Debiasing Few-Shot Recommendation in Mobile Games

Lele Cao
caolele@gmail.com
AI R&D, King Digital Entertainment,
Activision Blizzard Group
Stockholm, Sweden

Sahar Asadi
sahar.asadi@king.com
AI R&D, King Digital Entertainment,
Activision Blizzard Group
Stockholm, Sweden

Matteo Biasielli
matteo.biasielli@king.com
AI R&D, King Digital Entertainment,
Activision Blizzard Group
Stockholm, Sweden

Michael Sjöberg
michael.sjoberg@king.com
AI R&D, King Digital Entertainment,
Activision Blizzard Group
Stockholm, Sweden

ABSTRACT

Mobile gaming has become increasingly popular due to the growing usage of smartphones in day to day life. In recent years, this advancement has led to an interest in the application of in-game recommendation systems. However, the in-game recommendation is more challenging than common recommendation scenarios, such as e-commerce, for a number of reasons: (1) the player behavior and context change at a fast pace, (2) only a few items (few-shot) can be exposed, and (3) with an existing hand-crafted heuristic recommendation, performing randomized explorations to collect data is not a business choice that is preferred by game stakeholders. To that end, we propose an end-to-end model called DFSNet (Debiasing Few-Shot Network) that enables training an in-game recommender on an imbalanced dataset that is biased by the existing heuristic policy. We experimentally evaluate the performance of DFSNet both in an offline setup on a validation dataset and online in a real-time serving environment, illustrating the correctness and effectiveness of the trained model.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → *Neural networks.*

KEYWORDS

In-game recommendation, debiasing, mobile game, feedback loop, few-shot recommendation, A/B test

Reference Format:
Lele Cao, Sahar Asadi, Matteo Biasielli, and Michael Sjöberg. 2020. Debiasing Few-Shot Recommendation in Mobile Games. In *3rd Workshop on Online Recommender Systems and User Modeling (ORSUM 2020), in conjunction with the 14th ACM Conference on Recommender Systems, September 25th, 2020, Virtual Event, Brazil.*

1 INTRODUCTION

As smartphones expand the gaming market [21], mobile gaming has become a significant segment of the video game industry. Although recommendation systems such as [12] and [25] are widely adopted in e-commerce, the integration with mobile games is a relatively new area of research. Previous works have mostly focused on recommending game titles to potential players (e.g., [2], [13], [14], [23], and [24]). A few recent works have also explored in-game recommendation [3, 5, 8, 19], however, to the best of our knowledge the large-scale and real-time recommendation of in-game items has not reached its maturity in the industrial scenarios. One of the common business models in modern mobile games is free-to-play where the game can be played free of charge, and monetization occurs through micro-transactions of additional content and in-game items [1]. Therefore, in-game contents are continuously added to the game, which may easily overwhelm the players, causing an increase in churn probability. In-game item recommendation systems help to alleviate this problem by ranking items and selecting the ones that are more relevant to players in order to improve player engagement.

In-game recommender systems utilize user interaction data that describes *historical behavior* and *current context* of individual players to expose each player the right item at the right time. However, despite a few in-game recommendation trials [3, 5, 8, 19] evaluated mostly in an offline and batch fashion, there have not been many successful industrial applications of online in-game recommendation systems. This is mainly attributed to three unique requirements from mobile games:

(1) The recommendation is often calculated on remote servers and delivered to game clients in near-*real-time* with low latency (e.g., within the range of 100 milliseconds). Because of the fast-evolving game dynamics, the *behavior* of players and their *context* keep changing quickly; consequently, the recommendations (calculated from behavior and context data) become outdated easily. As a result, the optimal solution should continuously perform recommendation calculation and always deliver up-to-date prediction when item exposure is triggered. That is why the offline batch recommendation might only provide a suboptimal average policy.
(2) In mobile games, the items to purchase or to play are usually carefully crafted by game designers. To avoid distracting the players with an overloaded small mobile screen, i.e. only a minimal subset (e.g. as small as one to three items, hence termed *few-shot*) of those items is displayed at each exposure occasion. Therefore, the players' experience and

ORSUM@ACM RecSys 2020, September 25th, 2020, Virtual Event, Brazil

(a) LiveOps (dynamic content) (b) Daily gifts

Figure 1: Examples of scenarios where item recommendation can be applied in CCS (1a) and CCSS (1b).

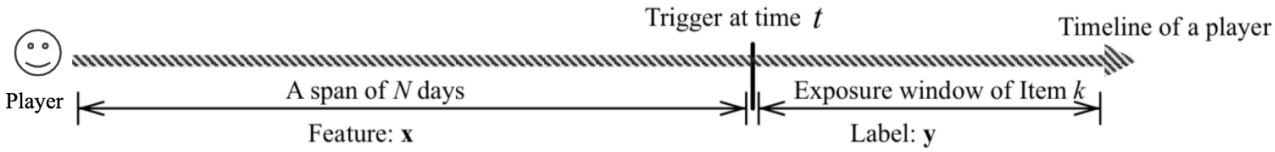


Figure 2: Illustration of collecting features and label for one player and one exposure trigger.

behavior will be more sensitive to recommendations than in e-commerce applications where a large number of items can be displayed at a time, leading to a stronger *direct feedback loop* [18].

(3) The carefully designed in-game items are often exposed to players following a pre-defined *heuristic policy* that contains a set of hard-coded rules regulating the particular item(s) to be exposed to player group(s) with certain attributes (e.g., IF a player has won more than z games in a day, THEN show item A instead of item B). The recommendation models largely fall into two main categories: Supervised Learning [10] and Reinforcement Learning [7], both of which work only when item exposure can be randomly explored. However, the existing heuristic policy heavily biases the experience of the players and hence the dataset, which makes it extremely difficult to train an unbiased model directly. Collection of randomized data is not often trivial. In many cases, stakeholders prefer to continue working with reasonably good heuristics which might not be optimal but avoid any potential business risks caused by randomization.

Our literature survey (till the date when this paper is written) shows that none of the related works [3, 5, 8, 19] managed to simultaneously address the three aforementioned challenges. The contributions of this paper is threefold: (1) we propose a Debiasing Few-Shot Network (DFSNet) that enables training an in-game item recommender merely using heavily biased and imbalanced data, (2) we discuss an approach to benchmark the trained DFSNet offline and (3) we put the model live to recommend items in real-time, and demonstrate how to monitor, evaluate, interpret, and iterate on DFSNet in a controlled A/B test framework.

2 THE PROPOSED APPROACH

There are many scenarios where in-game recommendations could be applied. In Figure 1, we exemplify a couple of examples for two of the King[1] games: Candy Crush Soda Saga (CCSS) and Candy Crush Saga (CCS). We notice that some occasions allow only one item (a.k.a. *one-shot*) to be exposed at a time such as Figure 1b, while others (e.g., Figure 1a) can display a few more (a.k.a. *few-shot*) items. Items can have no values specified as shown in these two examples, or have values attached. To simplify the introduction of our method and experiments, we use the one-shot setup where only one item k with value v_k can be recommended upon each trigger of an exposure opportunity. We will show that our approach can be easily applied to scenarios with few-shot exposure and items with no values. The overall optimization objective is to maximize the expected value of the potentially clicked items. In this section, we present a walk-through of our debiasing few-shot recommendation approach.

