t-RELOAD: A REinforcement Learning-based REcommendation for Outcome-driven Application

Debanjan Sadhukhan
Games24x7
Bangalore, India
debanjan.sadhukhan@games24x7.com

Sachin Kumar
Games24x7
Bangalore, India
sachin.kumar@games24x7.com

Swarit Sankule
Games24x7
Bangalore, India
swarit.sankule@games24x7.com

Tridib Mukherjee
Games24x7
Bangalore, India
Tridib.Mukherjee@games24x7.com

ABSTRACT

Games of skill provide an excellent source of entertainment to realize self-esteem, relaxation and social gratification. Engagement in online skill gaming platforms is however heavily dependent on the *outcomes* and experience (e.g., wins/losses). A user can behave differently under different win/loss experience. An intense engagement can lead to potential demotivation and consequential churn, while a lighter engagement can lead to more confidence for longer sustenance—all depending on the outcomes. Generating a relevant recommendation using reinforcement learning (RL) that can also lead to high engagement (both intensity and duration) is non-trivial because: (i) an early exploration through online-RL using a combined multi-objective reward can permanently hurt users; and (ii) a simulation environment to evaluate RL policies is hard to model due to the (unknown) outcome-driven natural volatility in user behaviour. This work addresses the question "*how can we leverage off-policy data for recommendation to solve cold-start problem while ensuring reward-driven optimality from platform-perspective in outcome-based applications?*". We introduce *t*-RELOAD: A REinforcement Learning-based REcommendation framework for Outcome-driven Application consisting of 3-layer-based architecture: (i) off-policy data-collection (through already deployed solution), (ii) offline training (using relevancy) and, (iii) online exploration with turbo-reward (*t*-reward, using engagement). We compare the performance of *t*-RELOAD with an XGBoost-based recommendation system already in-place to capture the effectiveness.

ACM Reference Format:
Debanjan Sadhukhan, Sachin Kumar, Swarit Sankule, and Tridib Mukherjee. 2023. *t*-RELOAD: A REinforcement Learning-based REcommendation for Outcome-driven Application. In *Proceedings of The Third International Conference on Artificial Intelligence and Machine Learning Systems (AIMLSystems 2023)*. ACM, Bangalore, India, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

AIMLSystems 2023, October 25–28, 2023, Bangalore, India

https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In online skill-gaming like Rummy-Circle (RC)[20], a user interacts in the platform to play skill-game to realize self-esteem, relaxation and social gratification. The user-engagement is obtained by combining both the duration (or frequency) and intensity of the game-play. Games are categorized in an ordinal form (from 0 to 9) based on the intensity. A novice player plays lower ordinal games, as they involve lower risk and skill, however a skillful player plays higher ordinal games. A wallet is maintained with amount initiated by the players, which reduces in proportion of ordinal choice of the intensity and the outcomes of the games. A user can not initiate a game play when the wallet gets fully depleted. In this work, *we aim to construct an agent to generate personalized, automated, ordinality-recommendation for such a system.*

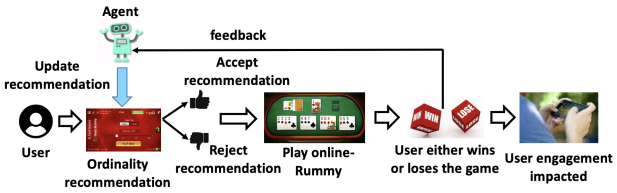
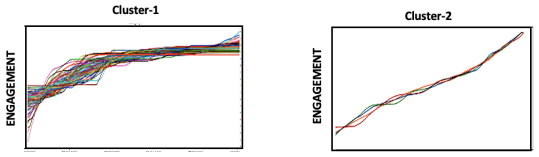


Figure 1: Flow diagram of user-engagement in RC

Background: The agent is responsible for generating the ordinality before starting a game (refer Fig. 1) which impacts the overall user-engagement. The agent intends to maximize both: (i) user-centric objective, i.e., the most relevant ordinality pertaining to the persona and intent; (ii) platform-centric objective, i.e., increases user-engagement on the platform. Unlike in most consumer-centric platforms requiring item recommendation, e.g., OTT [7], e-commerce [21], relevance of a recommendation may be orthogonal to engagement in outcome-based platforms such as online skill gaming, because of—(i) diversity of user intent, e.g., exploration, challenge, social gratification, requiring adaptation to often "unknown" preference; (ii) high volatility in these preferences, e.g., user starts for exploring and grows interests, depending on outcomes; (iii) transient interplay between ordinality choice / recommendation, game-play, outcomes, and the wallet-balance, further impacting subsequent intent; (iv) consequent unique longitudinal game-play behavioral patterns by the users' engagement and transaction choices; (v) huge variety and

volume of data generated by every user, e.g., same user showing different intent / behavior under different outcomes depending on longitudical evolution; and many more [20, 32].

