Received May 12, 2021, accepted June 2, 2021, date of publication June 7, 2021, date of current version June 16, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087406

DARES: An Asynchronous Distributed Recommender System Using Deep Reinforcement Learning

BICHEN SHI, ELIAS Z. TRAGOS, MAKBULE GULCIN OZSOY, RUIHAI DONG, NEIL HURLEY, BARRY SMYTH, AND AONGHUS LAWLOR
Insight Centre for Data Analytics, University College Dublin, Dublin 4, D04 V1W8 Ireland

Corresponding authors: Bichen Shi (bichen.shi@insight-centre.org) and Elias Z. Tragos (elias.tragos@insight-centre.org)

This work was supported in part by the Science Foundation Ireland through the Insight Centre for Data Analytics under Grant SFI/12/RC/2289_P2, and in part by Samsung Electronics Company, Seoul, Republic of Korea.

ABSTRACT Traditional Recommender Systems (RS) use central servers to collect user data, compute user profiles and train global recommendation models. Central computation of RS models has great results in performance because the models are trained using all the available information and the full user profiles. However, centralised RS require users to share their whole interaction history with the central server and in general are not scalable as the number of users and interactions increases. Central RSs also have a central point of attack with respect to user privacy, because all user profiles and interactions are stored centrally. In this work we propose DARES, an distributed recommender system algorithm that uses reinforcement learning and is based on the asynchronous advantage actor-critic model (A3C). DARES is developed combining the approaches of A3C and federated learning (FL) and allows users to keep their data locally on their own devices. The system architecture consists of (i) a local recommendation model trained locally on the user devices using their interaction and (ii) a global recommendation model that is trained on a central server using the model updates that are computed on the user devices. We evaluate the proposed algorithm using well-known datasets and we compare its performance against well-known state of the art algorithms. We show that although being distributed and asynchronous, it can achieve comparable and in many cases better performance than current state-of-the-art algorithms.

INDEX TERMS Recommender systems, reinforcement learning, distributed learning, click through ratio.

I. INTRODUCTION

Recommender Systems (RS) have lately attracted increased attention for their ability to provide personalised services to users in many domains such as movies, music, hotels, restaurants, jokes, news, health and fitness. RS applications use machine learning techniques to train recommendation models. However, the training is traditionally performed on centralised servers. Training of these global models is performed using as input rich user profiles, which are computed from users' data and interactions. When new items are added into the system or new data are produced through the continuous interaction of users with the system, existing models have to be retrained with great effort to improve their performance. In most case, this is very complex and time consuming activity which becomes more complex as the number of users and items increases through time.

Centrally computed RS models are usually able to achieve high accuracy because they are trained with all the available interaction information. Additionally, the centralised systems have a global view of the profiles of all users which helps accuracy. However, centralised RSs suffer from both scalability and privacy issues [29], [39]. When serving recommendations, the centralised server has to serve all requests for recommendations and send out all recommendations from a central point. The model's computational effort also increases non-linearly with the number of users, i.e., as having larger state and action space [15]. This increases the time between refreshes of the recommendations served, so the harder it is for the RS to provide "fresh" recommendations for trending items.

Privacy is also an issue with centralised RSs because all user profiles and the whole history of user interactions are

The associate editor coordinating the review of this manuscript and approving it for publication was Theofanis P. Raptis.

centrally stored and exploited to build the global recommendation models [38]. This information can be hacked from the outside or inappropriately accessed from within the RS service provider. For example, by analysing the news articles that a user reads, an intelligent system can predict the political views of that user. This also contradicts with the European Union's recent General Data Protection Regulation (GDPR), which introduces strict new regulations on the collection and processing of user data, mandating i.e. ''data minimisation'' and ''collection limitation''. Perturbing users' ratings or using sophisticated cryptographic tools may resolve the problem [6]. However these approaches are limited either because they can reveal the interaction between users and items or they require a high degree of cooperation among users [38].

The need for distributed, scalable and privacy-enhanced recommendation models represents a new opportunity for RS research, by highlighting the need for going beyond traditional approaches [6], [38]. In this work we address the issues of recommendation latency, bias towards more trending items and the central storage of data. We propose a **d**istributed **a**synchronous deep **re**inforcement learning (RL) based recommender **s**ystem (DARES). The asynchronous nature of the proposed framework lies in the asynchronous advantage actor-critic model (A3C) [27], while it also borrows concepts from federated learning (FL) [8], [20], [26].

The proposed algorithm uses a combination of on-device, local recommendation models and a complementary global model similar to A3C (See Figure 1). Inspired by ideas from Federated Learning (FL), the local recommendation models are copies of the global model. The local models are used for computing the gradients of the losses using the user-item interactions of a single user which are locally stored on the user's device. In contrast to FL which computes the global model by averaging the model parameters received from the users, in DARES the global recommendation model is actually trained on the server, by applying the loss gradients which are asynchronously communicated from the local devices. We demonstrate that our DARES model can deliver recommendation results that outperform the current state-of-the-art centralised and distributed algorithms. This work is an extension of the work [37] presented in the RecSys 2020 REVEAL workshop, presenting more details about the method and more results to prove the performance of the model.

In the remainder of this paper, we discuss the technical details of our approach and argue its benefits compared to alternative approaches. Furthermore, we demonstrate how these benefits are available without compromising performance. We make use of standard test datasets to compare our model against a number of benchmark algorithms. Finally, we argue that the approach is well suited to challenging, dynamic, high-throughput recommendation settings, such as news recommendation, where users have diverse and dynamic interests and where the half-life of a recommended item can be significantly truncated compared to more conventional domains.

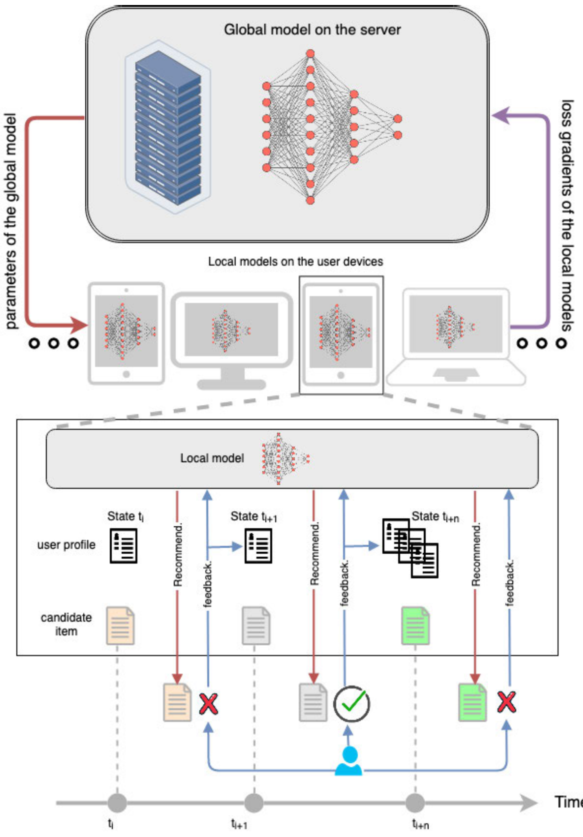

FIGURE 1. Overview of the privacy-preserving distributed deep reinforcement learning (DARES) framework.

II. RELATED WORK

We explore the related work in three domains: 1. Reinforcement Learning-based recommender algorithms, 2. Distributed and federated learning-based recommender algorithms and 3. privacy preserving recommender algorithms, and comparison of DARES algorithm with A3C and FL.

A. REINFORCEMENT LEARNING-BASED RECOMMENDERS

Significant effort has been directed to the application of RL techniques to make better recommendation models that are appropriate for online recommendations [21], [59]. There are two families of RL-based recommendation algorithms. The Multi-Armed Bandit (MAB) models and the Markov Decision Process (MDP) models. The former methods, such as [21], [43]–[46], [51], consider only the reward of the current iteration, but not of future iterations. The latter methods model an RS as an MDP with a finite set of states and actions, aiming to find an optimal policy which gives the best action to be performed for each state. MDP methods, such as [12], [13], [25], [32], [35], [42], [57], [59], aim to increase the overall reward by considering both current and future rewards. Recent RS methods adapt state of the art MDP-based RL methods to make recommendations in various domains, such as retail [13], [47], [54], [57], [58], [60], [62], movies/videos [11], [12], [14], [23], [24], [61], advertisements [52], [53] and news domains [59].

One of the key challenges for RL algorithms in the recommendation domain is that there are often a large number of items to recommend [52], [56]. Another challenge is the dynamic nature of the system where new items are added as recommendation targets when old ones are deleted or become stale. In order to solve this problem, Dulac *et al.* [15] introduced a new policy architecture and extended the Deep Deterministic Policy Gradient (DDPG) algorithm [22], Choi *et al.* [14] introduced a bi-clustering technique which models the RS as a grid-world game and Chen *et al.* [12] adapted the REINFORCE algorithm [49] for top-k recommendation for a system with extremely large action and state spaces, e.g., Youtube. Chen *et al.* [11] proposed Tree-structured Policy Gradient Recommendation (TPGR) algorithm to improve recommendation performance and to avoid inconsistency of DDPG related to the continuous representation of discrete actions. Zheng *et al.* [59] proposed DRN, which adapts Deep Q-Learning (DQN) [28] algorithm for news recommendation, and modelled the states and actions as the feature representations of the users and items respectively, which enables the system to scale up and learn more effectively. Chen *et al.* [13] proposed Robust-DQN, which applies stratified sampling replay and approximates regretted reward to improve the reward estimation in dynamic environments. They evaluated their RL method on a commercial tip recommendation system which has a highly dynamic environment. Zhao *et al.* [57] proposed their DEERS framework which uses DQN and integrate both negative and positive items to their RL-based RS method. They evaluated their method on a commercial real-world RS system. They also have explored making list-wise [58] and page-wise [54] recommendations using RL on a commercial real-world RS system. Also, they proposed DeepChain [55] which is a multi-agent RL approach that jointly optimises multiple recommendation strategies while sharing the same memory of users' historical data.

