TEE-based decentralized recommender systems: The raw data sharing redemption

Akash Dhasade, Nevena Dresevic, Anne-Marie Kermarrec, Rafael Pires
EPFL - Swiss Federal Institute of Technology
Lausanne, Switzerland
first.last@epfl.ch

***Abstract*—Recommenders are central in many applications today. The most effective recommendation schemes, such as those based on collaborative filtering (CF), exploit similarities between user profiles to make recommendations, but potentially expose private data. Federated learning and decentralized learning systems address this by letting the data stay on user's machines to preserve privacy: each user performs the training on local data and only the model parameters are shared. However, sharing the model parameters across the network may still yield privacy breaches. In this paper, we present REX, the first enclave-based decentralized CF recommender. REX exploits Trusted execution environments (TEE), such as Intel software guard extensions (SGX), that provide shielded environments within the processor to improve convergence while preserving privacy. Firstly, REX enables raw data sharing, which ultimately speeds up convergence and reduces the network load. Secondly, REX fully preserves privacy. We analyze the impact of raw data sharing in both deep neural network (DNN) and matrix factorization (MF) recommenders and showcase the benefits of trusted environments in a full-fledged implementation of REX. Our experimental results demonstrate that through raw data sharing, REX significantly decreases the training time by $18.3\times$ and the network load by 2 orders of magnitude over standard decentralized approaches that share only parameters, while fully protecting privacy by leveraging trustworthy hardware enclaves with very little overhead.**

***Index Terms*—privacy, security, recommender systems, SGX**

I. Introduction

Recommendation systems are now central in a wide variety of web applications to help users navigate through the exponentially growing volume of data. They help users to pick the items they are likely to buy on online stores, predict which movies they are willing to watch on streaming platforms [1] and decide which information to display on social media [2], to cite a few. While many approaches exist, collaborative filtering (CF) [3], [4] is arguably the most successful approach and has been widely adopted in industry. CF exploits the similarities between users to learn their preferences and accurately compute recommendations or predictions.

Precisely because recommenders learn users' preferences, they represent a serious privacy threat. User profiles are stored on service providers that may involuntarily leak them through data breaches or voluntarily release their databases for commercial purposes. CF has to face a dilemma, typically sacrificing accuracy or efficiency to guarantee privacy. For instance, relying on homomorphic encryption to encrypt user data provides a high level of privacy but is known to be notoriously impractical [5]. Differential privacy has been applied to recommendation systems at the price of significantly hampered accuracy [6].

More recently, federated learning (FL) [7] and decentralized learning systems (DLS) [8] took an orthogonal strategy: they have been introduced as an attractive alternative to address both scalability and privacy of machine learning (ML) systems. In a nutshell, FL and DLS consider a scenario where the data is fully distributed, *i.e.*, raw data is produced by users, who hold a single personal profile record and the model is trained locally. Such approaches require that users' data stay where it is produced, thus limiting their exposure. Yet, models need to be aggregated in order to provide relevant recommendations for unseen items. Learning tasks performed on user devices are merged on a central server in FL or through a gossip-based protocol in DLS. Instead of moving raw data, these approaches only allow nodes to share processed data (*e.g.*, weights and gradients), which unfortunately does not fully protect individuals' privacy. It has been shown that sharing model parameters across the network reveals some information about user profiles and yields privacy breaches [9]–[11]. Finally, sharing model parameters might induce considerable network traffic due to the large size of models. For instance, the deep neural network (DNN) model that we use in our experiments has more than $200\,000$ parameters that would be exchanged at each iteration of a decentralized learning training task, whereas by exchanging only a few data items per epoch we are able to achieve an equivalent test error target, as we will show later.

If only data could be shared safely in a network of machines, recommenders could achieve at once privacy, accuracy and scalability. In this paper, we propose REX, the first enclave-based decentralized recommender that achieves this three-dimensional goal. REX exploits trusted execution environments (TEE), such as Intel software guard extensions (SGX), that provide shielded environments within the processor. Since users cannot inspect what is being processed inside the enclaves on their own machines, REX enables raw data sharing among nodes. Given SGX assurances, REX enables to conceal sensitive data from adversaries both during the decentralized training and the communication phases. The benefits of enabling raw data sharing in decentralized systems with REX is threefold: (i) disseminating raw data in the

system speeds up the training time, (ii) sharing data instead of model parameters in recommender systems, where datasets are sparse, significantly reduces network traffic in the system, and (iii) the use of TEEs fully preserves users' privacy.