Recommending a "relevant" higher ordinal game at an inappropriate stage of career may end up losing all wallet-balance leading to eventual churn. In order to understand the users playing trajectory, we clustered the normalized cumulative engagement intensity of two kind-of-users (using hierarchical clustering with DTW distance metric [16] as shown in Fig 2). The first-kind of users are chosen such that, the engagement intensity of user is more initially however second-kind of users progressively increase the same. Note that, the engagement intensity has a negative impact on overall engagement for first-kind and these users eventually churn-out of the system. On the contrary, even though the intensity is less, the second-kind-of-users show higher long-term engagement (or longer sustenance). Hence, based on the longitudinal game-play behavioral patterns, recommending personalized ordinality for users involves analyzing the current state of the user, past transactions and game-play behavior to maximize both user and platform-centric objectives.



(a) Normalized engagement for Cluster 1

(b) Normalized engagement for Cluster 2

Figure 2: DTW-based hierarchical clustering

Motivation: Considering such factors, reinforcement leanring (RL)-based approaches are more suitable (when compared to supervised learning-based recommendations) for applications like skill-gaming because (i) RL considers the long-term impact of the recommendation (or actions) and select most appropriate action accordingly, (ii) captures sequential nature of the user (using Markov Decision Process or MDP), (iii) incorporates exploration and exploitation methodology to discover global optimal trajectories that maximizes expected discounted cumulative rewards, (iv) scalable and dynamic (captures change in user preferences), (v) support serendipity (having a diverse set of recommendation) through exploration, and (vi) approximates the user state information while recommending the suitable actions (if a user's current wallet balance is more then suggest higher ordinality games).

The RL-based techniques can be categorized into online or offline: online agent iteratively interacts with the environment to collect its experience with exploration, whereas the offline agent utilizes offline data to train itself. However, such exploration involves irrelevant recommendation to a few users which is not encouraged as they highly impact the engagement. Offline-based techniques involve training the model parameters in an offline setup (refer to as behavior policy (π_a) in Fig. 3) and deploy the trained model to generate sub-optimal recommendations, however often preferred over the online method as they minimize the opportunity cost during deployment. Recent work suggests offline strategies can be

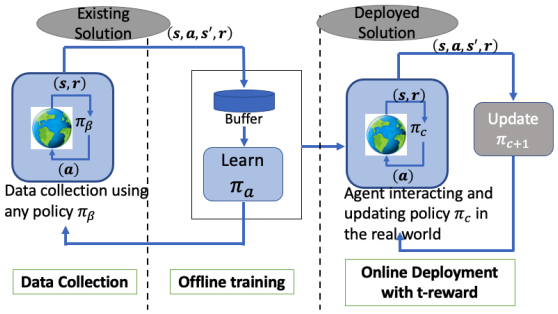


Figure 3: *t*-RELOAD

utilized to learn directly the optimal policy from the offline data using sampling estimation techniques [15] where the important weight or $\frac{\pi_*(a|s)}{\pi_\beta(a|s)}$ ($\pi_*(a|s)$ denotes optimal strategy and $\pi_\beta(a|s)$ denotes behavior strategy, refer Fig. 3) is estimated to adjust the off-policy return. However, in-reality the offline data may not contain the optimal trajectories and hence estimating $\frac{\pi_*(a|s)}{\pi_\beta(a|s)}$ is very difficult. In other words, the challenges are as follows:

- In outcome-driven online-Rummy such as RC, the exploration during early deployment in online RL strategies impacts the user-engagement and wallet balance in a negative way.
- Offline strategies are only limited to solve sub-optimal user-centric solution and often unable to optimize the platform-centric objective especially in absence of data with optimal trajectories.

Contributions: This work addresses the question "*how can we leverage off-policy data for recommendation to solve cold-start problem while ensuring reward-driven optimality from platform-perspective in outcome-based applications?*". We introduce *t*-RELOAD which combines the effectiveness of both offline and online strategies. Specifically, the agent utilizes offline-strategies to maximize user-centric and online-stratigies to maximize platform-centric objectives. Such a framework is also a good candidate to solve the cold-start problem which is expensive especially during the early phase of exploration in outcome-based applications. Especially in outcome-driven applications (such as stock-recommendation and health-care) where the simulation-building is very tedious, the proposed framework can be utilized. Specifically:

- We introduce *t*-RELOAD: A REinforcement Learning- based REcommendation framework for Outcome-driven Application consisting of 3-layer-based architecture as follows: (i) off-policy data-collection (through already deployed solution), (ii) offline training (using relevancy) and, (iii) online exploration using turbo-reward (t-reward).
- To the best of our knowledge, we are the first who tested and deployed deep reinforcement learning based personalized recommendation system in online, skill-gaming platform using a hierarchical Double-DQN based agent framework with noise clipping.