Zou *et al.* [60] proposed FeedRec which utilised RL for RS and evaluated their proposed method on a commercial platform. Their proposed framework consists of an LSTM based network for learning the user behaviours, i.e., policy learning, and a network to simulate the environment. The simulation network is used to assist the first network and to avoid the instability of convergence in policy learning. Later, they proposed Pseudo Dyna-Q (PDQ) [62] which is based on Dyna-Q [30], [40]. PDQ contains an environment simulator (i.e., world model) which is constantly updated as the recommendation policy is updated. The purpose is to avoid instability of convergence and high computational cost and to provide unlimited (simulated) user interactions. Liu *et al.* [24] proposed DRR which adapts DDPG algorithm and aims to explicitly model the interactions between users and items. They represent the states in three different ways which are combinations of the user representation and selected items' representations. Liu *et al.* [23] proposed End-to-end Deep Reinforcement learning based Recommendation (EDRR) framework. EDRR defines RL frameworks in three modules which include an embedding component, a state representation component and a policy component and supervised learning is used to guide the update of the embeddings and proposed three ways of incorporating these embeddings with the other two components. Additionally, there are RL-based recommendation methods which focus on other aspects of recommendations, such as Wang *et al.* [47] focus on explainable recommendations and Zou *et al.* [61] focus on diversification of recommendations.

The research on MDP-based RL methods for RS mainly focuses on how to apply existing methods in RS and how to overcome the state-action space explosion. Most of those methods make use of synchronous techniques. Recently, the Asynchronous Advantage Actor-Critic (A3C) [27] model has been proposed. A3C is an MDP-based RL method which executes in a distributed and asynchronous manner. A3C has been shown to be significantly faster to train in comparison to DQN in [27]. We use ideas from A3C in our proposed DARES framework, because its distributed asynchronous nature best fits to the distributed training scenario that we address.

B. DISTRIBUTED AND FEDERATED LEARNING-BASED RECOMMENDERS

Federated Learning (FL) [8], [20], [26] is a recently proposed approach to training a centralised model where the training data is distributed over a large number of devices (i.e., user mobile devices, clients). During each round in FL, the server selects a subset of clients, and each of these clients uses its local data to update the model. The clients then send model updates to the server, and the server aggregates these models (typically by averaging) to construct an improved global model [20]. This means that the learning process is not asynchronous, since the global server waits for the model updates from sampled clients for a pre-set time and then executes the model aggregation. When model aggregation is complete, the updated global model is sent to all clients simultaneously. Even though it is possible to relax the core FL assumptions and adopt new approaches [18], in this paper we used the standard FL definitions.

There are a number of existing application of FL to the recommendation domain. Chen *et al.* [10] incorporate the meta-learning algorithms with FL. In this approach, instead of a model, a global algorithm (meta-learner) is shared with local devices. After receiving the parameters of the algorithm, the local devices perform model training. Then they execute a test on a query set and upload the test results to the server. The parameter transmission from server to local devices happens synchronously, i.e., in episodes. Experiments were performed on various datasets, including a private, industrial recommendation dataset, and the results show the efficiency of their approach is better than the baseline FL. Ammad *et al.* [2] implement a Collaborative Filtering(CF)-based RS system in a FL setting using implicit feedback data. On local devices, they used an SGD-based matrix factorization method for the calculations. The local devices execute the SGD-based updates and calculate the gradient updates of the model weights, which are then sent back to the server. The server updates the global model according to the gradients collected. The results show that federated CF achieves similar recommendation performance to the standard CF method. Jalalirad *et al.* [17] aim to make rating predictions for an RS system. They use neural networks and execute training in two-phases. In the first phase, namely global training, they follow the classical synchronous FL setting where devices train models locally, share their parameters with global server, the global server aggregates the parameters and sends the updated parameters back to the local devices. In the second phase, namely local training, the devices stop sending their updated parameters back to global server and keep using their locally updated models. Arivazhagan *et al.* [3] focus on the personalisation aspect of users in the federated learning setup. They propose a neural network architecture which is composed of base and personalisation layers. In their approach, the base layers are trained in the base federated learning setting, in which updates from local devices are sent to the global server, the global server aggregates the updates and sends the updated models back to the local devices. The personalisation layers are trained only locally, such that they are not shared with a global server. Experiments were performed on two datasets, one of which is a RS dataset, and show that this approach is better at modelling the personalisation tasks over a baseline FL approach.

Even though these methods use FL settings, none of them are using RL ideas, such that they do not aim to consider the long term reward instead of immediate feedback. Also, unlike our proposed approach, DARES, the above-mentioned methods use only synchronous updates, which may require synchronisation between thousands or millions of devices, something that normally is difficult to do and can induce delays in the training process.

C. COMPARISON OF DARES, A3C AND FEDERATED LEARNING

DARES is a combination of ideas from A3C and Federated Learning. It uses the idea of distributed, asynchronous execution from the A3C algorithm and the idea of keeping the data locally on each device from FL. The latter is achieved by training a single local recommendation model for each user separately. Both DARES and A3C are distributed and asynchronous RL models, i.e., DARES uses A3C as the base framework. However, in the original A3C algorithm the workers (e.g., devices) do not keep their data locally but receive data from a server and normally include data from multiple users. As a result, the original A3C algorithm cannot be directly applied to a user-based recommendation setting, since each *worker* should keep only data from a single user.

Both DARES and FL use local data kept on the user's device. However, there are difference between these two frameworks. Federated learning executes a user sampling step [8], whereas in DARES there is no such sampling process and any client can participate asynchronously as long as she prefers. In federated learning, the global server executes the model update/aggregation step usually by averaging the model parameters sent by the users, and then the clients receive the same global model for the next round of training. However, in DARES the clients send the loss gradients (and not the model parameters) to the server, which then applies the gradients to update the model weights. In federated learning, the execution is usually split into rounds and the global model is sent back to clients at the beginning of each round [8], which means that the model update/aggregation is synchronous. As a result, for the clients to receive the latest model update, they have to wait until the end of the round of iteration so that the global model is averaged and updated on the server. In DARES algorithm the model update/aggregation is asynchronous, which means that when a client sends its local gradients, the server updates the global model and immediately sends the updated global model to the related client. As a result, at any given time the clients can request from the server and get the latest version of the updated model, without having to wait until the end of an iteration round.

In summary, we propose DARES framework for RS by using the ideas from A3C and FL frameworks. As a result, DARES is able to (i) capture the dynamic nature of interactions between users and the recommendation agent, (ii) always provide the latest model to the users for serving them with the best possible recommendations, and (iii) keep the user data locally on the devices.

III. PRELIMINARIES ON MDP-BASED REINFORCEMENT LEARNING

Reinforcement learning (RL) can be modelled as a Markov decision process (MDP), i.e., a tuple (S, A, T, R, γ), in which S is a set of states; A is a finite set of actions; for each action $a \in A$, T_a is a set of state transition probabilities determining how the state updates upon action a, $R_a : S \times S \rightarrow \mathbb{R}$ is the reward obtained when transitioning from state s to s' upon action a; and $\gamma \in [0, 1)$ is a discount factor. For an MDP, we define $\pi : S \rightarrow A$ to be a policy which gives the action to be performed for each state. The value of a policy π is calculated by $V^{\pi}(s) = \mathbb{E}_{\pi}[R_t|s_t = s]$ and $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, where $\mathbb{E}_{\pi}$ is the expected sum of discounted rewards under policy π, t is the current time point and r_{t+k} is the immediate reward at a future time step $t+k$. The objective of an agent in an MDP is to find an optimal policy π^* with the highest value.

Reinforcement learning methods can be divided into three categories:

(i) **Value-based methods:** (e.g., Q-learning [48] and Deep Q-learning [28]) target to maximise a value function in order to find an optimal policy π^*. Normally there are two ways to calculate a value function:

- using only a state $s \in S$ as shown in Equation 1

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi}\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\Big\}, \tag{1}$$

- using a state-action pair (s, a), as shown in Equation 2

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\pi}\Big\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\Big\} \tag{2}$$

The value function can also be approximated by a neural network with parameters θ using either $Q(s, a; \theta)$ or $V(s; \theta_v)$. For example, Q-learning directly approximates the optimal value function $Q^*(s, a) \approx Q(s, a; \theta)$ and learns the parameters by iteratively minimising a sequence of loss functions [27].

(ii) **Policy-based methods:** Policy-based methods such as Policy Gradients [41] or Proximal Policy Optimisation [34] aim to find an optimal policy π^* by calculating the probability of performing an action a when the agent is in a state s. The optimisation of the model parameters can be done directly by gradient ascent on the objective function $J(\pi_\theta)$, without using a value function. Policy-based methods can be applied when the action space is continuous or stochastic. The quality of a policy is typically measured using the total rewards of the episode.

(iii) **Hybrid (Actor-Critic) methods:** Hybrid RL methods are developed as a combination of value-based and policy-based approaches. The hybrid model (which is usually a neural network) is keeping both a policy and an estimation of the value function. The policy $\pi(a_t|s_t; \theta)$ controls the action that the model chooses, while the estimate of the value function $V(s_t, \theta_v)$ (also called the "Critic") provides a measure of the quality of the policy at any given time. An example of the Hybrid methods is the Actor-Critic model, in which the "Actor" is the policy and the "Critic" is the estimate of the value function. The Actor-Critic models train a critic to learn the approximation of the value function at each stage of the episode and do not use the global reward of the episode contrary to the policy based methods. By doing so, it reduces the variance in the training examples, and makes the learning more stable than pure policy based methods, however it introduces bias from value function methods [4], [19], [33].