In this paper, we make the following contributions:

- We propose the design and evaluation of REX [1], a novel decentralized recommender that avoids trading-off accuracy or efficiency for privacy. REX relies on SGX to quickly compute a recommendation model while limiting network bandwidth and without sacrificing users' privacy.
- We demonstrate the benefits of raw data-sharing over model sharing in recommendation systems through an extensive experimental study. We compared those approaches along both model quality and system metrics. We considered two publicly available datasets (MovieLens latest and 25M [12]), two different models (matrix factorization [13] and DNN [14]) as well as two network topologies (small-world [15] and random [16]), across two decentralized learning algorithms (RMW [8] and D-PSGD [17]), thus demonstrating the generality of our approach.
- We ran REX on Intel Xeon E-2288G CPUs and demonstrate its viability on a real system. More specifically, we show that the overhead of SGX remains low.

The rest of the paper is organized as follows: We present some background on recommenders, ML models and SGX technology in Section II. The design of REX is described in Section III. We present an extensive experimental evaluation demonstrating the benefits of REX over model sharing as well as the overhead of using SGX on a real implementation in Section IV. Related work is surveyed in Section V before concluding in Section VI.

II. BACKGROUND

A. Personalized recommendation

a) Collaborative filtering: While various approaches exist to achieve recommendation, in this paper we focus on the most popular one, namely collaborative filtering (CF) [18]. CF predicts the items a user will be interested in not only from the user's own past activities but those of every other user. We consider a set of n users $U = \{u_1, u_2, \ldots, u_n\}$ and a set of m items $I = \{i_1, i_2, \ldots, i_m\}$. To each user $u \in U$ is associated a profile P_u, which contains the user's opinions on the items she has seen/liked/clicked in the past. The profile P_u is a collection of tuples $< i, v >$ representing the rating v on item i by user u. Ratings may be binary or convey a particular value. The user-item interactions can then be represented as a matrix $A \in \mathbb{R}^{n \times m}$ composed of all n users and m items. Given that users only interact with a few items, the goal of CF is to fill up the matrix with predictions for the missing values.

b) Matrix factorization (MF): MF [13] is among the most popular approaches that decompose the user-item interaction matrix $A \in \mathbb{R}^{n \times m}$ into a product of two matrices of lower dimension $X \in \mathbb{R}^{n \times k}$ and $Y \in \mathbb{R}^{m \times k}$ representing embeddings that summarise user tastes and item profiles, respectively. The matrices X and Y can then be used to directly infer a score. Formally, a MF objective function can be defined as

$$J(X,Y) = \frac{1}{2}||A - XY^T||^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}(a_{ij} - \sum_{l=1}^{k} x_{il}y_{jl})^2$$

where the matrix XY^T is the rank-k approximation of A and the goal is to find X and Y that minimize the error function J. This is achieved by using some method for optimizing objective functions, like stochastic gradient descent (SGD).

Often in practice, matrix A is sparse where only some of the user-item interactions are known. In this case, the problem is modified to find an optimal rank-k approximation for only known values of A. Additional regularisation terms and regularisation parameter λ are added to stabilise the optimisation process. Also, bias vectors $b \in \mathbb{R}^{n \times 1}$ and $c \in \mathbb{R}^{m \times 1}$ are included to account for the fact that some users tend to give higher or lower ratings than others while particular items may receive higher or lower ratings. Including all terms, the loss function $J(X, Y, b, c)$ can be defined as

$$\frac{1}{2}\sum_{(i,j)\in I}(a_{ij} - b_i - c_j - \sum_{l=1}^{k} x_{il}y_{jl})^2 + \frac{\lambda}{2}||X||^2 + \frac{\lambda}{2}||Y||^2$$

where I represents the set of indices for known values in A. Upon learning matrix X and matrix Y, the predictions p_{ij} for user i and item j are obtained as $p_{ij} = X_i \cdot Y_j + b_i + c_j$.

c) Deep Neural Networks: Personalized recommendation can also be accomplished with deep learning approaches (DNN) [2]. A DNN is an artificial neural network that uses several layers of nodes with non-linear activation functions to learn complex functions which capture patterns in the input data to achieve a desired prediction. For the problem of predicting ratings, the data is represented as triplets of the form $< \mathrm{user}_i, \mathrm{item}_j, \mathrm{rating} >$. We add an intermediate embedding layer which can be considered equivalent to the lower-rank matrices described in the MF section above. Each pair $< \mathrm{user}_i, \mathrm{item}_j >$ indexes corresponding embeddings in matrices X and Y, which are concatenated and fed as input to the DNN. Its output, in turn, is unidimensional, representing the predicted rating for the given combination. The learning process occurs on both the weights of the neural network and the embedding matrices. Finally, learnt embeddings are used to predict ratings for unseen user-item pairs.

B. Decentralized recommenders

In a context where the number of items and users grows by the minute, one of the main challenges of centralized recommenders remains their scalability. To tackle this issue, decentralized approaches have been proposed for recommendation purposes in the context of matrix factorization [19], or K nearest neighbours (KNN)-based collaborative filtering [20] but also more generally for numerous machine learning problems [8], [21]–[23]. Most decentralized approaches rely on a gossip protocol to quickly disseminate information, typically

[1] Source code is available at https://github.com/rafaelppires/rex.

model parameters in decentralized learning systems or user profiles in KNN-based systems.