This type of framework provides the ability to have turbo-reward which transforms the network to serve from user-centric to platform-centric (refer Fig. 3). The organisation of the paper is as follows. The related work is discussed in Sec. 2. Sec. 3 proposes *t*-RELOAD based three layer architecture. Finally, Sec. 4 concludes the paper and provides pointers to future work.

2 RELATED WORKS

2.1 Deep Reinforcement Learning (DRL)

Initial work on DRL involves processing image pixels to develop agents for Atari games [19] using a value-function based method called Deep-Q-network (DQN). Several extension of DQN are proposed in [26, 28] to handle the overestimation problem involves in DQN. However, in policy gradient methods, the policy is optimized directly without associating any value function for each action. The idea is to utilize a parameterized function θ over the state features to directly obtain the policy distribution, $\pi_\theta(s, a)$ [25, 29]. However, a in hybrid strategy called Actor-critic combines both the value function and policy gradient based approaches [11, 12]. Further, the DRL-driven recommendation system (RS) methods can be categorized based on the state representation technique used and policy optimization methodologies utilized as follows.

2.1.1 State Representation. State information must not include all information but contain enough information from the past to summarize such that all relevant information is present [25]. However, if the item set is not small, feature extraction (combining user, item, and context information) is used [1]. Some DRL-based models utilize embedding techniques which convert the user, item and context information into a latent continuous, low dimensional, dense space for processing [31]. RNN [31], CNN [10], and GAN [10]-based models are also leveraged to build such embedding.

2.1.2 Policy Optimization. The policy optimization techniques used in DRL-based recommendations can be divided into value-based, policy gradient and actor-critic as discussed earlier. The value based network architecture can be divided into two broad categories where in the first type of architecture input to the model is the state information and the output is the Q-value for each state-action pair [18, 19]. This architecture works well with small set of actions, however for a large action set the second type of architecture is preferred where the input to the model is state and action pair, and the output is the corresponding Q-value. This type of architecture is also preferred where the action space is continuous and not discrete values [1]. In contrast to value-based methods, policy-based methods (such as REINFORCE [1]) encounter higher variance and slow learning. To further optimize the objective, instead of using ϵ-greedy strategy during exploration, a separate network called explore network is used [30].

2.2 Offline Reinforcement Learning

The offline learning techniques completely rely on static dataset D, and hence the exploration is not possible. If D fails to contain the state-action-space transitions with high-reward regions, the offline algorithm is limited to discover the sub-optimal solutions. Hence, the existing offline strategies assume the dataset D covers the space for the high-reward regions [15]. The objective of the offline policy is to learn a policy π_a better than the behavior policy π_β (refer Fig. 3) by executing actions different from the behavior policy. Generally, these techniques rely on evaluating the return more efficiently to come-up with such strategies. Important sampling-based estimation [6, 27] incorporate important-weight before estimating the return as follows:

$$J(\pi_\theta) = E_{\tau\sim\pi_\beta(\tau))}\left[\frac{\pi_\theta(\tau)}{\pi_\beta(\tau)}\sum_{t=0}^{n}\gamma^t r(s,a)\right] \quad (1)$$

where $r(s, a)$ denotes the reward generated at state s with action a, γ is a discounted factor, $\pi_\beta(\tau)$ and $\pi_\theta(\tau)$ respectively denotes the behavior and optimal policy. While estimating the Q-values for a state-action pair, such sampling techniques are utilized during return estimation. Further, important sampling based policy-gradient techniques are employed in [3, 13] which update the weights directly to estimate the optimal policy based on such sampling estimation during a trajectory. Normalizing techniques are further applied to contain the high-variance involve in such sampling [23].

3 *t* -RELOAD

This section introduces *t*-RELOAD recommendation framework for outcome driven applications with three layer architecture: (i) off-policy data-collection (through already deployed solution), (ii) offline training (using relevancy) and, (iii) online exploration with turbo reward. In an offline setting, reward can only be formulated as a function of relevancy or can only be user-centric as the recommendation have no impact over the engagement.

Preliminaries: We utilize a Markov decision process (MDP), defined as $M = (S, A, T, r, \gamma)$, where S denotes the set of states $s \in S$, A denotes the set of actions $a \in A$, T denotes the transitional probability $T(s_{t+1}|s_t, a_t)$, $r : S \times A \rightarrow R$ denotes the reward, and $\gamma \in [0, 1]$ is a scalar for discounted factor. The objective of the agent is to learn the optimal policy π_* that maximizes the returns. A policy, $\pi(a|s)$, is defined as a function that maps between state to possible actions in set A. A policy is optimal if the expected return ($R_t = E[\sum_{k=0}^{\infty}\gamma^k r_{t+k}]$) is at least same or more (from all the states) than any other possible policy. More formally, if $v_\pi(s)$ denotes the value of a state (where $v_\pi(s) = E_\pi[R_t|s_t = s]$), then a policy π is better than a policy π' if and only if $v_\pi(s) \geq v_{\pi'}(s)\forall s \in S$. The value of all the states in the optimal policy is at least same or more than value in any other policy. Similar to the value of a state, the value of a state-action pair ($q_\pi(s, a) = E_\pi[R_t|s_t = s, a_t = a]$) denotes the expected return the agent receives from state s with action a at time t under the policy π. This is also known as the Q-value, and the optimal Q-value is denoted by $q_*(s, a) = E_{s'}[r_{t+1} + \gamma\, max_{a'} q_*(s', a')]$. In other words, the $q_*(s, a)$ denotes the maximum achievable expected return from state s while taking an action a under the current policy.