2.1 Features and Label

Each sample in the dataset corresponds to a complete item exposure event triggered at time t for a player. As illustrated in Figure 2, we calculate the player features, noted as $\mathbf{x} \in \mathbb{R}^D$, using historical data of the last N days before the time t. The D-dimensional features fall into two categories: *behavioral* (e.g., the total number of game rounds played) and *contextual* (e.g., the latest inventory status). In addition, at time t, the exposed item k (following a heuristic policy) is recorded. Within the time window that item k is exposed, we log if the player eventually clicks on it or not, which is treated as a

[1] https://king.com

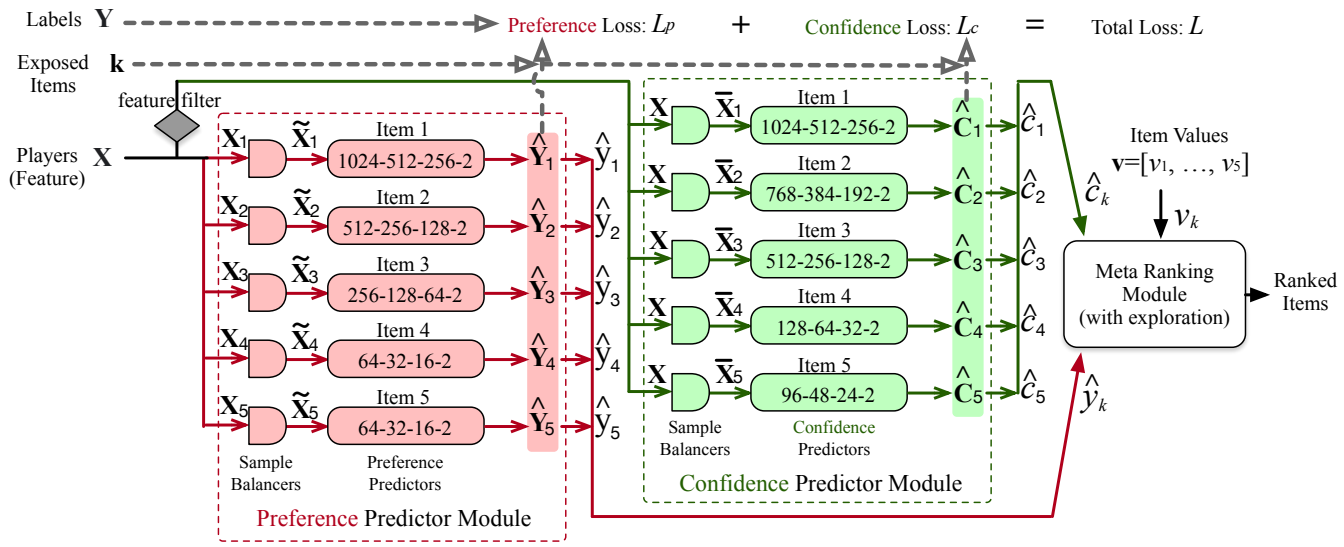


Figure 3: The architecture of DFSNet (best viewed in color). In the example shown here, we set K, the number of items, to 5.

binary label $\mathbf{y} \in \{0,1\}^2$. The raw dataset is extremely biased due to the presence of the existing heuristic policy, and it is imbalanced concerning the label and distribution of exposed item types.

2.2 The End-To-End Model: DFSNet

In this section, we propose to train a debiasing few-shot network, DFSNet, to perform a few-shot in-game recommendation using only the heavily biased and imbalanced dataset. The goal of DFSNet is to rank K items where the k-th item has a value v_k for a player (that is represented by a D-dimensional feature vector $\mathbf{x} \in \mathbb{R}^D$), in order to maximize the expected click value. As shown in Figure 3, DFSNet consists of three modules: *preference predictors*, *confidence predictors*, and *meta ranking*. The training is conducted in a mini-batch fashion; the input is a matrix $\mathbf{X} = \{\mathbf{x}^{(m)}\}_{m=1}^{M} \in \mathbb{R}^{M \times D}$, where M is the number of samples in each mini-batch and D is the number of features in each sample. For the sake of conciseness, we use the general terms $\mathbf{x}$ and $\mathbf{X}$ to denote any sample and mini-batch, respectively.

2.2.1 Preference Predictors. For a player $\mathbf{x}$, the *preference predictor module* (cf. the red dashed bounding box in Figure 3) predicts the probability $\hat{y}_k$ that player $\mathbf{x}$ will click item k if this item is exposed. During training, the mini-batch $\mathbf{X}$ is firstly divided into K subsets (noted as $\mathbf{X}_1, \ldots, \mathbf{X}_K$), so that the k-th subset $\mathbf{X}_k \in \mathbb{R}^{M_k \times D}$ only contains the M_k players that were exposed to the k-th item. As a result, each item k has its own architectural branch, which sequentially propagates $\mathbf{X}_k$ through a sample balancer and a preference predictor, and eventually yields the click/non-click probability $\widehat{\mathbf{Y}}_k \in \mathbb{R}^{M_k \times 2}$.

Since the number of clicked items usually represents a small fraction of the entire exposed item set, there are far more negative samples ($\mathbf{y}$=[1,0]) than the positive ones ($\mathbf{y}$=[0,1]) in $\mathbf{X}_k$. In many recommendation methods such as [16, 20], positive and negative samples are manually balanced by random sampling, and the rich information embodied by negative samples is lost. We propose a *minority subsampling* technique (cf. *sample balancers* in Figure 3) to automatically balance $\mathbf{X}_k$ during training. We split $\mathbf{X}_k$ into two sets $\mathbf{X}_k^+$ and $\mathbf{X}_k^-$, where $\mathbf{X}_k^+$ contains all M_k^+ positive samples, $\mathbf{X}_k^-$ contains the M_k^- negative samples, and $M_k^+ \ll M_k^-$. We randomly pick (without replacement) $max(min(M_k^+, M_k^-), 1)$ samples from $\mathbf{X}_k^-$ and put them in a set $\widetilde{\mathbf{X}}_k^-$. We construct the balanced mini-batch subset $\widetilde{\mathbf{X}}_k$ by

$$\widetilde{\mathbf{X}}_k = \mathbf{X}_k^+ \cup \widetilde{\mathbf{X}}_k^-, \text{ where } \widetilde{\mathbf{X}}_k \in \mathbb{R}^{[\max(\min(M_k^+, M_k^-), 1) + M_k^+] \times D}. \quad (1)$$

This minority subsampling balancer is conceptually similar to the negative sampling in [22] that enforces each mini-batch to contain only one positive sample; therefore, our approach results in a far more balanced mini-batch. Similarly to negative sampling, in minority subsampling, $\mathbf{X}_k$ must contain at least one sample of the minority class.

The output of the Sample Balancer from the k-th branch (i.e. $\widetilde{\mathbf{X}}_k$ in Figure 3) is then fed to a *preference predictor* implemented with a 4-layer Deep Neural Network (DNN) binary classifier. The ELU (Exponential Linear Unit) activation function [9] is applied to all hidden layers except the last one, which is a softmax layer with two neurons. Dropout could be applied to avoid overfitting, yet we choose to empirically scale the first three layers of the k-th DNN proportionally (from a base architecture `32-16-8`) to the exposure ratio of the corresponding item: $M_k / \sum_{k=1}^{K} M_k$. The loss to optimize the preference predictor module, L_p, is formulated as

$$L_p = \frac{1}{2K} \sum_{k=1}^{K} \left[\frac{1}{\widetilde{M}_k} \sum_{m=1}^{\widetilde{M}_k} \left\| -\mathbf{y}_k^{(m)} * \log\left(\hat{\mathbf{y}}_k^{(m)}\right) \right\|_1 \right], \quad (2)$$

where "$*$" represents the element-wise multiplication, $\widetilde{M}_k = \max(\min(M_k^+, M_k^-), 1) + M_k^+$ is the number of samples in $\widetilde{\mathbf{X}}_k$, the notation $\mathbf{y}_k^{(m)}$ is the label (one-hot encoded vector) of the m-th

sample in $\widetilde{\mathbf{X}}_k$, and $\hat{\mathbf{y}}_k^{(m)}$ is the predicted probability vector for the same sample.

2.2.2 Confidence Predictors. To explicitly model the bias from the pre-dominant heuristic, we introduce a *confidence predictor module* (cf. the green dashed bounding box in Figure 3) to DFSNet. The *confidence predictors* estimate the probability c_k that player $\mathbf{x}$ has recently been exposed to the item k. Thus, $\hat{c}_k$ can be treated as an approximation of the confidence we have for the predicted click probability $\hat{y}_k$. Similar to preference predictors, this module also employs K branches (for K items), each of which has a sample balancer and a DNN binary classifier.