Asynchronous Advantage Actor-Critic (A3C) [27] is an **Actor-Critic** method, which maintains both a policy $\pi(a_t|s_t; \theta)$ (i.e., the Actor), that controls how the agent behaves, and an estimate of the value function $V(s_t; \theta_v)$ (i.e., the Critic), that measures how good an action is. It utilises an **Advantage** Actor-Critic objective function, i.e., $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$, which calculates the difference between the value of taking an action at a certain state and the value of being in a certain state. In A3C, the Q values are not calculated directly, instead R_t is used as an estimate of $Q(s_t, a_t)$, i.e., $A(s_t, a_t; \theta, \theta_v) = R_t - V(s_t; \theta_v)$. It is **Asynchronous**, because it trains multiple agents in parallel, where each agent has its own copy of the model and the environment that it interacts with. The workers asynchronously share gradients with the central server and their local neural networks are used for thee target task, e.g., prediction, recommendation.

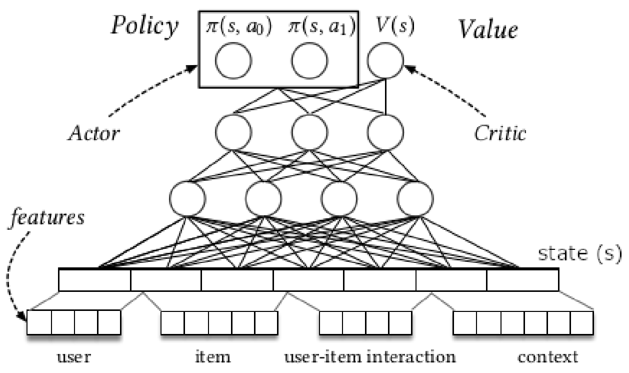

FIGURE 2. The DARES neural network structure.

IV. ASYNCHRONOUS DISTRIBUTED DEEP REINFORCEMENT LEARNING FOR RS

In our proposed method, DARES, we adapt the A3C algorithm to the item recommendation scenario while keeping the data locally on devices, inspiring from FL. In DARES we execute two steps:

- First, we create a state/action representation that is amenable to the constant creation of new items and removal of old items that is encountered in the recommender system world.
- Second, we introduce an Actor-Critic deep neural network (DNN) that can efficiently handle new users and new items.

The overview of the proposed system architecture is given in Figure 1. The local devices get the latest version of the model from the server, propose recommended items to the user, analyse the user feedback and create the model loss gradients which are then sent back to the central server for the model update. More details are given in the sections that follow.

A. PROBLEM STATEMENT

We model the recommendation problem as a classification problem. In its binary setting, our model predicts the class (relevant or not relevant) associated with an item, given a user-item pair. For example, in a recommender system scenario, the binary setting will have two actions for an item: recommend or not recommend. In an explicit rating context, our system can associate the item with one of multiple classes, i.e. five classes corresponding to the points on a 5-star rating system. Hence, we model an action as the classification of a given item for a given user and states to correspond to user-item pairs (Figure 2).

The goal of DARES is to learn a global RL model using the information from the local devices. In order to do so, the framework consists of a global deep neural network (DNN) that exists on a central server and a local DNN (which is a copy of a version of the global model) on each user device. The local deep neural models on the devices may use as input all the available information on the user device, i.e. the interactions of the user or extra information about the user herself (i.e. gender) or the interaction (i.e. location, time/day). This information is used to compute the gradient losses of the local model with respect to the latest model that was received from the server. These gradient losses are then sent to the central server which trains the global model and updates its weights. Then, a new version of the global model is sent back for the next session of the training. Contrary to FL, due to the asynchronous nature of the proposed model, every user will get a (slightly) different version of the global model when they start their local training process. This ensures though that they will have the latest version of the model at any given time and they will always exploit the accumulated knowledge from all the users.

B. THE PROPOSED MDP COMPONENTS

We cast the recommendation problem as a MDP-based RL problem and adapt the A3C algorithm with single-user data per worker, aiming to simulate the distributed training scenario with one user per device. The target for our RL algorithm is to maximise a reward function based on the cumulative feedback of users over time. MDP-based RL systems are known to suffer from state space explosion, i.e., as the number of candidate items increases, the number of states increase exponentially and transition data becomes sparser, leading to scalability issue [15]. Inspired by [59], as a solution, we introduce a state-action mapping and let the local recommendation agent to choose to recommend (or not) a single item to the target/local user. This setting avoids the curse of dimensionality issue and is thus more scalable in production.

We model the MDP components as follows:

States: A state s is represented as a concatenation of user features, user profile history and target item features. More specifically, we use features including user demographic information (e.g., gender, age), item features (e.g., title, description), context features for interactions (e.g., time, location) and user-item similarity features, as shown in Figure 2.

For user, item and context features, we directly used the information provided in the dataset. In order to construct a user history profile, we average the values of the features of the past items they have interacted with. Then, we create the user-item similarity features by computing the cosine similarity of the values of user history profile with the values of the features of the item at hand. In our experiments, for creating the initial user profile history, we select randomly 3 items from the user rating profile and exclude them from training. Each new item that is used for training is then added to the history of the user and the user profile is updated accordingly.

Below, we give an example of the state representation for the ''Adressa'' dataset that we use in the experiment session. As seen in Table 2, the user and item features are of different types, i.e. integer, category, text. The feature extraction process, scales integers to theur unit variance removing their average value, creates a ''one hot encoding'' of the category type features and creates a tf-idf (term frequency–inverse document frequency) vector for the text features keeping the most frequently used terms. As a result, in the ''Adressa'' example, the ''total clicks'' and the ''average active time'' are scaled down as integers, the ''author'', ''device'', ''OS'', ''region'' and ''city'' are one hot encoded and the ''keywords'' and ''title'' of the articles are converted to tf-idf vectors. For creating the user history profile, the vectors of the item features for those items in the user history are averaged. For computing the ''state'' of the model for a target interaction, the cosine similarities of each of the ''category'' and ''text'' features are computed and then concatenated with the ''integer'' features of the current item and the user features.

Our state representation has two key properties:

- we use the features of users and items and not their ids or a list of ids. Therefore, new items can easily be incorporated into our system, as long as the items can be represented with their features. Such modelling of states allows us to avoid hard-coding candidate items, which is beneficial in a dynamic environment such as news recommendation.
- we use the similarity of user-item pairs. As explained previously, the calculated similarity depends on the features of the items with which the user has interacted previously and the features of the candidate item. This kind of representation helps us to handle the dynamic nature of RS, where users' preferences evolve over time, i.e., user-item similarity features update as the user's preference evolve through time.

As seen in Figure 2, the states are used by the local RL agent to find out the best policy (make recommendation or not) for a specific user-item pair.

Actions: The recommendation agent may accommodate either binary actions, such as recommend or not an item, or multi-class actions, such as rating predictions at a scale of 0 to 5. Previous works have defined actions either as the transition among items, a list of recommendation items or directly the items themselves. However, in this paper, we define binary actions, a_0 and a_1, (i.e, recommend/not recommend), as shown in Figure 2. This kind of binary definition supports handling a more dynamic environment, such that we can deal more easily with cold-start (i.e., new) items, which is a common problem in RS domain. Also defining actions in this way allows definition of more advanced actions, e.g., sequences of items, which we want to utilise in our future works.

Rewards: Each target action should get either a positive or a negative reward from the model based on the user feedback. After the recommendation agent suggests an item based on an action a and state s, the user may provide explicit feedback in terms of an explicit rating or implicit feedback such as an interaction with the recommended item. In other words, if the agent recommends an item to a target user and the user interacts with the item, e.g., clicks or rates the item, the recommender agent receives a positive reward. If the user doesn't interact with the item, then the agent can receive either a zero reward or a negative reward as a punishment for recommending a bad action. Similarly, if the recommender agent decides not to recommend an item and the user really does not interact with the item, the agent receives a positive reward. If the user indeed interacts with the non-recommended item, again the agent can get a zero reward or a negative reward for punishment.

Transitions: Once the user gives her feedback, as shown in Figure 1, the recommendation agent updates its state from s_{t_i} to $s_{t_{i+1}}$ and moves on to the next round, where a new item will be considered for recommendation. If the feedback is positive (i.e. the user interacted with the item), then the user profile history is updated, by adding the item to her past interactions, computing a new average for the past item features and computing the updated user-item similarity features.

In addition to classical MDP components, we make use of *user sessions* which are used by the original A3C method. We break down the interactions of the user as a sequence of short-term interactions. As shown in Figure 1, a user may visit our system and interact with items sequentially over time (i.e., t_i, t_{i+1},..., t_{i+m}). Our recommender agent will then interact with the user by recommending items or not, until their interaction reaches a terminal state.

C. THE PROPOSED ARCHITECTURE

The DARES neural network model shown in Figure 2 learns the probability of associating each of the possible classes (i.e., actions) with the user-item pair. Even though the current MDP components and DNN architecture are set for a binary classification-like task, they can be easily set for standard top-N recommendations, i.e., by classifying each user-item state to its relevance class (using as a relevance score the 'value' of the policy action), or for rating prediction in its multi-class setting.

Figure 3 shows the main process of the proposed DARES algorithm. As previously explained, there are two main elements in the DARES system: Central server and local user devices. The central server keeps a global DNN model and the local user devices receive a local copy of a version of the global model (local DNN) whenever they need to start or continue the training process. The local DNN model has input nodes related to the user features, item features, the user-item interaction features and the context features, as shown in Figure 3. On the local devices the process of extracting the features from the input data, creating the state representations and executing the local model to update the policy and the states are taking place. The local DNN model computes the losses and the gradients for each state and this continues for a batch of m items or until no more items are available. Then, the loss gradients with respect to the latest batch are sent back to the server asynchronously so that the server applies them to the global model and updates the global model weights according to these gradients. The newly trained global model is then sent back to update the local model on the device and start the next batch of the training process.