Problem Formulation: In order to solve the cold-start problem, *t*-RELOAD learns in an offline setting the most relevant ordinality based on the user profile and past interactions at time t. The problem is modelled as a MDP which involves (S, A, T, r, γ) as follows.

State(s_t): State $s_t \in S$ includes the user's current wallet balance, last ordinality, distribution of various ordinality, user's geographic

location, win-rate, lost-rate, number of practice games, and many more (refer Table. 1).

Action(a_t^{rec}): The action $a_t^{rec} \in A$ denotes the recommended ordinality at time t. Instead of predicting the ordinality (ranges from 0 to 9) directly, we predict the jump from the last ordinality (LO). As there are a total of 10 ordinals from 0 to 9 and hence the action can be in the range between $0 \leq a_t \in \mathbb{N} \leq 9$, where 0 means no push, 1 means 1-more-step push than LO, and so on.

Reward(r_t): As the training is done in an offline setting, the relevancy (such as predicting the relevant amount of push) is optimized. Every time the recommended action a_t^{rec} matches with the actual action a_t^{actual}, a positive reward is generated (and otherwise a negetive reward). Hence, the reward function $r(s, a_t^{rec}, a_t^{actual})$ is denoted by

$$\begin{aligned} r_t(s, a_t^{rec}, a_t^{actual}) &= \lambda \times a_t^{actual}, \text{if } a_t^{actual} == a_t^{rec} \\ &= -\beta |a_t^{actual} - a_t^{rec}| a_t^{actual}, \text{else} \end{aligned} \quad (2)$$

where λ and β are stationary constants. In practice, as the higher ordinal games are associated with higher engagement-intensity (refer Fig. 5c), our objective is to accurately identify such instances at a cost of hurting the relevancy of low-push users. The reward function is designed in a way such that more rewards are associated with accurately recommending higher push classes (using λ) compared to lower push classes. Similarly, more penalty is associated with miss-classifying higher push classes (using β) compared to lower push classes. The term $|a^{actual} - a^{rec}|$ gives more penalty if the recommended action is farther away from the actual action.

Objective: The objective of the t-RELOAD based offline agent is to find the optimal policy (π_*) that maximizes the expected achievable return $E_\pi[R_t | s_t = s]$ for all state-action pair.

3.1 Double DQN with noise clipping

In t-RELOAD based offline training based framework, the overall objective is to find the optimal Q-value for any tuple (s, a^{rec}, r, s') that follows Bellman equation

$$Q_*(s, a^{rec}) = E_{s'}[r(s, a^{rec}, a^{actual}) + \gamma \, max_{a'} Q_*(s', a')] \quad (3)$$

where $r_t(s_t, a_t^{rec}, a_t^{actual})$ denotes the reward generated as shown in Eq. 2 at time t. The optimal policy can be greedy such that from any state s, the action giving maximum Q-value is selected. However, in reality the state-action space is huge and estimating optimal Q-value using Bellman equation is not feasible. Hence, deep neural network based approximators ($Q_*(s, a^{rec}) \approx Q_*(s, a^{rec}; \theta)$) are used by minimizing the following loss function [5, 31]

$$L(\theta) = E_{s, a^{rec}, r, s'}[(y - Q_*(s, a^{rec}; \theta))^2], \quad (4)$$

where $y = E_{s'}[r(s, a^{rec}, a^{actual}) + \gamma \, max_{a'} Q_*(s', a')]$ is the target for the current iteration. This technique is known as deep Q-network (DQN) [5, 31]. Further, to increase the stability of the network, current updated weights are kept fixed and fed into a second (duplicate) network (known as target network) whose outputs are used as Q-learning target. In other words, $max_{a'} Q_*(s', a'; \theta)$ comes from the target network (with weights θ') and replaced by $max_{a'} Q_*(s', a'; \theta')$. In order to alleviate the overestimation problem induced in DQN, Double-DQN [26] replaces the target value as $y = E_{s'}[r(s, a^{rec}, a^{actual}) + \gamma \, Q_*(s', arg\, max_{a'} Q_*(s', a'; \theta); \theta')]$.