The mini-batch input $\mathbf{X} \in \mathbb{R}^{M\times d}$ is fed into the sample balancer of each branch indiscriminately. To prevent the confidence predictors from simply memorizing the heuristic rule and lose the generalization capability, it is important to remove the features (if any) that are used in heuristic policy, hence $\mathbf{X}$'s second dimension d may be smaller than the original dimension D. The sample balancer in this module first divides $\mathbf{X}$ into two subsets $\mathbf{X}_k \in \mathbb{R}^{M_k\times d}$ and $\mathbf{X}_{\neg k} \in \mathbb{R}^{(M-M_k)\times d}$, where $\mathbf{X}_k$ only contains M_k players exposed to item k, and $\mathbf{X}_{\neg k}$ has the rest $M - M_k$ samples. Due to the pre-existing heuristic policy, the item exposure was not randomized, making the size of $\mathbf{X}_k$ and $\mathbf{X}_{\neg k}$ imbalanced. To that end, we need a sample balancer for each branch to produce a balanced mini-batch $\overline{\mathbf{X}}_k$ using

$$\overline{\mathbf{X}}_k = \begin{cases} \mathbf{X}_{\neg k} \cup \left(\mathbf{X}'_k \in \mathbb{R}^{[\max(M-M_k,1)]\times d}\right) & , M_k \geq M - M_k \\ \mathbf{X}_k \cup \left(\mathbf{X}'_{\neg k} \in \mathbb{R}^{[\max(M_k,1)]\times d}\right) & , M_k < M - M_k \end{cases}, \quad (3)$$

where $\mathbf{X}'_k$ and $\mathbf{X}'_{\neg k}$ are obtained via *minority subsampling* (without replacement); specifically, the former term contains $\max(M - M_k, 1)$ samples randomly selected from $\mathbf{X}_k$, and the latter contains $\max(M_k, 1)$ randomly picked samples from $\mathbf{X}_{\neg k}$. $\overline{M}_k$ denotes the number of samples in $\overline{\mathbf{X}}_k$, hence $\overline{\mathbf{X}}_k \in \mathbb{R}^{\overline{M}_k\times d}$.

$\overline{\mathbf{X}}_k$ is then fed to a *confidence predictor* implemented in the same way as in preference predictors except that each DNN is scaled proportionally to factor $\overline{M}_k / \sum_{k=1}^{K} \overline{M}_k$. The loss L_c to optimize this module has a similar form to Equation (2):

$$L_c = \frac{1}{2K} \sum_{k=1}^{K} \left[\frac{1}{\overline{M}_k} \sum_{m=1}^{\overline{M}_k} \left\| -\mathbf{c}_k^{(m)} * \log\left(\hat{\mathbf{c}}_k^{(m)}\right) \right\|_1 \right], \quad (4)$$

where $\mathbf{c}_k^{(m)} \in \{0, 1\}^2$ is the constructed confidence label specifying if the m-th player/sample in $\overline{\mathbf{X}}_k$ actually saw item m or not, and $\hat{\mathbf{c}}_k^{(m)}$ is the predicted confidence probability vector for the same sample. The preference and confidence predictors are jointly optimized with a total loss $L = L_p + L_c$.

2.2.3 Meta Ranking. During model serving/prediction, the sample balancers will be omitted, meaning that the input $\mathbf{X} \in \mathbb{R}^{M\times D}$ (representing M players) will be directly fed to DNNs in all branches, in order to simultaneously generate real-valued preference ($\widehat{\mathbf{Y}} \in [0, 1]^{M\times K\times 2}$) and confidence ($\widehat{\mathbf{C}} \in [0, 1]^{M\times K\times 2}$) predictions. The second values in the last dimension of $\widehat{\mathbf{Y}}$ are the click probabilities, while those of $\widehat{\mathbf{C}}$ are the confidence levels expressed as probabilities. To simplify the discussion that follows, we will use $\hat{y}_k$ and $\hat{c}_k$ to denote, respectively, the predicted click probability and confidence level of item k for an individual player $\mathbf{x}$. The *meta ranking* module (cf. the right-most box in Figure 3) ranks items by calculating a *propensity score* R_k for each item using three factors: $\hat{y}_k$, $\hat{c}_k$, and v_k; the term v_k is the value of item k, which is usually predefined. We propose a piece-wise formula for computing R_k:

$$R_k = \begin{cases} \frac{v_k}{\min(\mathbf{v})} & , (\hat{c}_k \geq \frac{1}{2}) \wedge (\hat{y}_k \geq \frac{1}{2}) \\ \hat{c}_k \cdot \hat{y}_k \cdot \frac{v_k}{\max(\mathbf{v})} & , \text{otherwise} \end{cases}, \quad (5)$$

where functions $\min(\mathbf{v})$ and $\max(\mathbf{v})$ respectively return the minimum and maximum element from vector $\mathbf{v} = [v_1, \ldots, v_K]$. Generally speaking, R_k is obtained by calibrating $\hat{y}_k$ with $\hat{c}_k$ and v_k, so that random exploration data would not be mandatory (at least initially). To the best of our knowledge, only [17] discussed the possibility of removing item position bias using an adversarial network, yet our approach manages to deal with much stronger item exposure bias using a more explainable strategy; and explainability is valued highly in industrial environments [11]. If v_k is not available (Figure 1b and 1a), we can adapt Equation (5) to

$$R_k = \begin{cases} \hat{y}_k & , (\hat{c}_k \geq \frac{1}{2}) \wedge (\hat{y}_k \geq \frac{1}{2}) \\ \hat{c}_k \cdot \hat{y}_k & , \text{otherwise} \end{cases}. \quad (6)$$

We can conveniently assume that K items are already sorted by their values $\mathbf{v}$, hence the propensity scores $\mathbf{R} = [R_1, \ldots, R_K]$ are also sorted accordingly. An overly drastic change of item exposure (e.g. a player who used to see item 1 according to the heuristic which suddenly gets item K from a newly deployed recommender system) may undermine the player experience and game ecosystem. To avoid that situation, it is a good practice to enforce a *heuristic deviation threshold* (noted as $k_s \in \{1, \ldots, K-1\}$) in the online production environment. Specifically, we mask R_k with

$$\widetilde{R}_k = \begin{cases} 0 & , |k_h - k| > k_s \\ R_k & , \text{otherwise} \end{cases}, \quad (7)$$

where k_h is the item from the pre-existing heuristic policy. With $\widetilde{\mathbf{R}} = [\widetilde{R}_1, \ldots, \widetilde{R}_K]$, both one-shot and few-shot in-game recommendation are possible. When recommending items based on $\widetilde{\mathbf{R}}$, we can sometimes choose to apply ϵ-greedy to slowly accumulate more diversified data for follow-up model iterations.

3 EXPERIMENTATION AND EVALUATION

We apply DFSNet to a real-time item recommendation scenario for the CCSS game. There is a total of five items (K=5) in this scenario, yet only one item k can be shown on the mobile screen when the player triggers the exposure event. The item k has a value v_k. Items are sorted by value in an ascending order, i.e., $v_1 < v_2 < v_3 < v_4 < v_5$. If a player clicks on the exposed item k, a value v_k will be added to the game ecosystem; and we choose to maximize the value of the clicked item. The details of the concrete use case and items are considered to be sensitive proprietary information and therefore anonymized in this paper.