The devices send the gradient losses to the server after a maximum of m time steps (interactions), or when a terminal state is reached (e.g., user leaves the system). At that point, the local recommendation agent computes (i) the value loss L_v by using Equation 3, (ii) the policy loss L_p by using Equation 4 where $H(\pi(s_t;\theta))$ is the entropy of the policy that improves the model's exploration and β is a parameter that controls the strength of the entropy regularisation term, and (iii) the gradients of the losses, for each step executed.

$$L_v = \sum (R_t - V(s_t;\theta_v))^2 \tag{3}$$
$$L_p = -\log(\pi(a_t|s_t;\theta))A(s_t,a_t;\theta,\theta_v) - \beta H(\pi(s_t;\theta)) \tag{4}$$

Gradients are computed by using the following update rules:

$$\begin{aligned} d\theta &= d\theta + \nabla_\theta L_p \\ d\theta_v &= d\theta_v + \partial(L_v)/\partial\theta_v, \end{aligned} \tag{5}$$

θ and θ_v represent the parameters of *policy* and *value*.

As demonstrated in Figure 1, update and synchronisation can happen at different time steps for the local agents and there is no restriction that all devices have to be online at the same time or learn at the same speed (as in other synchronised distributed methods). This allows the server to continuously update the model and quickly adapt to new trends and changes in such a dynamic environment as i.e. news recommendations. yeah

V. PRIVACY ANALYSIS

In centralised RSs, users are sending their interaction history, preferences and profiles to central servers in order to train the global RS models. This centralised training is considered to increase the risk to user privacy, since all private user information is transmitted to remote servers and stored and processed there, being susceptible to a plethora of attacks. On the other hand, distributed training is considered more privacy preserving by default, since the central server does not store any identifiable user information. The proposed DARES algorithm is a distributed algorithm and, as such, does not store any user information such as profiles, ratings, etc. on a central server. DARES only makes *temporary* use of the user loss gradients in order to compute the weights of the global model. These gradients are not stored but are only used for updating the global model on the server. However, as has been shown in the literature [50], the loss gradients may be used to extract some parts of the user profile in a man-in-the-middle-attack. It is not within the scope of this work to deal with the privacy issues of the proposed model and how easy/difficult will be to extract user information from the gradients. To improve the privacy, techniques such as differential privacy can be used, as discussed in [1], [31]. However, such an analysis is deferred for future research and is not in the scope of this work.

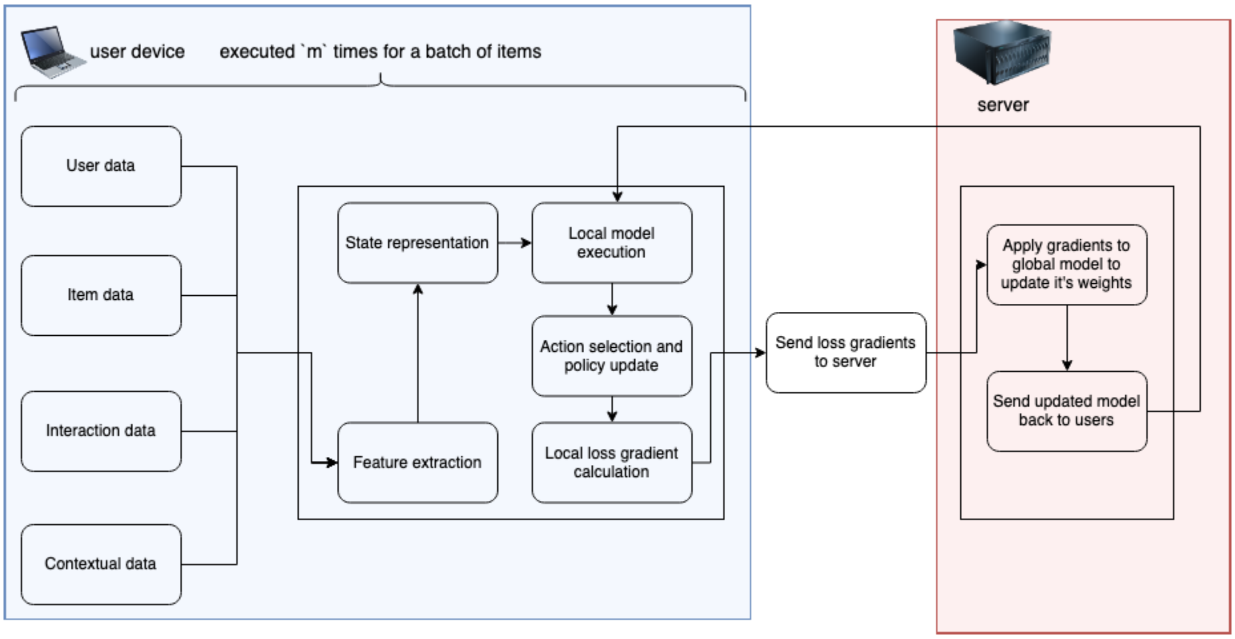
user device
executed `m` times for a batch of items
User data
Item data
Interaction data
Contextual data
State representation
Local model execution
Action selection and policy update
Feature extraction
Local loss gradient calculation
Send loss gradients to server
server
Apply gradients to global model to update it's weights
Send updated model back to users

FIGURE 3. The DARES process.

VI. EXPERIMENTAL EVALUATION

We compare our proposed DARES algorithm against both centralised and distributed baseline RL methods, and we analyse how the recommendation quality and cost is affected by various hyper parameters. We also analyse the convergence speed and stability of DARES, as well as the training time. We finally discuss how the results are affected by the specific characteristics of the datasets.

A. DATASETS

We use three public datasets in our experiments, two from the news domain and one from the movies domain, aiming to verify that our results are transferable across multiple recommendation domains. We use a subset of the SmartMedia News Adressa [16], the Outbrain dataset,[1] keeping only the publisher with id = 43 and the MovieLens100k[2] (ml100k) dataset. The first two datasets contain detailed user interaction logs of news platforms, which allow us to replay the user click sessions offline, and the ml100k dataset is one of the most commonly used benchmark datasets in the recommender system domain. We pre-process the datasets as follows:

- we extract dense subsets of the three datasets by removing users with too few interactions. For Adressa, we remove users with less than 3 interactions (because our algorithm uses initially 3 items to create the user historical profile) and for Outbrain we remove users with less than 16 interactions. Movielens-100k is a dense dataset so no users were removed.

[1]https://www.kaggle.com/c/outbrain-click-prediction

[2]https://grouplens.org/datasets/movielens/100k/

TABLE 1. Statistics of the Adressa, the Outbrain dataset and the MovieLens datasets.

	Adressa	Outbrain	ml100k
Interactions	70,226	441,213	100,000
Users	13,158	22,713	943
Items	731	2,302	1,682
Avg len(session)	5.34	19.43	84.84
Min len(session)	4	13	16
Max len(session)	20	182	590

- we perform an 80% − 20% temporal training/test split of each user's interaction data. Some statistics of the datasets are presented in Table 1, including the Average length of the user session, which is the average number of rated items per user. As shown, in Adressa and Outbrain, users interact with very few articles per session, which makes the recommendation task a significant challenge for all algorithms.

B. EXPERIMENTAL SETTINGS

Throughout the experiments, we use the task of predicting the next item to be recommended to a given user. In other words, the algorithms use a single item per user as a target item and try to find the optimal action a for that item (recommend/not recommend) in the current state s, in order to create the policy $\pi(a_t|s_t;\theta)$.

For simulating the recommendation task, we used PyrecGym [36], an RL environment based on the OpenAI gym [9] which processes the data through a generalised pipeline and simulates the agent-environment interaction process. This environment utilises the logs of historical interactions of the

TABLE 2. Features of each dataset used in the experiments.

Feature type	Adressa	Outbrain	ml100k
User features			
Category	device, OS, region, city	platform, geo-location, traffic source	gender, occu-pation, postcode
Item features			
Integer	total clicks, average active time	total clicks, publish time	total clicks
Category	author	source id	-
Text	keywords, title	-	genre
Dictionary	-	category, topic, entity	-

users and items. For training, we use 3 random items of the user interactions to create the initial user profile, which is continuously updated as new interactions are being used in the training process. The rest of the user interacted items are the recommendation candidates, and the same number of negative candidates are randomly sampled from the entire item pool. We used one negative sample per user rated item. We constructed user-item feature vectors to formulate the states. The features that were used in the experiments are shown in Table 2.

We used the same MDP components, i.e., states, actions, for the baseline and proposed methods. We train the models for all users in all three datasets and their full interaction history. The metric that we use for evaluation is the Click Through Rate (CTR) [5], which is the number of clicks that the users have interacted with an item that was recommended by the model divided by the total number of recommended items.

For DARES, since it is asynchronous, we follow a similar evaluation process as in [27] and we average over the best 5 models from a range of experiments using different learning rates. On the Outbrain dataset, the learning rate was 5×10^{-5}, on Adressa the learning rate was 10^{-4}, while on ml100k, the learning rate was 5×10^{-5}. For the DNN we used a dense neural network with 100 units and a $\ell 1$ regularisation of 0.001. We also used a dropout layer of 30% and early stopping when necessary to avoid overfitting and diverging of the neural networks.