Table 1: Feature Description

Feature	Rational
Current wallet-balance	Higher current wallet implies user can spend more with higher ordinality
Win rate for each game type	User is expert at current level and can be pushed at next level
Losing rate for each game type	If the user lost more games, needs to play at lower ordinality
Total practice games played	User knows how to play in the platform
Last ordinality amount for each game type	User can be pushed more often at lower ordinality vs higher ordinality
Total number of games played with each ordinality & game-type	Distribution of previous play-pattern denotes the behavior of the user

Further to avoid the over-fitting to narrow peaks in the value function, y is replaced by $y = E_{s'}[r(s, a^{rec}, a^{actual}) + \gamma \, Q_*(s', arg\, max_{a'} Q_*(s', a'; \theta); \theta') + \epsilon]$ where $\epsilon \in \text{clip}(\mathcal{N}(0, \sigma))$ [9].

3.2 Multi-layer Hierarchical RL Agent-based Framework

The overall objective of the t-RELOAD based offline user-centric learning is to correctly identify whether the user played with less, same or more than last ordinality (LO) and the correct amount of push required to maximize engagement intensity. Specifically, we want to associate class -1 for playing less (than last ordinality), 0 for playing same, 1 for 1 step more, 2 for 2 step more, and so on. In order to accurately identify the relevancy of push classes and improve the overall model performance, we utilize a hierarchical agent based network as shown in Fig. 4. Hierarchical, cascading multi-layer framework in classification often outperforms simple single layer structure due to the fact that single layer structure needs to learn the complex approximate function in a single neural network, however in hierarchical structure the complexity is divided across multiple layers [4, 22].

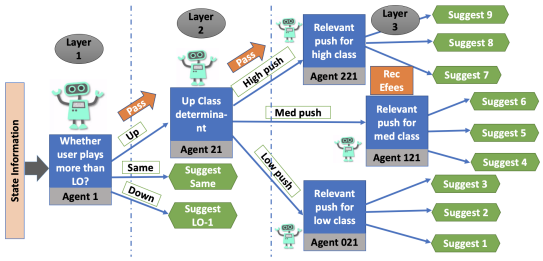
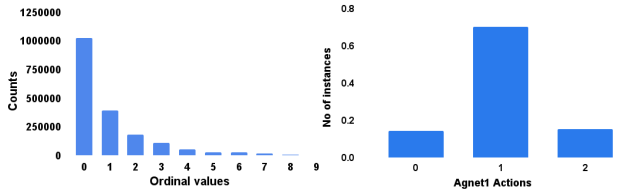
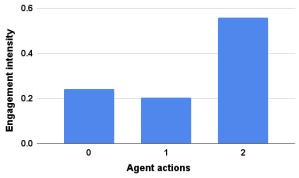


Figure 4: Multi-layer hierarchical framework for offline t-RELOAD analysis

As shown in Fig. 4, agents are divided into three layers. For a given state information, agent-1 in the first layer determines whether the user plays less (down), same, or more (up) than the last ordinality. If the user is going up, agent-21 in layer 2 is responsible identifying whether the user required high push, medium push or low push. In practice, assume the current-wallet balance is very high and the user played with very low ordinality, in such a scenario agent-1 recommends the user to play high ordinality in the next game. The third layer finally determines the amount of push needed.



(a) Ordinality distribution (b) Agent1-action distribution



(c) Distribution of normalized engagement intensity

Figure 5: Distribution of data with importance of each classes

3.3 Data

During the *t*-RELOAD based offline training process, 6-months of recommendation is utilized containing the recommended ordinality and selected ordinality for each user-state information. Along with the users' demographic information, the data also include past selection choices for each ordinality type and game type as shown in Table 1. In Table 1, the column '*Rational*' shows the reasoning behind including the corresponding feature. Note that, our objective is to find out whether the user is going to play at a higher/same or lower ordinality than the last ordinality in the platform.

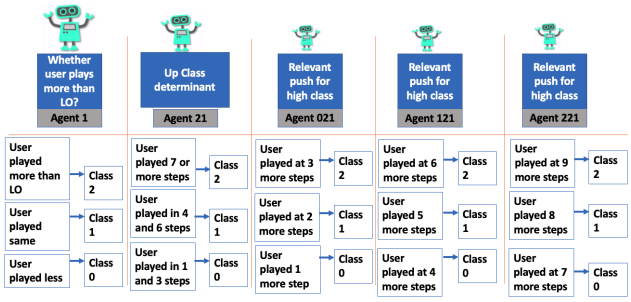


Figure 6: Details of class labels for all agents

The size of the dataset is (1855131, 124) where total number of user-ids are 181970 with average number of recommendations per user is around 20. Fig. 5a shows the distribution of the Agent1 actions which follows an exponential decay pattern because selecting a higher ordinality is very less likely than selecting a lower ordinality. Moreover, the users mostly prefer sticking to the same ordinality game (refer Fig. 5b). Fig. 5c shows the normalized engagement intensity contribution for each agent-1 actions, which is maximum for action-2. Hence, identifying class-2 is more important than identifying lower ordinal classes for agent-1.