As illustrated in Figure 4, the raw dataset is collected (cf. Section 2.1) using a Flink[2] based stateful streaming platform [6]. The collected dataset contains approximately 22 million samples, each of which has $D = 48$ features. We apply different transformations

[2]https://flink.apache.org

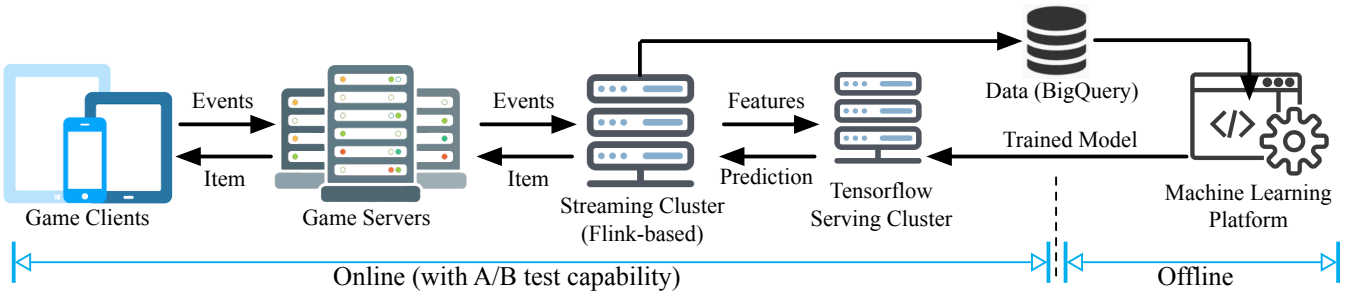


Figure 4: High-level system topology of offline model development and online model serving.

(e.g., min-max, z-score, and logarithmic) to numerical features and perform either one-hot encoding or embedding to categorical features. The dataset pre-processing and model development is carried out on a machine learning platform developed by King.

We will present both offline (training and evaluation) and online (serving and monitoring) evaluation of DFSNet in the following sections. DFSNet is implemented in Tensorflow[3]; the preference and confidence DNNs are scaled as depicted in Figure 3. The training is carried out with *Adam* optimizer [15], using 70,000 steps and a mini-batch size of 2,048. The learning rate is initialized to 5×10^{-3}, and then it exponentially decays to 2×10^{-6}. During serving, we set $k_s = 2$ in Equation 7 to obtain one single item to recommend.

3.1 Offline Performance: Model Training and Validation

To perform offline model evaluation, we create a *validation dataset* (noted as $\mathbf{U} = \{\mathbf{x}^{(u)}\}_{u=1}^{U}$) by randomly selecting 1% data from the raw dataset (thus $U \approx 0.22$ million), and use the rest for training.

3.1.1 Generalization evolvement during training. We evaluate the performance of the current model on the validation dataset during the training. Since the datasets are highly imbalanced, accuracy is not an informative metric to monitor during training. We also find that recall and precision are having a hard time competing with each other (showing no clear trend) during the training, hence not ideal for monitoring the training performance. AUC-ROC (Area Under the Curve of Receiver Operating Characteristics), on the other hand, is a stable metric that reliably tells how much the model is capable of distinguishing between classes; therefore, the evolution of per-item AUC-ROCs (see Figure 5) indicates how the generalization ability of the model improves during the training process. At the end of the training, we also measure the recall and precision for each item, which are visualized as red bars in Figure 12.

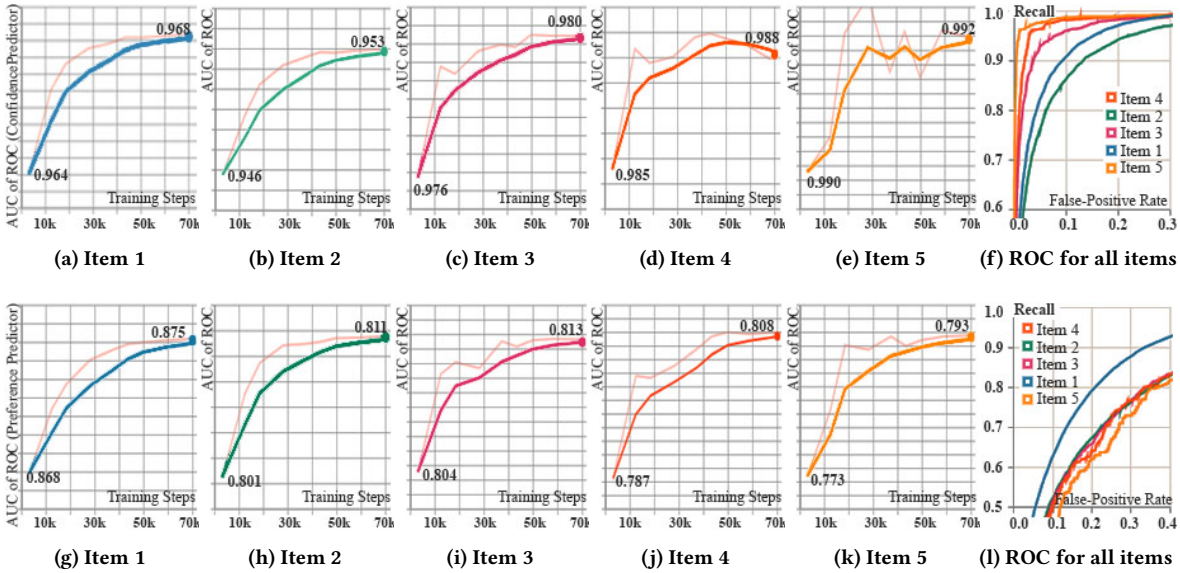
3.1.2 Policy change quantization: heuristic vs. DFSNet. We use the trained DFSNet to obtain predictions on the validation dataset. We first measure the overall change of item exposure distribution. The results are reported in Figure 6a. In our experiment, we observed no significant change in item allocation for players due to the strong confidence constraint imposed, yet there is a slight shift towards the higher-valued items. The ratio of players that see a different item (than heuristic) is about 7.4%. To decompose the policy change, we illustrate, in Figure 6b, a Policy Transition Matrix (PTM), where each cell at position (i, j) indicates the ratio of players who were supposed to get item j, according to heuristic policy, but are now exposed to item i according to DFSNet. It can be seen that the diagonal has the majority of the unchanged exposures, and each row largely follows a truncated normal distribution.

3.1.3 Distribution of preference and confidence predictions. For each sample in the validation dataset, DFSNet produces ten probabilities: five click probabilities ($\hat{y}_1$ to $\hat{y}_5$) and five confidence probabilities ($\hat{c}_1$ to $\hat{c}_5$). Figure 7 visualizes $\hat{y}_k$ and $\hat{c}_k$ jointly to answer four questions:

(1) Does $\hat{y}_k$ reflect the low click ratio of item k? The five red area plots on the diagonal are the distributions of $\hat{y}_k$, all of which show that clicking tends to be a rare event.
(2) Does $\hat{c}_k$ match the exposure ratio of item k? The five green bar plots on the diagonal represent the distributions of $\hat{c}_k$; the majority of exposures come from item 1, which coincides with the heuristic item exposure distribution in Figure 6a.
(3) Does $\hat{y}_k$ show general item preference? The lower triangular portion has pair-wise scatter plots of click probabilities. Each data point in the plot for item i and j has a coordinate of $(\hat{y}_i, \hat{y}_j)$, thus if the point is below the line of $\hat{y}_i$=$\hat{y}_j$, the corresponding player prefers item i over j, and vice versa. To examine the general trend, we fit linear models (red straight lines going through the original points) for pair-wise plots. We observe that in average, players prefer items with lower values.
(4) Can DFSNet be confident with multiple items for the same player? The upper triangular portion in Figure 7 contains pair-wise scatter plots of confidence probabilities $\hat{c}_k$. Every point in the plot for item i and j is located at $(\hat{c}_i, \hat{c}_j)$. Intuitively, implied by Equation (5), the points (representing players) in the green shaded areas are likely eligible to more than one item.