C. COMPARISON WITH BASELINE REINFORCEMENT LEARNING METHODS

We compare our DARES with other popular RL techniques, both centralised and distributed:

1) LinUCB, a MAB-based approach as presented in [21].
2) Deep Q-Network (DQN), similar to the work in [59].
3) Double Deep Q-Network (DDQN), similar to the work in [59].
4) a distributed version of the Advantage Actor-Critic model (A2C-D),[3] A2C in its original distributed synchronous setting, where the data are distributed randomly at each worker.
5) Advantage Actor-Critic (A2C) in federated-liked setting (A2C-F), which is similar to our proposed DARES with each worker having all the data of each user, but is updated in a synchronous way.

All baseline models use their best performing hyper parameter values.

Figure 4 shows the global average CTR that the methods attain during the *training* process and for the three different datasets. As shown in the figure, DARES achieves comparable results with the state-of-the-art algorithms, which are either centralised (LinUCB, DQN, DDQN) or synchronous (A2C-F, A2C-D). More specifically, we can see that in the Adressa dataset, DARES is very close to LinUCB and A2C-F, while it outperforms DQN, DDQN and A2C-D. On the Outbrain dataset, we can see that DARES outperforms all other algorithms. DARES was run with a low learning rate ($5 * 10^{-5}$) to avoid divergence and it starts slow, while LinUCB and DDQN almost immediately converge to their final value. However, DARES manages to have better training CTR in the end. Similarly, on the ml100k dataset, DARES outperforms all other algorithm and continuously improves while other algorithms converge around 15k episodes.

Next, we evaluated the fully trained models on a realistic testing environment, employing the One-Plus-k-Random-Items evaluation protocol [7]. For each user, we use the training dataset to create the initial user history profile, and we consider for testing all the q rated items of her test set. Then, we randomly select $q \times k$ items from the entire candidate item (except from the ones in the historical profile) set as irrelevant examples, and add them to the test set. We use the fully trained models to recommend items in the test set to users, and calculate the overall test CTR.

The performance of all methods on the three data sets, with different values of k are presented in Table 3. The results reveal that DARES outperforms the rest of the models on all three datasets for all k. The LinUCB and A2C-F have a close training CTR as DARES on Adressa dataset when $k = 1$, but they fall short on the rest of the scenarios. It is very interesting to notice that even when the negatively sampled items are significantly more than the user interactions (k more than 50) the performance of DARES is far superior than the rest of the models, with A2C-F being the closest competitor.

Among the five baseline methods, we did not find a significant performance difference between centralised (LinUCB, DQN, DDQN) and distributed (A2C-F, A2C-D) models. For instance, A2C-F achieves similar CTR comparing to LinUCB

[3]`https://github.com/openai/baselines`

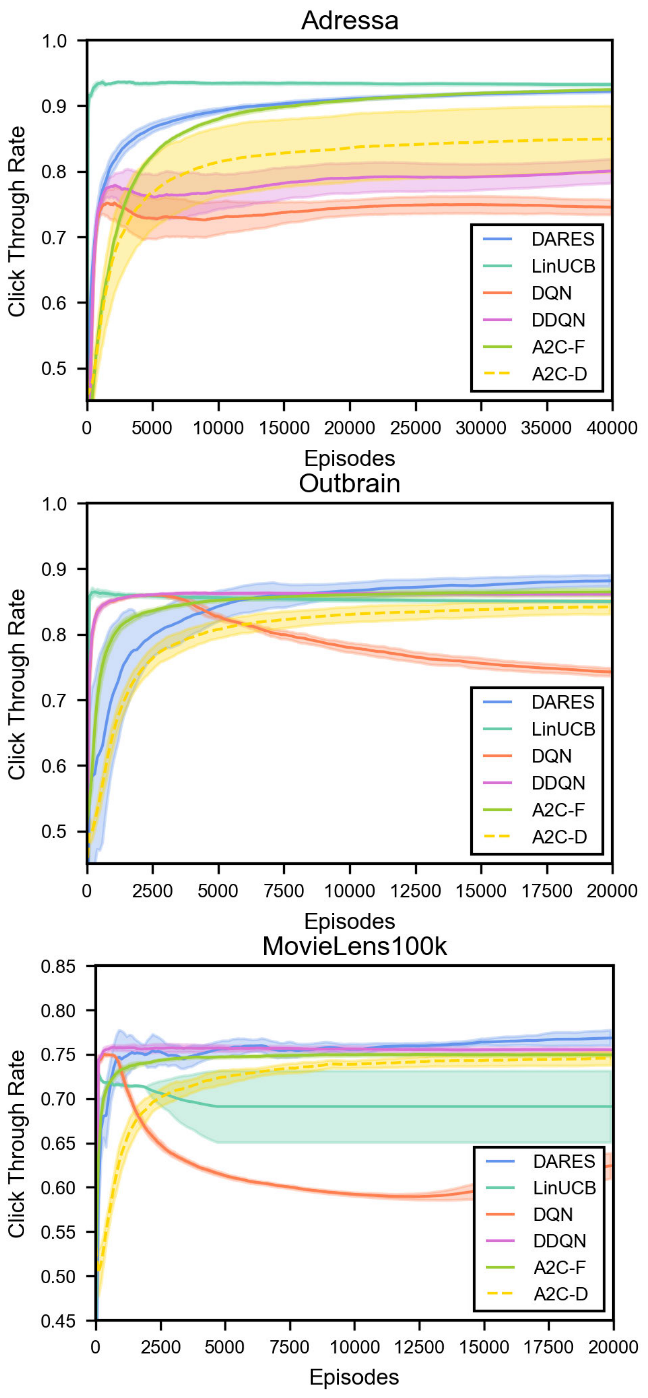

FIGURE 4. Global average CTR vs number of training steps for (a) Adressa and (b) Outbrain.

on Adressa, and similar to DDQN on Outbrain and ml100k. However, between the two A2C models, federated-like version is constantly better than the original distributed one. The main difference is that a A2C-F worker only interacts with one user at one time, but multiple users for a A2C-D worker. In an interactive recommendation scenario, we believe the federated-like setting is important, and our DARES model also support this finding.

TABLE 3. The Click Through Rate (CTR) of the 6 methods in One Plus K Random Items test setting on the datasets.

	k				
Adressa	1	10	20	50	100
LinUCB	0.930	0.608	0.417	0.198	0.102
DQN	0.839	0.517	0.359	0.173	0.090
DDQN	0.867	0.603	0.448	0.234	0.123
A2C-F	0.929	0.679	0.510	0.266	0.140
A2C-D	0.920	0.636	0.497	0.251	0.134
DARES	**0.935**	**0.722**	**0.560**	**0.311**	**0.169**
Outbrain					
LinUCB	0.856	0.359	0.213	0.096	0.049
DQN	0.883	0.417	0.261	0.120	0.063
DDQN	0.881	0.403	0.246	0.113	0.059
A2C-F	0.888	0.428	0.261	0.122	0.064
A2C-D	0.869	0.381	0.240	0.109	0.057
DARES	**0.917**	**0.569**	**0.394**	**0.202**	**0.111**
ml100k					
LinUCB	0.780	0.268	0.152	0.068	0.035
DQN	0.747	0.227	0.128	0.055	0.028
DDQN	0.747	0.228	0.127	0.055	0.028
A2C-F	0.741	0.220	0.121	0.052	0.026
A2C-D	0.719	0.199	0.110	0.047	0.024
DARES	**0.882**	**0.454**	**0.301**	**0.146**	**0.079**

In the next subsections, we analyse the performance of the proposed DARES algorithm against various hyper-parameters.

D. COMMUNICATION COST AND ACCURACY

In this section, we analyse how the recommendation accuracy and the communication cost (number of messages sent from the devices to the central server) change as we vary the parameter m of the DARES model (which is shown in Figure 3). This parameter is the batch size of the number of interactions that are used to train locally the model before sending a model update to the central server and is a measure of the frequency that the local devices send the gradients to the central server. This experiment shows how (i) the model test CTR is affected and (ii) the number of messages sent to the server changes, as we increase m. The experiments were run for seven m values in the range $(1 - 100)$: 1, 2, 5, 10, 20, 50, 100.

Figure 5 shows the CTR and number of messages sent to the server per user per each iteration as we increase the m time steps parameter. In Figure 5a, we can see that for Adressa and Outbrain datasets, the batch size doesn't significantly affect the testing CTR value. When m is low, the CTR for both datasets starts slightly lower and increases when m is around 10, while after it that drops slightly. The results are different in a dense dataset like ml100k, in which the CTR is low when m is 1 or 2, and the maximum CTR is achieved when m is above 15. This can be justified because of the

