of the same size as Q-Networks, and is trained to classify the response type for the item recommended by the agent given the items interacted so far.

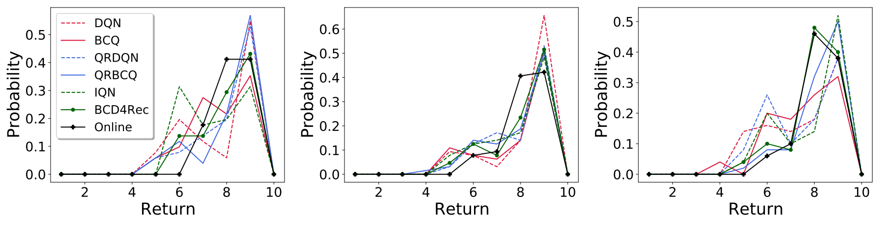


(a) BCQ (DN) (b) QRBCQ (DN) (c) BCQ (RecSim-2) (d) QRBCQ (RecSim-2)

Figure 2: Comparison of online evaluation metrics (BR/CTR) with offline evaluation metrics ($R@3/\bar{Q}$) for varying values of β and K. The higher correlation between online metric BR/CTR and the offline metric $\bar{Q}$ indicates higher reliability of $\bar{Q}$ over $R@3$ for hyperparameters selection in batch RL settings. Same is observed for BCD4Rec.

Table 5: Comparison of the learned value distributions of various recommender agents against the corresponding value distribution from online policy in terms of Wasserstein distance metric for different user types for RecSim-2.

Users	**DQN**	**BCQ**	**QRDQN**	**QRBCQ**	**IQN**	**BCD4Rec**
$User_1$	0.055	0.039	0.051	0.047	0.039	**0.031**
$User_2$	0.053	0.043	0.047	0.044	0.047	**0.034**
$User_3$	0.056	0.052	0.040	0.016	0.036	**0.010**



(a) $user_1\ (s_0, a_0)$ (b) $user_2\ (s_0, a_0)$ (c) $user_3\ (s_0, a_0)$

Figure 3: The learned value distributions $Z^\pi(s_0, a_0)$ for various agents given same initial state-action pair (s_0, a_0) for RecSim-2 agents. Initial (s_0, a_0) are randomly chosen, and returns are evaluated across randomly sampled 50 users from three different user types with discounted rewards over 20 steps with $\gamma = 0.9$.

E Comparison of learned value distributions of offline RL agents w.r.t. Online Policy

We observe the learned value (return) distributions of various RAs. These RAs are trained using the logs generated by RecSim-2 behavior policy. For this, we group the users having maximum interest in three randomly chosen categories as $User_1$, $User_2$ and $User_3$, respectively. The returns are evaluated across randomly selected 50 users from each of the three user types with discounted rewards over 20 steps with γ = 0.9, given the same initial (s_0, a_0) pair. We compare these learned value distributions against the corresponding value distribution from the online policy using *Wasserstein distance*. The Wasserstein distance related numbers are shown in Table 5 whereas the respective learned value distributions are shown in Fig. 3. These results depict that the value distribution obtained using BCD4Rec agent is closer to the corresponding value distribution from the online agent in comparison to the value distributions obtained from other RAs.