3.1.4 Best-effort estimation of recall, precision, and uplifts. On the offline validation dataset $\mathbf{U} \in \mathbb{R}^{U \times D}$, it is impossible to measure the "quality" of a recommendation that is different than what was actually exposed; hence, a sub-optimal solution is to create a subset (from $\mathbf{U}$) containing only the players for whom both DFSNet and the heuristic policy recommended the same item. We use $\mathbf{U}' \in \mathbb{R}^{U' \times D}$ ($U' < U$) to denote that subset. On that subset, we measure per-item recall and precision for preference predictors (cf. the red

[3]https://www.tensorflow.org



(a) Item 1 (b) Item 2 (c) Item 3 (d) Item 4 (e) Item 5 (f) ROC for all items

(g) Item 1 (h) Item 2 (i) Item 3 (j) Item 4 (k) Item 5 (l) ROC for all items

Figure 5: Training performance of confidence predictors (a-f) and preference predictors (g-l). (a-e),(g-k) The AUC of ROC measured on the validation dataset for each item during training. (f),(l) The item ROCs on the validation dataset at the end of the training.

(a) Comparison of item distribution. (b) PTM.

Figure 6: Comparison of heuristic and DFSNet policy on the validation dataset: (6a) overall item exposure distribution and (6b) policy transition matrix (PTM). Each row in PTM adds up to 1.0.

bars in Figure 12). To provide uplift baselines of average click rate ($\frac{\#\_clicked\_items}{\#\_items}$) and click value ($\frac{total\_value\_of\_clicked\_items}{\#\_clicked\_items}$), we calculate both metrics for both the heuristic and DFSNet policies. The results are presented in Table 1. Offline uplifts will be then compared with the ones obtained during online model serving (cf. Section 3.2.3).

3.2 Online Performance: Real-Time Serving and Monitoring

After the DFSNet model is trained and validated in an offline environment, it is then deployed in a Tensorflow Serving[4] cluster. As illustrated in Figure 4, a prediction client (sharing the entire feature collection logic described in Section 2.1) is also deployed on the streaming cluster. To validate the online performance of the DFSNet model, we run an A/B test on a small fraction of players on CCSS. For each game player in the test group, the prediction client makes a request to the DFSNet prediction service (in real-time) as soon as any pre-defined triggering event emerges. In the life cycle of a real-time recommendation system, it is often required to iterate on the model serving periodically (cf. Figure 8 for an example of two serving iterations) to incorporate bug fixes or new models trained on more recent data.

During online serving, we track several metrics (aggregated into temporal windows of 5 minutes) to monitor the key system performance, some examples of which include model response time, model exceptions, and model raw output distribution. The definition of those system metrics remains the same for different recommendation models. These metrics are indicators of the system health, and therefore, they play a critical role in the validity of the model. To monitor model performance, we log all features, predictions, and labels in a BigQuery[5] database, and visualize them

[4]https://www.tensorflow.org/tfx/guide/serving

[5]https://cloud.google.com/bigquery

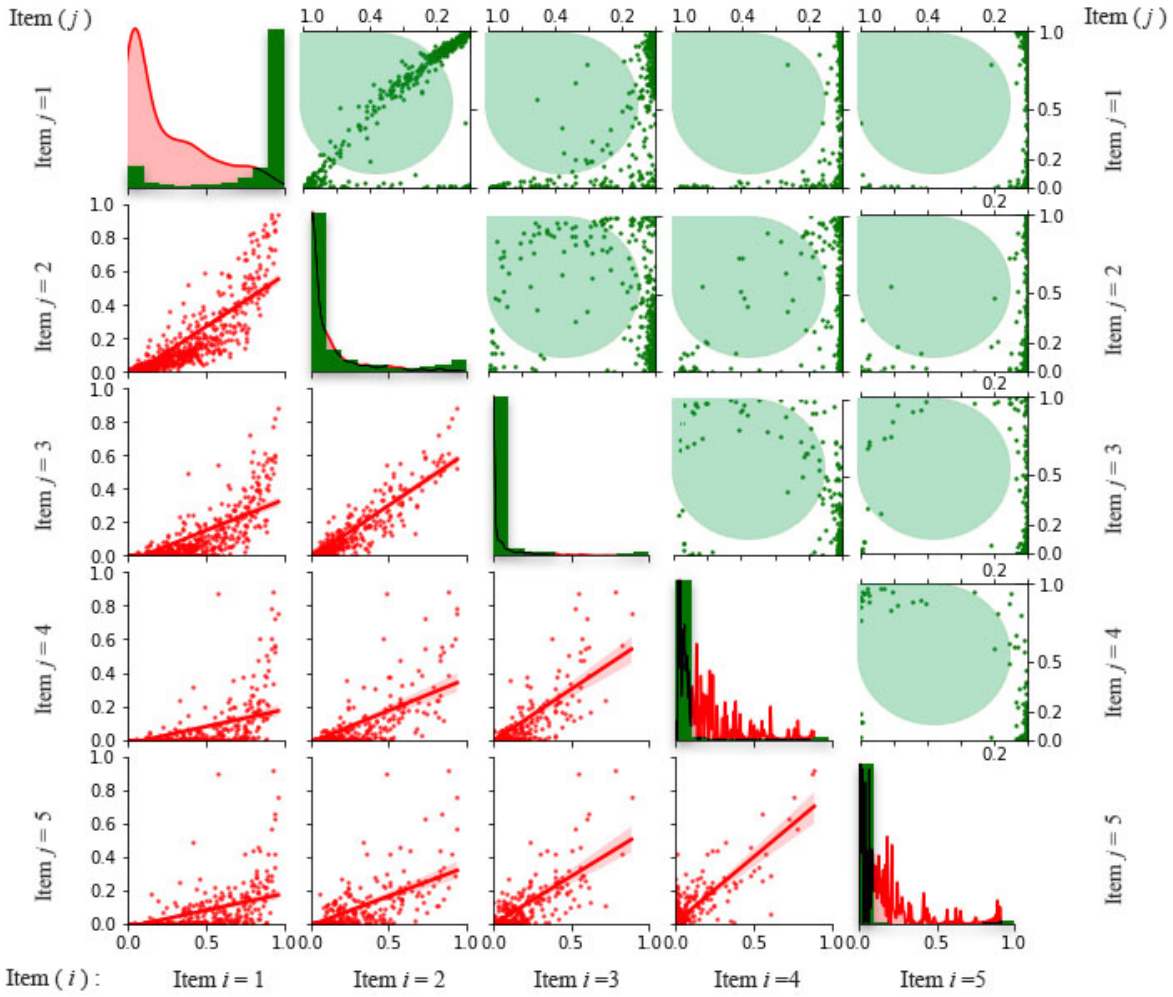


Figure 7: The visualization of preference (red plots) and confidence (green plots) predictions.

Table 1: The estimated click rate and value obtained from the validation dataset. For the DFSNet policy, if item k (with a predicted click probability $\hat{y}_k$) is ranked highest, and if $\hat{y}_k > 0.5$, we assume the player will click the recommended item k, generating a value v_k.

Metrics (obtained on validation dataset)	Heuristic Policy	DFSNet Policy	Uplift (%)
Average click rate	0.1022	0.1312	28 %
Average click value	1.52	1.92	26 %

in a dashboard that is updated on hourly basis. We will hereafter emphasize our online evaluation on several key perspectives, all of which are adapted from the monitoring dashboard.