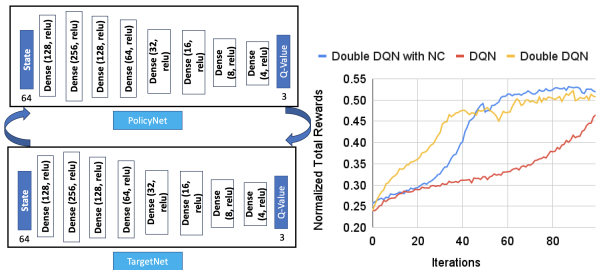
3.4 Performance Evaluation

3.4.1 Offline Performance Comparison. We first compare the performance of Agent-1 for various offline-based deep reinforcement learning algorithms (such as multi-arm bandit [24], DQN [5, 31], Double-DQN [26], and Double-DQN with Noise Clipping [9]). Fig. 7a shows the architecture of offline DQN with input feature set as shown in Table 1 and output as (0,1,2) (refer Fig 6) with loss function as Eq. 4.

Similar architecture is adopted for Double DQN [26] and Double DQN with Noise Clipping [9]. The y-class labels of the data-set contains the actual ordinal step-increase with respect to last ordinality ("LO" in Fig. 6). For example, if the LO is 3 and current ordinality is 4, then the implicit push is 1. However, agent-1 is responsible to detect if the user is going up, same or low. Hence, if the actual step-increase is more than 0, the y-class label for agent-1 is 2, else if the actual step-increase is equal to 0 then y-class label for agent-1 is 1, otherwise 0. Similarly the y-class labels for all other agents are given in Fig. 6.

Algorithm 1 Offline training

1: Weights are initialized randomly for both policy(θ) and target network(θ')
2: Collect batches of data
3: **for** $batch = 1, 2, \ldots, N$ **do**
4: **for** each new training sample (s, a^{actual}, s', r) **do**
5: Select an action either by exploration or exploitation
6: Set a^{rec} = action from explore/exploitation
7: Estimate reward $r(s, a^{rec}, a^{actual})$ following Eq. 2
8: Estimate target $y = [r(s, a^{rec}, a^{actual}) + \gamma Q_*(s', arg\,max_{a'} Q_*(s', a'; \theta); \theta') + \epsilon]$
9: **end for**
10: Train policy network using Eq. 4
11: After some iteration, copy weights from policy network to target network
12: **end for**



(a) Model architecture (b) DRL Performance

Figure 7: Performance of Offline RL-based techniques

During each iteration, the model is trained for many epochs and the loss between actual and predicted Q-values are minimized. Fiq. 7b shows the normalized total rewards during each iteration. Multiple such experiments are conducted and we provide the average results. The normalized total rewards are estimated as the total

positive rewards (normalized across number of samples) the agent-1 receives after each training iteration. If the recommended action (a^{rec}) matches with the actual action, the agent receives positive reward else negative reward (refer Eq. 2). Note that, compared to simple DQN, Ddqn and Ddqn-with Noise-clipping increases the normalized rewards, and hence *t*-RELOAD based offline training process involves training each agent using double-DQN with noise clipping (refer Subsection 3.1). The *t*-RELOAD based offline algorithm is shown in Algo 1, where the following gradient for both policy and target network is used for training:

$$\nabla_\theta L(\theta) = E_{s,a^{rec},r,s'}[[r(s,a^{rec},a^{actual}) + \gamma\, Q_*(s', arg\,max_{a'} Q_*(s',a';\theta);\theta') + \epsilon] - Q_*(s,a^{rec};\theta))]\nabla_\theta Q_*(s,a^{rec};\theta)$$

The target y is estimated as $y = E_{s'}[r(s,a^{rec},a^{actual}) + \gamma\, Q_*(s', arg\,max_{a'} Q_*(s',a';\theta);\theta') + \epsilon]$ and the policy network is trained with Eq. 4 (refer Algo. 1). Note that while estimating the $Q_*(s', arg\,max_{a'} Q_*(s',a';\theta);\theta')$, we utilize the target network. Finally, the weights of the target network are updated after a few iterations for better convergence. The hyper-parameters adopted during model training are given in Table 2.