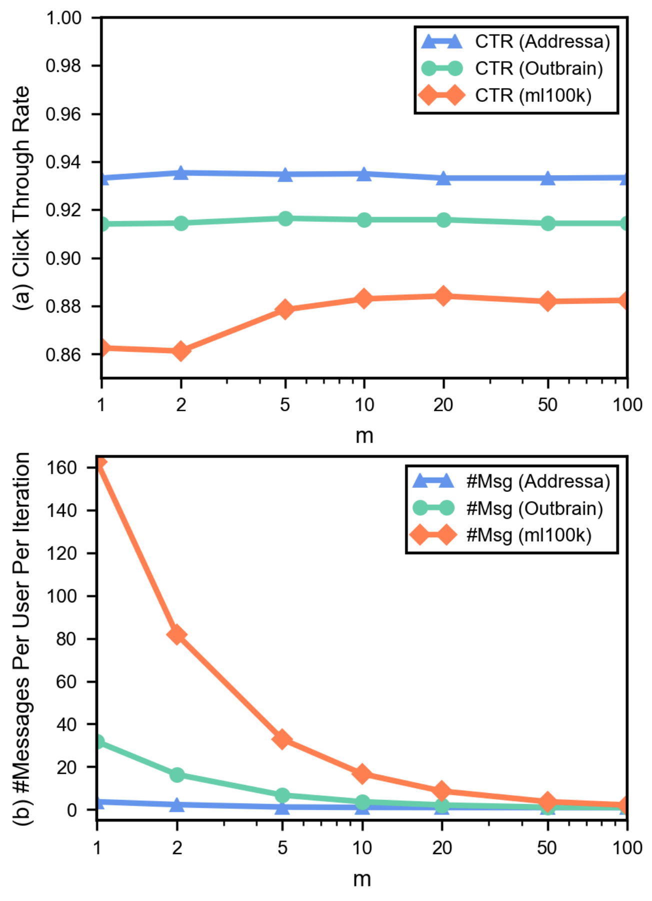

FIGURE 5. **(a) CTR and (b) number of messages exchanged between user devices and central server vs. different values of time steps parameter *m*.**

asynchronous nature of the algorithm and the fact that when m is very low (i.e. 1) the users are continuously updating the global model with a single gradient loss. This means that the global model is not being trained properly, because it doesn't get an accurate representation of the users' losses and the global model is trained continuously by a very small loss sample. Adressa has a very small average length session, so the changes in CTR are minor when m is above 10, since only a very small number of users have a training session longer that. In Outbrain we can see a small peak when m is between 5 and 15 and then it drops slightly. This shows that in this dataset the algorithm overfits slightly when the batch size is higher than the average length of the user session. In contrast, in a very dense dataset like ml100k, the algorithm needs a high batch size to get a good estimation of the user profile and properly update the model. For this reason, in the low m values, the CTR is low, since model updates with a very few interactions won't be correctly representing the user's preferences.

In Figure 5b, we can see that, as expected, the number of messages is reduced as m increases, because the local devices use a batch of interactions to train the local model and only send the gradient losses to the server after m local training steps. When $m = 1$ it means that the devices send a gradient loss after each user interaction is passed in the local model execution and only 1 gradient loss is sent. So, for example, for Adressa and $m = 1$, this means that the communication cost will be approximately 4.68 messages per user- we calculate the cost like this: the average length of a user session is 5.34 interactions (as seen in Table 1), but 3 items are used for creating the user profile history, which means that the training interactions are 2.34 per user; for training we use one negative item for each positive, thus the actual total number of training interactions are 4.68 per user. When m is larger, the communication with the server is less frequent. The change in the communication cost is more evident in a dense dataset like ml100k, where the average session length is much larger.

E. CONVERGENCE AND STABILITY FOR DIFFERENT BATCH SIZES

In this experiment, we analyse the effect of the batch size m on the convergence speed and the stability of the DARES algorithm. Figure 6 shows the testing CTR over time for different values of m. This experiment shows the test CTR at different steps through the training process, with each "testing step" equal to 2000 training steps (users). As the figure shows, on Adressa, the batch size doesn't have a significant difference in the convergence speed or stability. This is expected due to the short average session length of the users. On Outbrain, it is noticed that the convergence is much faster with low values of the batch size and the model diverges when large values are used. On the contrary, the behaviour of DARES is opposite in ml100k. While the model still reaches faster the maximum CTR with low values of m, it also diverges easily. However, with large values of m, although the model converges slower to the maximum value, it is stable and doesn't diverge. This result for ml100k justifies the results of Figure 5, which showed that for ml100k the maximum CTR is achieved with high values of m. A justification for this behaviour is that because of the long average session length of the users in ml100k and the density of the dataset, a large m is required in order to properly update the model with a representative model update.

F. CTR WHEN ARYING THE NUMBER OF WORKERS

In this experiment, we aim to analyse if and how the number of workers used for training the model affects the results. We have to note here that DARES is an extension of A3C which distributes the training process over a number of workers. In the ideal case, for simulating a real-world scenario with one user per worker, it would require i.e. in the Outbrain scenario to launch 22713 processes, something that is not feasible on a standalone server. The experiments in the previous sections were run with 5 workers, meaning that 5 users are asynchronously updating the model at any given time. This can be considered as similar to a federated learning scenario in which we average the model over 5 selected users. The implementation of the asynchronous updates was done using

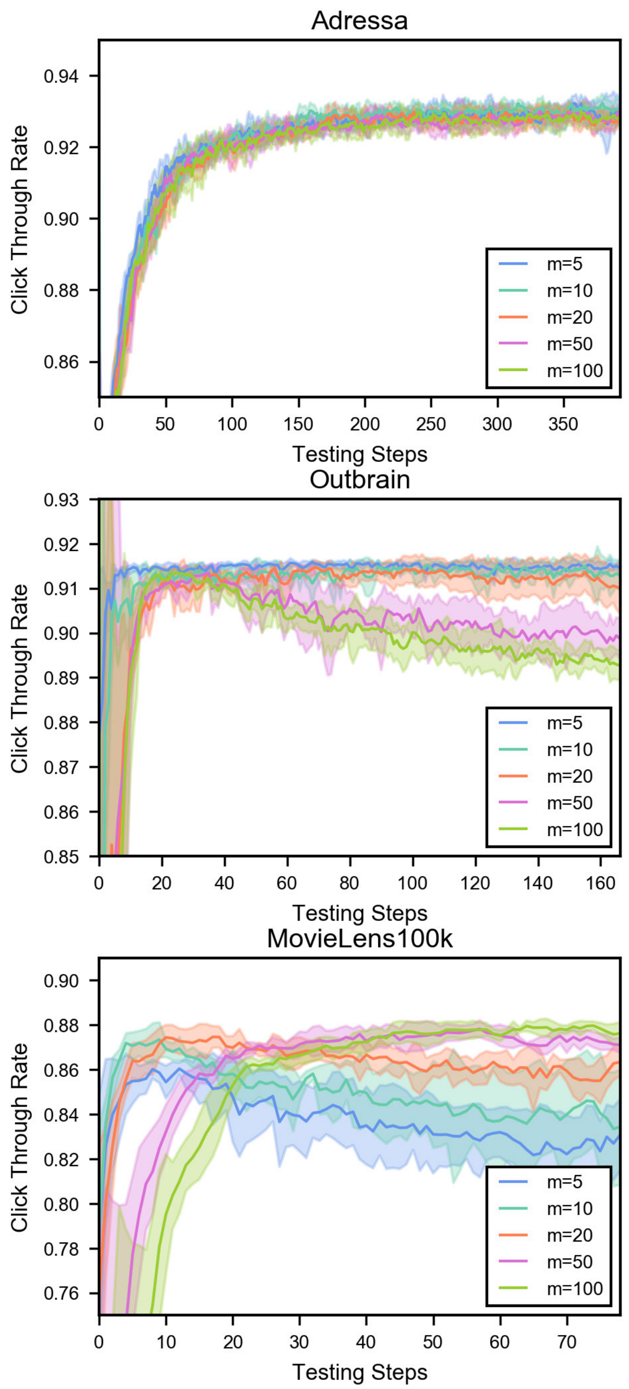

FIGURE 6. The testing CTR of DARES vs different values of time steps parameter *m* for Adressa, Outbrain and MovieLens100k datasets.

python multiprocessing and a redis[4] queue for the exchange of messages between the workers/users and the server. Each worker keeps a local version of the model, which changes when the worker trains a new user, by fetching the latest model from the server.

As shown in Figure 7, the number of workers doesn't significantly affect the results. Especially for Outbrain, the results are almost identical even with a large number

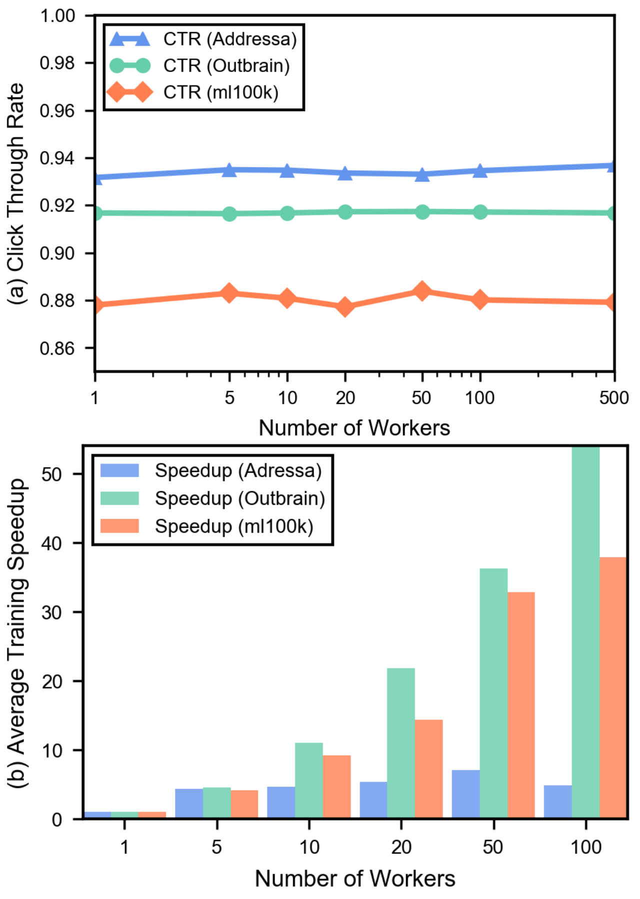

FIGURE 7. (a) The CTR and (b) the average training speedup vs. number of workers of DARES over Adressa, Outbrain and MovieLens100k datasets.

of workers. For ml100k there is a small variance, but it is also negligible. In Adressa, the CTR is also not affected significantly with the number of workers, although it seems that a small number of workers (i.e. <10) causes a small drop in the CTR.