3.2.1 Heuristic deviation trend. The foremost questions to answer about model performance are twofold: (1) what the scale of the model impact is and (2) how this impact evolves along the timeline. To answer these two questions, we illustrate the overall heuristic deviation trend of two adjacent model serving iterations in Figure 8, where the red curve shows the ratio of players (in the DFSNet A/B test group) for which DFSNet and the heuristic policy recommended different items. As reported in Section 3.1.2, this ratio is approximately 7.4% (represented by a straight green line) when measured on the offline validation dataset. So, the expectation is that the ratio of impact should reach around 7.4%; this trend can be clearly seen in each model serving iteration. However, some input features need individual players to respond to certain game components, which takes about three days in the use case discussed here; and

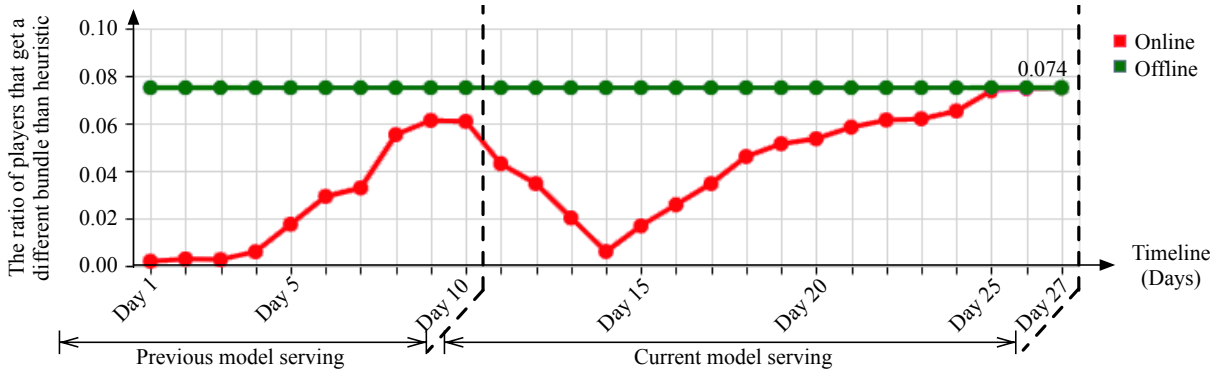


Figure 8: The ratio of players that see different items from the heuristic policy: a span of 27 days containing two model serving iterations.

items are served using the heuristic policy to players that still have incomplete features. As a result, for each model serving iteration, the ratio always starts from a fairly low point before reaching 7.4%. Furthermore, between two subsequent iterations, the ratio drops, in this use case, for three days while rebuilding features before picking up the ascending trend again. We believe that the daily monitoring of the heuristic deviation trend helps tracking the scale of the model impact effectively.

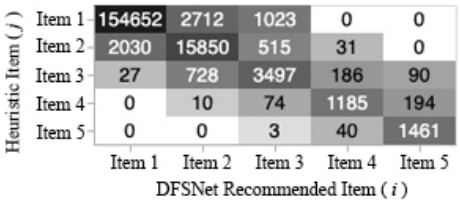
3.2.2 Player Transition Matrix (PTM). To have a better insight of the underlying changes contributing to the overall model impact, we break down the impact analysis further using PTMs. Here, a PTM is a 2D matrix that puts players into each grid cell according to how their experience changed from the default heuristic policy (rows) to the model policy (columns). Figure 9 summarizes the results over a period of 14 days since `Day11` (cf. Figure 8) and illustrates four different PTMs (a-d) that enable model impact decomposition from four different perspectives:

(a) Grid cell (i, j) denotes the number of players that are now exposed to the item i but would have originally got item j. The gray-scale background of Figure 9a is also used in (c) and (d).
(b) Calculated by dividing each number in (a) by the sum of the corresponding row. It should have the highest ratio values on the diagonal, as obtained in the offline evaluation and reported in Figure 6b. Because the presence of incomplete features leads to a policy fallback (to the heuristic, as mentioned in Section 3.2.1), the results presented in Figure 9b are more conservative than those shown in Figure 6b.
(c) To inspect how the moved players impact the click probability on PTM, we calculate the click percentage (of item i) for the player cohort in each grid cell, and then overlay the percentage values on top of Figure 9a's gray-scale background, resulting in Figure 9c. The blue values in the diagonal cells are the click percentage for the control group. Each off-diagonal cell $(i, j)_{i \neq j}$ contains the players that are moved from cell (j, j) to cell (i, i). Intuitively, we expect the model to guarantee the click percentage in $(i, j)_{i \neq j}$ to be larger than either percentage values in cell (i, i) or (j, j); we use red boxes to highlight the cells that fail to satisfy that expectation. In practice, it is acceptable to have a few red boxes as long as most of the densely populated cells satisfy the expectation.
(d) The PTM in Figure 9d principally serves the same purpose as Figure 9c except that it computes the average click value of item i instead.

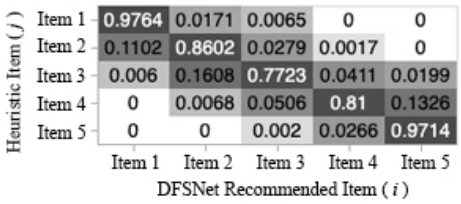
Based on the PTM analysis, we expect to see an improvement in the user engagement with the examined game feature compared to using the heuristic solution. Further analysis of the PTM can help us to understand better user behaviors. In our analysis, we used an eight-dimensional space to describe players behavior. Figure 10 shows different user behavioral patterns (in the form of radar charts) on top of the grayscale background from Figure 9a. The KPI calculation and the actual values are considered to be sensitive proprietary data and therefore removed from the charts.

3.2.3 Uplifts of click ratio, count, and value. The previous sections presented a drill-down process of analyzing the model behavior in comparison with the heuristic policy. We now zoom out and compare the item click dynamics with the control group. We choose to focus on the accumulated 14-day uplift of three metrics: click count, click ratio, and click value (these metrics are defined in Section 3.1.4). The online uplift is computed by subtracting the metrics (normalized by the population size of A/B test groups) of the control group from the ones of the DFSNet group. Hence, uplift can have both positive and negative values. All uplift values are considered sensitive proprietary data and therefore scaled.

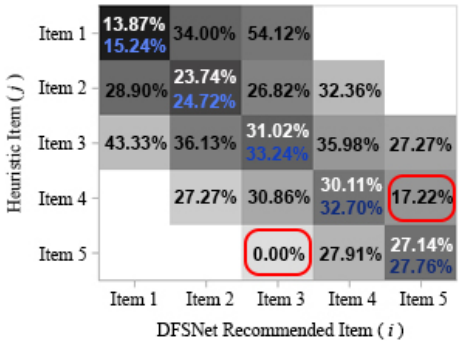
Figure 11a shows that DFSNet group is losing clicking counts on items 1 and 4 while gaining more click counts on other items; therefore, the players moved away from those buckets are mostly item clickers, and they bring more absolute click counts for items 2, 3, and 5. However, the click ratio of item 5 is reversed in Figure 11b, which is a consequence of the lower click ratio in the player cohort moved to item 5. Figure 11c shows the uplift of accumulated click value for each item. We observe that the loss of click value (from items 1 and 4) is compensated by the increased click value for items 2, 3, and 5, leading to a net positive value uplift (approximately +0.71% over the control group). As a result, the offline uplift estimations (Table 1) are overly optimistic compared to that measured online.



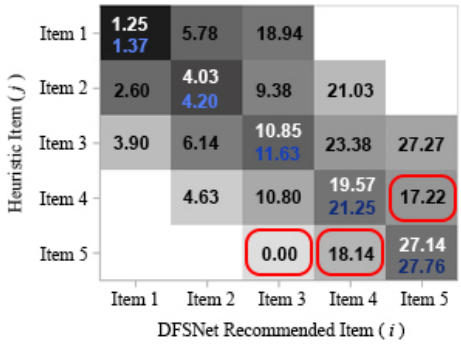
(a) Number of players "moved" from j to i.



(b) Ratio version of (a): each row sums to 1.



(c) Click percentage of item i for players who should see item j according to the heuristic.



(d) Average click value of item i for players who should see item j according to the heuristic.

Figure 9: The player transition matrices: each grid cell at coordinate (i, j) indicates (9a-9b) the volume, (9c) average click ratio, and (9d) average click value for players who would have been exposed to item j according to the heuristic policy and got recommended item i from DFSNet. The blue numbers along the diagonal in 9c, 9d denote the corresponding statistics obtained from the control group in A/B test.