3.4.2 Online Performance of t-RELOAD. As mentioned earlier the objective of the *t*-RELOAD is to generate the recommendation in a way such that to move the users from cluster-1 to cluster-2 by recommending correct ordinalitys (refer Fig. 2). In order to address the question "*how can we transfer the objective from relevancy to maximizing engagement?*", *t*-RELOAD utilizes turbo-reward during online-exploration. Initially, the weights are same as offline-learning. However, during online-training *t*-RELOAD updates the weights of the DQN-network using incremental reinforcement learning, where we gradually modify the weights of the policy and target network based on the rewards received from turbo-reward function (for a given learning rate) (refer Algo. 2). The learning rate determines the rate at which the network's weights are updated. The turbo-reward function is designed based on the platform objective (or engagement) which is incorporated by introducing factors such as number-of-games (within each session) and inter-session-duration within the reward function. Note that, if number-of-games increases and inter-session-duration decreases, the overall engagement increases. Hence, the action lead to maximizing (long-term) engagement are given higher rewards.

Algorithm 2 Online training

1: Set ϵ as learning rate
2: Weights are copied using Algo 1 for both policy and target network
3: **for** each $batch = 1, 2, \ldots, N$ **do**
4: Generate actions either by exploration/exploitation
5: Estimate platform-centric reward r (e.g., engagement)
6: Update the weights of policy network using r and ϵ
7: After some iteration, copy weights from policy network to target network
8: **end for**

The overall deployment architecture is shown in Fig. 8 contains (i) Front-end: responsible to generate user-interface and ensures

Table 2: Details of Hyper-parameters

Name	Value	Name	Value
Learning rate	0.00005	#iterations	500
Optimizer	Adam	#epochs in each iterations	5
Loss	mean squared error	#layers	9
lambda	0.5	Activation function	Relu
beta	0.1	epsilon_min	0.01
action_size	4	epsilon_decay	0.05
state_size	124	update target weights	once in every 10 iterations
gamma	0.7		

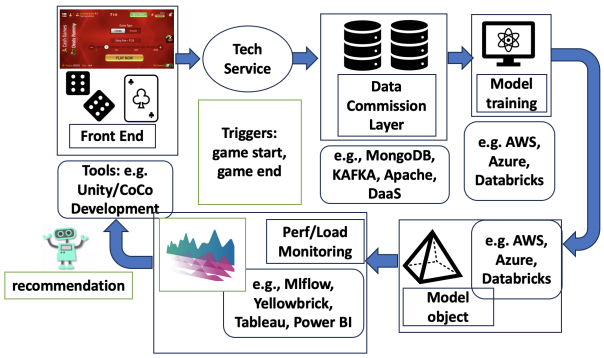


Figure 8: Deployment Architecture

rendering of the recommendations, (ii) Tech service: generate appropriate triggers on game-start or add-cash event to subsequent layers for recommendations, (iii) Data commission layer: utilizes data commission tools such as KAFKA [17] to connect data-streams with ML/AI frameworks and ensures flow of recommendations from ML/AI model to front-end, (iv) Online model training using tools such as AWS [8] to leverage sagemaker-jobs for parallel-training, (v) finally the trained object is located in ML life-cycle platforms such as AWS/Data-bricks for real-time inference to ensure concurrency and redundancy. Moreover, tools such as Millow, Yellowbrick, Tableau [14] are also utilized to support performance monitoring of the model and load monitoring of the instances and AWS/S3 is utilized to store the logs. The model object is updated according to the turbo-reward received using online-exploration and the real-time inference is done using the modified model-object.

The existing RC-platform utilizes an XGBoost [2] based technique for ordinality recommendations. Table. 3 compares the performance of the existing method with *t*-RELOAD. *t*-RELOAD correctly identifies 80%, 66%, and 79% of 0,1, and 2 classes for agent-1, whereas XGBoost identifies 65%, 55%, and 78%, respectively. *t*-RELOAD shows performance improvements when compared with XGBoost. Table 4 shows the overall final performance of *t*-RELOAD at ordinal-level ($C_{i,j}$), where 0 to 9 denotes final predicted ordinals. The confusion matrix is diagonal heavy, which means *t*-RELOAD can identify the ordinal-values correctly. Here, $C_{0,0}$ denotes the users actually chosen ordinal 0, and the model also predicted ordinal 0. Note that, as most of the users prefer staying at the same ordinal (refer Fig. 5b), out-of-which approximately 20-25% of the instances are pushed-up at the next ordinal-level which is consistent with ComParE [20].