G. TRAINING TIME WHEN VARYING THE NUMBER OF WORKERS

In this experiment, we analyse the training time of the model by varying the number of workers. Figure 7 shows the speedup in training time when the number of workers increases compared to when the training is done only with one worker. The speedup is measured using the following equation:

$$speedup_k = \frac{training\_time\_single\_worker}{training\_time\_k\_workers} \tag{6}$$

As shown in Figure 7, for Outbrain and ml100k the speedup is significant when the number of workers increases. The speedup is much higher for Outbrain that has a high number of users, but it is also quite high for ml100k. On the contrary, the speedup is not very noticeable in Adressa and is increases very slightly up to 50 workers, while after that it decreases. This can be justified by the fact that because of the

[4]https://redis.io

very small session length per user in Adressa, the overhead of creating and managing a large number of users creates a bottleneck when the number of workers is very high. The session length is much longer in Outbrain and ml100k, which justifies why this bottleneck is not evident in these datasets.

VII. CONCLUSION

We propose DARES, a flexible, distributed and asynchronous RL framework for RS, that is part of an effort to adapt recommender systems to the new challenges of providing accurate and timely recommendations in a distributed and decentralised environment. Inspired by A3C and federated learning (FL) frameworks, DARES allows users to keep their data on the devices and run their local RS model on their devices, sharing only loss gradients with a central agent for the purpose of training a global recommendation model. Sharing gradients in this way offers significant benefits – sharing loss gradients is considerably less revealing than sharing personal user data, and the user benefits from a global model which is trained on data from many users. Users can have full control over their data which is only stored on their device and never shared. Users receive model updates asynchronously whenever they have completed a set of m steps (interactions) or when they leave the system.

Our evaluation shows that we can achieve these important benefits without sacrificing recommendation performance, since DARES achieves state-of-the-art levels of CTR on three well-known real-world datasets. We also provide comparisons to a number of benchmark state of the art approaches, showing that in most cases we can outperform them. The communication costs associated with the sharing of gradients and synchronising the global models across user devices can remain low, while still achieving very high CTR values. We also demonstrate the performance speedup that we can achieve by using multiple workers and show that the number of workers doesn't have a significant affect on the model performance.

Future work in this area will cover the theoretical analysis of the convergence of the model under various constraints and the optimal use of early stopping to get the maximum model performance. Future work will also focus on the effect of using additional local training to update the local model and analyse the effect on the global model. This will also be a step towards personalising the local models on the user devices, so that they provide more user-tailored recommendations. Additionally, the privacy of the model will be investigated in the future, analysing the potential to improve the privacy using i.e. differential privacy. Finally, the current model requires the users to share their loss gradients, so in the future we will modify the model to share the model weights, moving closer to the original federated learning scenario.

REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*, 2016, pp. 308–318.

[2] M. A.-U. Din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, "Federated collaborative filtering for privacy-preserving personalized recommendation system," 2019, *arXiv:1901.09888*. [Online]. Available: https://arxiv.org/abs/1901.09888

[3] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," 2019, *arXiv:1912.00818*. [Online]. Available: https://arxiv.org/abs/1912.00818

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," 2017, *arXiv:1708.05866*. [Online]. Available: https://arxiv.org/abs/1708.05866

[5] P. A. Clemenshia and M. Vijaya, "Click through rate prediction for display advertisement," *Int. J. Comput. Appl.*, vol. 975, p. 8887, Feb. 2016.

[6] S. Badsha, X. Yi, I. Khalil, and E. Bertino, "Privacy preserving user-based recommender system," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1074–1083.

[7] A. Bellogin, P. Castells, and I. Cantador, "Precision-oriented evaluation of recommender systems: An algorithmic comparison," in *Proc. 5th ACM Conf. Recommender Syst. (RecSys)*, 2011, pp. 333–336.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: https://arxiv.org/abs/1902.01046

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*. [Online]. Available: https://arxiv.org/abs/1606.01540

[10] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," 2018, *arXiv:1802.07876*. [Online]. Available: https://arxiv.org/abs/1802.07876

[11] H. Chen, X. Dai, H. Cai, W. Zhang, X. Wang, R. Tang, Y. Zhang, and Y. Yu, "Large-scale interactive recommendation with tree-structured policy gradient," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3312–3320.

[12] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, "Top-K off-policy correction for a REINFORCE recommender system," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 456–464.

[13] S.-Y. Chen, Y. Yu, Q. Da, J. Tan, H.-K. Huang, and H.-H. Tang, "Stabilizing reinforcement learning in dynamic environment with application to online recommendation," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1187–1196.

[14] S. Choi, H. Ha, U. Hwang, C. Kim, J.-W. Ha, and S. Yoon, "Reinforcement learning based recommender system using biclustering technique," 2018, *arXiv:1801.05532*. [Online]. Available: https://arxiv.org/abs/1801.05532

[15] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," 2015, *arXiv:1512.07679*. [Online]. Available: https://arxiv.org/abs/1512.07679

[16] J. A. Gulla, L. Zhang, P. Liu, O. Özgöbek, and X. Su, "The Adressa dataset for news recommendation," in *Proc. Int. Conf. Web Intell.*, New York, NY, USA, 2017, pp. 1042–1048, doi: 10.1145/3106426.3109436.

[17] A. Jalalirad, M. Scavuzzo, C. Capota, and M. Sprague, "A simple and efficient federated recommender system," in *Proc. 6th IEEE/ACM Int. Conf. Big Data Comput., Appl. Technol. (BDCAT)*, Dec. 2019, pp. 53–58.

[18] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*. [Online]. Available: https://arxiv.org/abs/1912.04977

[19] V. R. Konda and J. N. Tsitsiklis, "OnActor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, Jan. 2003.

[20] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016, *arXiv:1610.05492*. [Online]. Available: https://arxiv.org/abs/1610.05492

[21] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World Wide Web (WWW)*, 2010, pp. 661–670.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: https://arxiv.org/abs/1509.02971

[23] F. Liu, H. Guo, X. Li, R. Tang, Y. Ye, and X. He, "End-to-end deep reinforcement learning based recommendation with supervised embedding," in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 384–392.

[24] F. Liu, R. Tang, X. Li, W. Zhang, Y. Ye, H. Chen, H. Guo, and Y. Zhang, ''Deep reinforcement learning based recommendation with explicit user-item interactions modeling,'' 2018, *arXiv:1810.12027*. [Online]. Available: https://arxiv.org/abs/1810.12027

[25] Z. Lu and Q. Yang, ''Partially observable Markov decision process for recommender systems,'' 2016, *arXiv:1608.07793*. [Online]. Available: https://arxiv.org/abs/1608.07793

[26] B. McMahan and D. Ramage. (2017). *Federated Learning: Collaborative Machine Learning Without Centralized Training Data*. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, ''Asynchronous methods for deep reinforcement learning,'' in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and S. Petersen, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, p. 529, 2015.

[29] R. Pagare and S. A. Patil, ''Study of collaborative filtering recommendation algorithm scalability issue,'' *Int. J. Comput. Appl.*, vol. 67, no. 25, pp. 10–15, Apr. 2013.

[30] B. Peng, X. Li, J. Gao, J. Liu, and K.-F. Wong, ''Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning,'' in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, vol. 1, 2018, pp. 2182–2192.

[31] N. Phan, X. Wu, and D. Dou, ''Preserving differential privacy in convolutional deep belief networks,'' *Mach. Learn.*, vol. 106, nos. 9–10, pp. 1681–1704, Oct. 2017.

[32] P. Rojanavasu, P. Srinil, and O. Pinngern, ''New recommendation system using reinforcement learning,'' *Special Issue Int. J. Comput., Internet Manage.*, vol. 13, no. SP3, pp. 1–5, 2005.

[33] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, ''High-dimensional continuous control using generalized advantage estimation,'' 2015, *arXiv:1506.02438*. [Online]. Available: https://arxiv.org/abs/1506.02438

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, ''Proximal policy optimization algorithms,'' 2017, *arXiv:1707.06347*. [Online]. Available: https://arxiv.org/abs/1707.06347

[35] G. Shani, D. Heckerman, and R. I. Brafman, ''An MDP-based recommender system,'' *J. Mach. Learn. Res.*, vol. 6, pp. 1265–1295, Sep. 2005.

[36] B. Shi, M. G. Ozsoy, N. Hurley, B. Smyth, E. Z. Tragos, J. Geraci, and A. Lawlor, ''PyRecGym: A reinforcement learning gym for recommender systems,'' in *Proc. 13th ACM Conf. Recommender Syst.*, Sep. 2019, pp. 491–495.

[37] B. Shi, E. Z. Tragos, R. Nong, B. Smyth, N. J. Hurley, and A. Lawlor, ''A distributed asynchronous deep reinforcement learning framework for recommender systems,'' in *Proc. 14th ACM Conf. Recommender Syst. (RecSys)*, 2020, pp. 1–3.

[38] R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J.-P. Hubaux, ''Preserving privacy in collaborative filtering through distributed aggregation of offline profiles,'' in *Proc. 3rd ACM Conf. Recommender Syst. (RecSys)*, 2009, pp. 157–164.

[39] K. Shyong, D. Frankowski, and J. Riedl, ''Do you trust your recommendations? An exploration of security and privacy issues in recommender systems,'' in *Proc. Int. Conf. Emerg. Trends Inf. Commun. Secur.* Berlin, Germany: Springer, 2006, pp. 14–29.

[40] R. S. Sutton, ''Dyna, an integrated architecture for learning, planning, and reacting,'' *ACM SIGART Bull.*, vol. 2, no. 4, pp. 160–163, Jul. 1991.

[41] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, ''Policy gradient methods for reinforcement learning with function approximation,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1057–1063.

[42] N. Taghipour, A. Kardan, and S. S. Ghidary, ''Usage-based Web recommendations: A reinforcement learning approach,'' in *Proc. ACM Conf. Recommender Syst. (RecSys)*, 2007, pp. 113–120.