3.2.4 Iterating on the DFSNet model. The dataset used to train DFSNet is highly biased due to the pre-existing heuristic rules. Our approach achieves debiasing by incorporating confidence predictors, thus demonstrating a mild impact of less than 8% (cf. Figure 8). Nonetheless, that impact continuously changes the players' experiences, which nudges the input feature distributions around. This creates a *direct feedback loop*, which gradually compromises the generalization and discriminative capability of the model being served. It is a form of analysis debt [18], in which it becomes increasingly difficult to predict the behavior of a given model before it is released. Iterating the model periodically using more recently collected data can reduce the intensity of the feedback loop. However, we need a metric to determine the time to train a new model. Accuracy is not an option since the logged data is heavily imbalanced (click event is rare), and we care more about correctly predicting the clicking events. Practically, AUC-ROC (cf. Section 3.1) and response distribution charts [4] can also be used to monitor the feedback loop, yet they are not as sensitive as precision and recall. We propose to monitor the precision and recall of preference predictors (i.e., the green bars in Figure 12) to identify the "right" moment for model iteration.

In Figure 12, the red bars represent the precision and recall estimated using a subset of the validation dataset (as explained in Section 3.1.4), while the green bars in Figure 12a and 12b are respectively precision and recall calculated 14 days after the model got deployed (a snapshot on `Day24` in Figure 8). We observe that online precision and recall reach much higher values than the offline reference initially, hence we argue that the offline evaluation tends to underestimate the true values of precision and recall. Figure 12c and 12d reflect the situation four days later. The trend is clear: in four days, both precision and recall have declined significantly; when the majority of green bars go under the red bars, it is probably the time to retrain/finetune the DFSNet using fresher data.

4 CONCLUSION AND PERSPECTIVES

In-game recommendation aims to enable providing more relevant items to each player. The in-game recommendation use cases usually allow exposing only a few items at a time; thus, change in the choice of items can have a large impact on game dynamics leading to a short feedback loop. In addition, player preferences change quickly due to the change in the game dynamics and player context. As a result, the model gets outdated sooner in real-time prediction. In-game item exposures are mostly dominated by some hand-crafted heuristics, which heavily bias the data, and randomized exploration to train an unbiased recommendation model is usually not favored by stakeholders. We propose DFSNet that enables training an unbiased few-shot recommender using only the biased and imbalanced data.

During training, AUC-ROC is a stable indicator of the modelâĂŹs generalization ability. We demonstrate several ways to estimate the model performance offline on a validation dataset. We also evaluate the online DFSNet performance in an A/B test. We start with monitoring the overall model impact by looking at the heuristic deviation trend. Then, we further decompose the model impact using

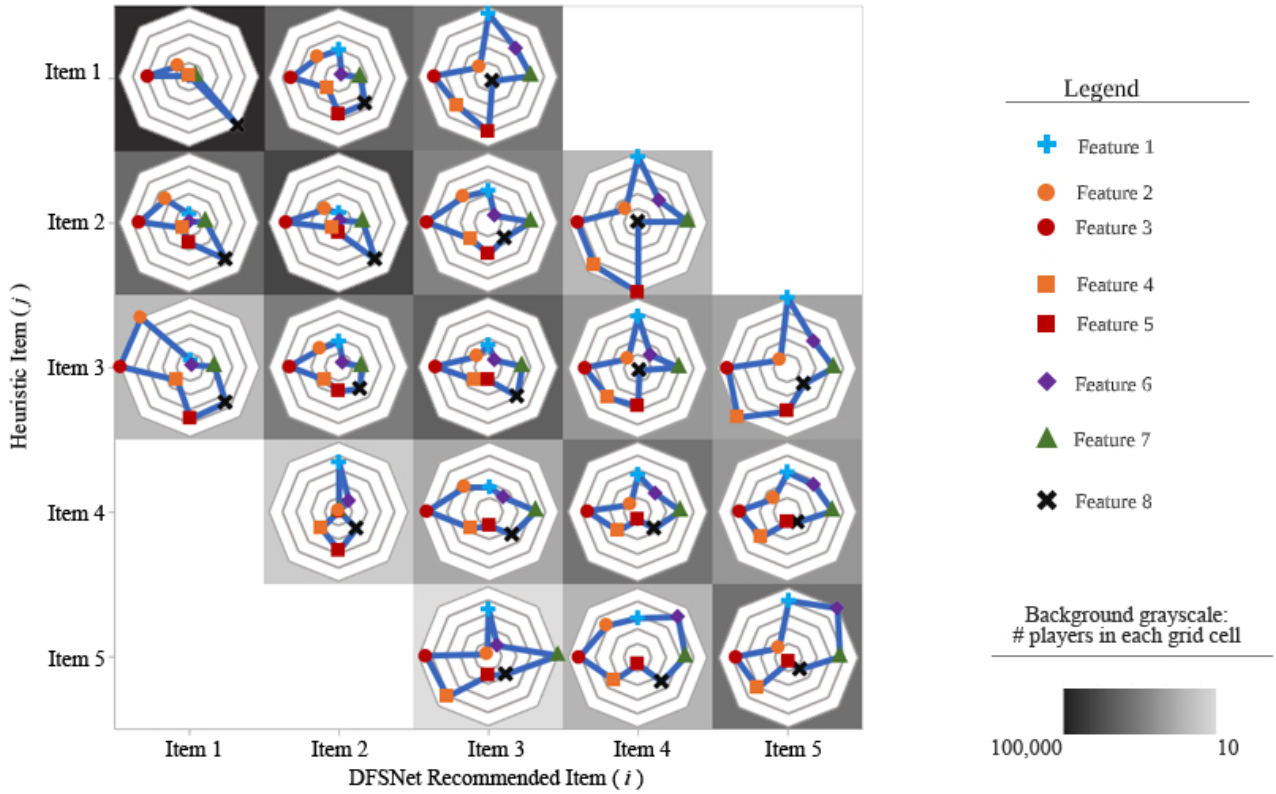
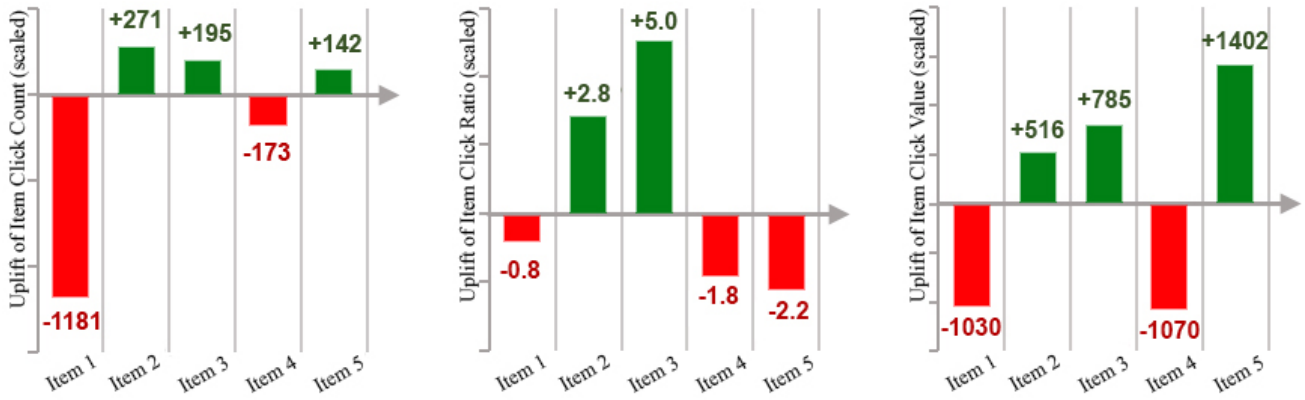


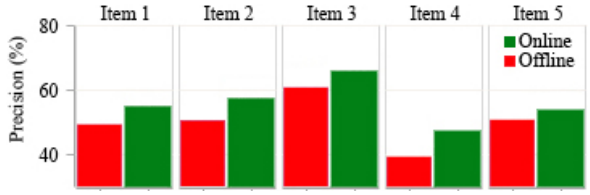
Figure 10: The overlay of player behavioral features (in an eight dimensional space visualized in radar charts) over the policy transition matrix; each player dimension is calculated over a period of the past N days and then rescaled to the range of [0,1].