Table 3: Performance comparison of *t*-RELOAD and XGBoost

	t-RELOAD			XGBoost		
	0	1	2	0	1	2
Actual	0.80	0.10	0.10	0.65	0.24	0.10
Actual	0.13	0.66	0.20	0.10	0.55	0.34
Actual	0.04	0.15	0.79	0.02	0.20	0.78

Table 4: Final Performance of *t*-RELOAD at ordinal-level

		Predicted									
Actual		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	0	0.73	0.18	0.08	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	1	0.18	0.45	0.33	0.04	0.00	0.00	0.00	0.00	0.00	0.00
	2	0.01	0.14	0.62	0.19	0.04	0.00	0.00	0.00	0.00	0.00
	3	0.00	0.01	0.16	0.53	0.24	0.05	0.01	0.00	0.00	0.00
	4	0.00	0.00	0.01	0.21	0.46	0.24	0.07	0.01	0.00	0.00
	5	0.00	0.00	0.00	0.01	0.23	0.41	0.24	0.08	0.01	0.00
	6	0.00	0.00	0.00	0.00	0.01	0.06	0.54	0.19	0.19	0.01
	7	0.00	0.00	0.00	0.00	0.00	0.01	0.05	0.71	0.16	0.06
	8	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.05	0.82	0.13
	9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.96

4 DISCUSSION AND FUTURE WORK

This framework provides the ability to provide turbo-reward which transforms the network from user-centric to platform-centric while solving cold-start problem. Further, final recommendations can be generated combining both (user and platform) type-of-network. User engagement depends not only on ordinality but also offers, promotions and bonuses available in the platform. Hence, future work involves extending the same framework with incorporating multi-objective for user-journey optimization.

REFERENCES

[1] M Mehdi Afsar, Trafford Crump, and Behrouz Far. 2022. Reinforcement learning based recommender systems: A survey. *Comput. Surveys* 55, 7 (2022), 1–38.

[2] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[3] Ching-An Cheng, Xinyan Yan, and Byron Boots. 2020. Trajectory-wise control variates for variance reduction in policy gradient methods. In *Conference on Robot Learning*. PMLR, 1379–1394.

[4] Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. 2020. Cdt: Cascading decision trees for explainable reinforcement learning. *arXiv preprint arXiv:2011.07553* (2020).

[5] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. 2020. A theoretical analysis of deep Q-learning. In *Learning for Dynamics and Control*. PMLR, 486–489.

[6] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. 2018. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning*. PMLR, 1447–1456.

[7] Scott Fitzgerald. 2019. Over-the-top video services in India: Media imperialism after globalization. *Media Industries Journal* 6, 2 (2019), 00–00.

[8] Chris Fregly and Antje Barth. 2021. *Data Science on AWS*. " O'Reilly Media, Inc.".

[9] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*. PMLR, 1587–1596.

[10] Rong Gao, Haifeng Xia, Jing Li, Donghua Liu, Shuai Chen, and Gang Chun. 2019. DRCGR: Deep reinforcement learning framework incorporating CNN and GAN-based for interactive recommendation. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1048–1053.

[11] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (2012), 1291–1307.

[12] Minghao Han, Lixian Zhang, Jun Wang, and Wei Pan. 2020. Actor-critic reinforcement learning for control with stability guarantee. *IEEE Robotics and Automation Letters* 5, 4 (2020), 6217–6224.

[13] Jiawei Huang and Nan Jiang. 2020. From importance sampling to doubly robust policy gradient. In *International Conference on Machine Learning*. PMLR, 4434–4443.

[14] Bibhudutta Jena. 2019. An Approach for Forecast Prediction in Data Analytics Field by Tableau Software. *International Journal of Information Engineering & Electronic Business* 11, 1 (2019).

[15] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).

[16] Maciej Łuczak. 2016. Hierarchical clustering of time series data with parametric derivative dynamic time warping. *Expert Systems with Applications* 62 (2016), 116–130.

[17] Cristian Martín, Peter Langendoerfer, Pouya Soltani Zarrin, Manuel Díaz, and Bartolomé Rubio. 2022. Kafka-ML: Connecting the data stream with ML/AI frameworks. *Future Generation Computer Systems* 126 (2022), 15–33.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

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

[20] Koyel Mukherjee, Deepanshi Seth, Prachi Mittal, Nuthi S Gowtham, Tridib Mukherjee, Dattatreya Biswas, and Sanjay Agrawal. 2022. ComParE: A User Behavior Centric Framework for Personalized Recommendations in Skill Gaming Platforms. In *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*. 186–194.

[21] Abiodun E Onile, Ram Machlev, Eduard Petlenkov, Yoash Levron, and Juri Belikov. 2021. Uses of the digital twins concept for energy services, intelligent recommendation systems, and demand side management: A review. *Energy Reports* 7 (2021), 997–1015.

[22] Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. 2021. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–35.

[23] Doina Precup. 2000. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series* (2000), 80.

[24] Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning* 12, 1-2 (2019), 1–286.

[25] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[26] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[27] Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudık. 2017. Optimal and adaptive off-policy evaluation in contextual bandits. In *International Conference on Machine Learning*. PMLR, 3589–3597.

[28] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.

[29] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[30] Yisong Yue and Thorsten Joachims. 2009. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 1201–1208.

[31] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1040–1048.

[32] Jichen Zhu and Santiago Ontañón. 2020. Player-centered AI for automatic game personalization: Open problems. In *International Conference on the Foundations of Digital Games*. 1–8.