[43] L. Tang, Y. Jiang, L. Li, and T. Li, ''Ensemble contextual bandits for personalized recommendation,'' in *Proc. 8th ACM Conf. Recommender Syst. (RecSys)*, New York, NY, USA, 2014, pp. 73–80, doi: 10.1145/2645710.2645732.

[44] L. Tang, Y. Jiang, L. Li, C. Zeng, and T. Li, ''Personalized recommendation via parameter-free contextual bandits,'' in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, New York, NY, USA, Aug. 2015, pp. 323–332, doi: 10.1145/2766462.2767707.

[45] H. Wang, Q. Wu, and H. Wang, ''Learning hidden features for contextual bandits,'' in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, New York, NY, USA, Oct. 2016, pp. 1633–1642, doi: 10.1145/2983323.2983847.

[46] H. Wang, Q. Wu, and H. Wang, ''Factorization bandits for interactive recommendation,'' in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2695–2702.

[47] X. Wang, Y. Chen, J. Yang, L. Wu, Z. Wu, and X. Xie, ''A reinforcement learning framework for explainable recommendation,'' in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 587–596.

[48] C. J. C. H. Watkins, ''Learning from delayed rewards,'' Ph.D. dissertation, King's College, Cambridge, U.K., 1989.

[49] R. J. Williams, ''Simple statistical gradient-following algorithms for connectionist reinforcement learning,'' *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[50] F. Yan, S. Sundaram, S. V. N. Vishwanathan, and Y. Qi, ''Distributed autonomous online learning: Regrets and intrinsic privacy-preserving properties,'' *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2483–2493, Nov. 2013.

[51] C. Zeng, Q. Wang, S. Mokhtari, and T. Li, ''Online context-aware recommendation with time varying multi-armed bandit,'' in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2016, pp. 2025–2034, doi: 10.1145/2939672.2939878.

[52] X. Zhao, C. Gu, H. Zhang, X. Yang, X. Liu, J. Tang, and H. Liu, ''DEAR: Deep reinforcement learning for online advertising impression in recommender systems,'' 2019, *arXiv:1909.03602*. [Online]. Available: https://arxiv.org/abs/1909.03602

[53] X. Zhao, L. Xia, J. Tang, and D. Yin, '''Deep reinforcement learning for search, recommendation, and online advertising: A survey' by Xiangyu Zhao, long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator,'' *ACM SIGWEB Newslett.*, vol. 2019, pp. 1–15, Jul. 2019.

[54] X. Zhao, L. Xia, L. Zhang, Z. Ding, D. Yin, and J. Tang, ''Deep reinforcement learning for page-wise recommendations,'' in *Proc. 12th ACM Conf. Recommender Syst.*, New York, NY, USA, Sep. 2018, pp. 95–103, doi: 10.1145/3240323.3240374.

[55] X. Zhao, L. Xia, L. Zou, H. Liu, D. Yin, and J. Tang, ''Whole-chain recommendations,'' 2019, *arXiv:1902.03987*. [Online]. Available: https://arxiv.org/abs/1902.03987

[56] X. Zhao, T. Xu, Q. Liu, and H. Guo, ''Exploring the choice under conflict for social event participation,'' in *Proc. Int. Conf. Database Syst. Adv. Appl.* San Francisco, CA, USA: Springer, 2016, pp. 396–411.

[57] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin, ''Recommendations with negative feedback via pairwise deep reinforcement learning,'' in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1040–1048.

[58] X. Zhao, L. Zhang, L. Xia, Z. Ding, D. Yin, and J. Tang, ''Deep reinforcement learning for list-wise recommendations,'' 2017, *arXiv:1801.00209*. [Online]. Available: https://arxiv.org/abs/1801.00209

[59] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, ''DRN: A deep reinforcement learning framework for news recommendation,'' in *Proc. World Wide Web Conf. (WWW)*, 2018, pp. 167–176.

[60] L. Zou, L. Xia, Z. Ding, J. Song, W. Liu, and D. Yin, ''Reinforcement learning to optimize long-term user engagement in recommender systems,'' in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2810–2818.

[61] L. Zou, L. Xia, Z. Ding, D. Yin, J. Song, and W. Liu, ''Reinforcement learning to diversify top-N recommendation,'' in *Proc. Int. Conf. Database Syst. Adv. Appl.* Springer, 2019, pp. 104–120.

[62] L. Zou, L. Xia, P. Du, Z. Zhang, T. Bai, W. Liu, J.-Y. Nie, and D. Yin, ''Pseudo Dyna-Q: A reinforcement learning framework for interactive recommendation,'' in *Proc. 13th Int. Conf. Web Search Data Mining*, Jan. 2020, pp. 816–824.

BICHEN SHI received the Ph.D. degree in computer science from University College Dublin, Ireland. She is currently a Postdoctoral Researcher with the Insight Centre for Data Analytics, School of Computer Science, University College Dublin. Her research interests include deep reinforcement learning, recommender systems, and machine learning.

ELIAS Z. TRAGOS received the master's degree in business administration (MBA) in techno-economics and the Ph.D. degree in wireless communications. He is currently a Research Project Manager with the Insight Centre for Data Analytics, UCD, Ireland. He has been actively involved in many EU and National research projects as a researcher, a technical manager, and the project coordinator. He has been the Technical Coordinator of the SmartCities Project RERUM. He has been extensively involved in the IERC, leading the Activity Chain 05 for Trusted IoT and co-leading the AC 03 for IoT Results Exploitation, contributing also every year in drafting the IoT strategic roadmap in the IERC *Cluster Book*. His research interests include in the areas of machine learning, distributed artificial intelligence, wireless and mobile communications, the Internet of Things, cognitive radios, network architectures, fog computing, security, and privacy.

MAKBULE GULCIN OZSOY received the B.Sc., M.Sc., and Ph.D. degrees from the Department of Computer Engineering, Middle East Technical University (METU), Ankara, Turkey, in 2008, 2011, and 2016, respectively. She was a Visiting Student with the University of Calgary, Calgary, AB, Canada, from October 2014 to August 2015. She is currently a Postdoctoral Researcher with the Insight Centre for Data Analytics, University College Dublin (UCD), Dublin, Ireland. Her research interests include recommender systems, machine learning, and information retrieval.

RUIHAI DONG is currently an Assistant Professor with the School of Computer Science, University College Dublin. His research interests include broadly in machine learning and deep learning, and their applications in recommender systems and finance. He has published in top peer-reviewed journals and conferences, such as WWW, RECSYS, IUI, and IJCAI. In 2018, he was awarded the Outstanding Research Award 2018 by the UCD School of Computer Science for a series of significant publications that year. He has a track record of collaboration with industry and has worked with companies, including Eagle Alpha, SkillPages, and Samsung, and individually winning funding from Enterprise Ireland for commercialization studying of his research.

NEIL HURLEY received the M.Sc. degree in mathematical science from UCD, in 1988, and the Ph.D. degree with a focus on knowledge-based environments for engineering design, in 1995. In 1989, he joined the Hitachi Dublin Laboratory, a computer science research laboratory-based at Trinity College Dublin. A patent for the system was granted in the USA and Europe. He became a Group Leader of the Parallel Computing Group, HDL, in 1995. During this period he worked on the development of parallel simulation and optimization software, such as parallel finite element analysis and dynamic load balancing. In 1998, he was appointed as a Manager of HDL. He became a Lecturer with the Department of Computer Science, UCD, in 1999. In 2001, he established the Information Hiding Laboratory, a research laboratory focused on technology for embedding information in digital content. His research focus switched to parallel and distributed computing. He has won over €1 million euro in research funding from Enterprise Ireland, Science Foundation Ireland, the EU, and industrial partners.

BARRY SMYTH received the Honorary Doctor of Technology (D.Tech.) degree (Hons.) from Robert Gordon University, U.K., in 2014. He holds the Digital Chair of computer science with University College Dublin and the Director of the Insight Centre for Data Analytics. He was the Director of the Clarity Centre for Sensor Web Technologies, from 2008 to 2013, and previously held the position of the Head of the School of Computer Science and Informatics, UCD. In 1999, he co-founded ChangingWorlds based on his personalization research and helped grow the company to more than 150 employees before it was acquired by Amdocs Inc., in 2008. In 2008, he co-founded HeyStaks based on his social search research. He has helped HeyStaks to raise more than 3.5m in venture capital funding to date and continues to advise the company on its technology and market strategy. He is actively involved in the Irish startup scene as an advisor and an investor, and he serves on the boards of a number of local startups. His research interests fall within the field artificial intelligence and include case-based reasoning, machine learning, recommender systems, user modeling, and personalization. Since 1992, he has been published over 400 peer-reviewed articles. His research has attracted more than 13 000 citations. He has an H-index of 58 and received more than 20 best paper awards for his work. In 2014, he was named the SFI Researcher of the Year. His research interests extend beyond the laboratory and over years he has established a track-record for successfully translating his research into commercial opportunities and received numerous awards for his commercialization endeavour. He has been a fellow of the European Coordinating Committee on Artificial Intelligence (ECCAI), since 2003. He has been a member of the Royal Irish Academy, since 2011. He is also a member of the Irish Times Trust. In 2006, he was a Finalist in the Ernst and Young Entrepreneur of the Year and he received the Irish Software Association's (ISA) Inaugural Award for Outstanding Academic Achievement, in 2012.

AONGHUS LAWLOR received the Master of Science and Ph.D. degrees from University College Dublin (UCD). He is currently an Assistant Professor with UCD and a Principal Investigator with the Insight Centre for Data Analytics. He has long experience in the areas of machine learning, artificial intelligence, explainable AI, natural language processing, and recommender systems. He has been actively involved in several EU and National research projects in the above mentioned areas and has been the co-PI of very large industrial projects. He has also received several funded grants from Enterprise Ireland and the Science Foundation of Ireland, leading to patent applications.