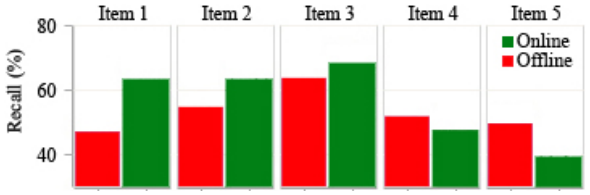
(a) Uplift of click counts. **(b) Uplift of click ratio.** **(c) Uplift of click value.**

Figure 11: The accumulative scaled uplift (DFSNet over control A/B test group) of (11a) click count, (11b) click ratio, and (11c) click value per exposed item type.

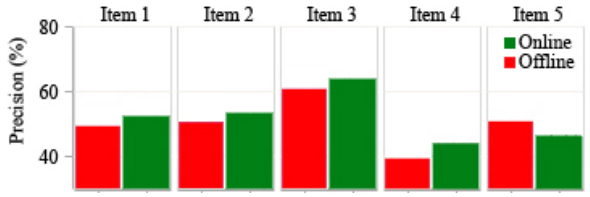
PTMs. We carried out data analysis to understand user behaviors and discern the key factors causing players to be exposed to a different item than the heuristic recommendation. This work proposes a solution to address the problem of bias and imbalanced data in the domain of in-game recommender systems. We suggest offline and proxy metrics as a way to have an estimate of model online performance. We discuss and showcase the challenges of an online solution in an A/B test. The comparison between the control and DFSNet test groups show a net +0.71% uplift of click value, which is less optimistic than the best-effort offline estimation. We show that



(a) Precision of click prediction up till Day24.



(b) Recall of click prediction up till Day24.



(c) Precision of click prediction up till Day28.

(d) Recall of click prediction up till Day28.

Figure 12: The online precision and recall of DFSNet preference predictions compared to the offline best-effort estimates.

continuous comparison of offline and online precision/recall can help determine the appropriate time to retrain the model. Further analysis is required before putting the proposed solution live. In the scenario presented in this paper, we chose the click-through rate as one of the evaluation metrics, which is widely adopted in e-commerce. However, this metric might not be a good proxy for business metrics for an in-game recommender system. Future work will explore the choice of metrics that constitute a better proxy of the model's online performance.

In addition, future work includes (1) designing long-term labels that better approximate the business targets, (2) explicitly modeling the interactions between different in-game features to eliminate the implicit feedback loop, and (3) replacing model iteration with online reinforcement learning approaches.

REFERENCES

[1] Kati Alha, Elina Koskinen, Janne Paavilainen, Juho Hamari, and Jani Kinnunen. 2014. Free-to-play games: Professionals' perspectives, In DiGRA Nordic: Proceedings of the 2014 International DiGRA Nordic Conference. *Proceedings of nordic DiGRA* 11, 1–14.

[2] S. M. Anwar, T. Shahzad, Z. Sattar, R. Khan, and M. Majid. 2017. A game recommender system using collaborative filtering (GAMBIT). In *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. 328–332.

[3] Vladimir Araujo, Felipe Rios, and Denis Parra. 2019. Data mining for item recommendation in MOBA games. In *Proceedings of the 13th ACM Conference on Recommender Systems*. Association for Computing Machinery, New York, NY, USA, 393–397.

[4] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 2019. 150 Successful Machine Learning Models: 6 Lessons Learned at Booking.Com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Anchorage, AK, USA) *(KDD '19)*. Association for Computing Machinery, New York, NY, USA, 1743–1751. https://doi.org/10.1145/3292500.3330744

[5] P. Bertens, A. Guitart, P. P. Chen, and A. Perianez. 2018. A Machine-Learning Item Recommendation System for Video Games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, Maastricht, The Netherlands, 1–4.

[6] Paris Carbone, Stephan Ewen, Gyula Fóra, Seif Haridi, Stefan Richter, and Kostas Tzoumas. 2017. State management in Apache Flink®: consistent stateful distributed stream processing. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1718–1729.

[7] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. 2019. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) *(WSDM '19)*. Association for Computing Machinery, New York, NY, USA, 456–464. https://doi.org/10.1145/3289600.3290999

[8] Zhengxing Chen, Christopher Amato, Truong-Huy D Nguyen, Seth Cooper, Yizhou Sun, and Magy Seif El-Nasr. 2018. Q-deckrec: A fast deck recommendation system for collectible card games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, IEEE, Maastricht, The Netherlands, 1–8.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:1511.07289 [cs.LG]

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[11] Krishna Gade, Sahin Cem Geyik, Krishnaram Kenthapadi, Varun Mithal, and Ankur Taly. 2019. Explainable AI in industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, ACM, New York, NY, USA, 3203–3204.

[12] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time personalization using embeddings for search ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Association for Computing Machinery, New York, NY, USA, 311–320.

[13] Rama Hannula, Aapo Nikkilä, and Kostas Stefanidis. 2019. GameRecs: Video Games Group Recommendations. In *Welzer T. et al. (eds) New Trends in Databases and Information Systems. ADBIS*. Springer, Cham, Switzerland.

[14] JaeWon Kim, JeongA Wi, SooJin Jang, and YoungBin Kim. 2020. Sequential Recommendations on Board-Game Platforms. *Symmetry* 12, 2 (2020), 210.

[15] Diederick P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, Yoshua Bengio and Yann LeCun (Eds.). San Diego, CA, USA, 13 pages.

[16] Jianxun Lian, Fuzheng Zhang, Xing Xie, and Guangzhong Sun. 2018. Towards better representation learning for personalized news recommendation: a multi-channel deep fusion approach. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 3805–3811. https://doi.org/10.24963/ijcai.2018/529

[17] John Moore, Joel Pfeiffer, Kai Wei, Rishabh Iyer, Denis Charles, Ran Gilad-Bachrach, Levi Boyles, and Eren Manavoglu. 2018. Modeling and Simultaneously Removing Bias via Adversarial Neural Networks. arXiv:1804.06909 [cs.LG]

[18] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. MIT Press, Cambridge, MA, USA, 2503–2511.

[19] Rafet Sifa, Raheel Yawar, Rajkumar Ramamurthy, Christian Bauckhage, and Kristian Kersting. 2020. Matrix-and Tensor Factorization for Game Content Recommendation. *KI-Künstliche Intelligenz* 34, 1 (2020), 57–67.

[20] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. DKN: Deep knowledge-aware network for news recommendation. In *Proceedings of the 2018 world wide web conference*. ACM, New York, NY, USA, 1835–1844.

[21] Robert Williams. 2020. *Mobile games sparked 60% of 2019 global game revenue, study finds.* Mobile Marketer. Retrieved January 2, 2020 from https://www.mobilemarketer.com/news/mobile-games-sparked-60-of-2019-global-game-revenue-study-finds/569658/

[22] Chuhan Wu, Fangzhao Wu, Mingxiao An, Jianqiang Huang, Yongfeng Huang, and Xing Xie. 2019. NPA: Neural news recommendation with personalized attention. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, 2576–2584.

[23] Hsin-Chang Yang and Zi-Rui Huang. 2019. Mining personality traits from social messages for game recommender systems. *Knowledge-Based Systems* 165 (2019), 157–168.

[24] Hsin-Chang Yang, Cathy S Lin, Zi-Rui Huang, and Tsung-Hsing Tsai. 2017. Text mining on player personality for game recommendation. In *Proceedings of the 4th Multidisciplinary International Social Networks Conference*. Association for Computing Machinery, New York, NY, USA, 1–6.

[25] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In *Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, New Orleans, Louisiana, USA, 4564–4571.