In such a system, we assume that nodes are connected according to a specific topology such as a random graph. Periodically, each node after having performed some local learning task, picks a number of neighbors in the topology to forward them some information [24]. This can be, for instance, the output of local learning tasks. Relying on such a gossiping protocol enables the data or model to be disseminated in the network until convergence is reached. REX relies on such a gossip protocol, which will be detailed in Section III.

C. SGX

Since late 2015, Intel processors come with a hardware shielding subsystem called software guard extensions (SGX). It consists of a user-level protection against any other process in the system, including higher-privileged ones that belong, for instance, to the operating system (OS) or the hypervisor. This is achieved by automatic memory encryption, attestation, integrity and freshness guarantees ensured by hardware.

Applications that leverage this technology must be split into trusted and untrusted partitions. While the former is limited in terms of instructions they can perform, like input and output (I/O), the latter is free to use the entire instruction set. The reason for this constraint is that such instructions require the intervention of higher privileged (and untrusted) entities. As a consequence, transitioning between trusted and untrusted modes entails context switches that involve cryptographic operations, memory copies and translation lookaside buffer (TLB) flushes [25], which incur high performance overheads.

From a software development perspective, transitions from the enclave to untrusted mode are made through outside calls (ocalls), whereas the opposite is called enclave calls (ecalls). Conceptually, these are similar to remote procedure calls (RPCs), where functions and arguments are marshalled together and executed in a separate memory space. Due to the limitation of executing I/O instructions from trusted code, we have to resort to proxying these operations through ocalls. This makes it harder to port legacy applications and libraries in SGX enclaves, as forbidden instructions have to be traced and replaced by such proxies.

D. SGX remote attestation

Attestation is a crucial feature of SGX. It allows for other processes (or other enclaves) to be sure about what code is running inside a *target enclave* (the one being attested) at initialization time. Once trust is established, exchange of sensitive data can take place.

In a nutshell, the target enclave generates a *report* that contains a hash (or *measurement*) of its initial state (code, data and other attributes) computed by hardware upon the enclave initialization. Such report can only be locally verified by another enclave running on the same processor, as it is signed with a key only known by the local platform. In case the verifying node (verifier) is remote, a special platform enclave called quoting enclave (QE) is in charge of verifying the target's *report* and converting it into a *quote*. This, in turn, is signed with a private key before being sent to the verifier. The verifier then checks this signature with the aid of another service, namely data center attestation primitives (DCAP), which finally confirms or refutes the authenticity of the signature.

III. REX

In this section, we provide a detailed description of REX, our novel SGX-based decentralized algorithm. We first describe the establishment of trust between nodes (III-A), the enclave execution of our protocol (III-B) and REX's raw data sharing algorithm (III-C). Finally, we discuss the parallelization aspects (III-D) and implementation details (III-E) of REX.

A. REX *attestation*

For designing REX, we departed from classical decentralized learning algorithms. Decentralized systems are typically composed of processes that share the same code, with no pre-established hierarchy among them. REX is no different in this regard. We however enforce this feature with the help of the SGX attestation protocol (Section II-D). In REX, each pair of SGX nodes must mutually attest themselves before exchanging sensitive data, regardless of when they join the system. This gives the guarantee that all enclaves share the exact same initial code, practically nullifying the possibility of having rogue (or Byzantine) enclaves, as they cannot deviate from the expected behavior.

After a fruitful attestation, each node is convinced about the integrity of each other's initial code and data segments. In addition, a shared secret must be established for confidential communication. In order to obtain this key, we take advantage of the *user data* field in the quote, which is filled with the public key of a elliptic-curve Diffie–Hellman (ECDH) scheme [26]. Once attestation is confirmed, the other node's public key that piggybacked the quote is combined with the local private key for obtaining the shared secret.

At this point, we have a confirmation that the other node runs in a safe and genuine SGX platform, apart from having established a symmetric key for encrypted communication. We however do not yet know *what code* that node is running. This is achieved by comparing the *measurement* within the quote to an expected value. In REX, we require all nodes to run the exact same code, so that this expected value must be equal to the checker's own measurement. If we wanted to allow enclaves with different code-bases, the distinct measurements would have to be either hard-coded in the enclave binary or somehow provided from trusted sources [27], increasing the complexity of the attestation procedure [28].

B. Enclave interface and REX protocol

In REX, we restrict the trusted computing base (TCB), *i.e.*, the amount of code that runs within enclaves, to the strict minimum, so as to reduce the chances of having software bugs and vulnerabilities, which grow with the amount of lines of code. The TCB consists of the C++ standard template library

Algorithm 1: Untrusted code, responsible for the bootstrap of REX and I/O operations

```
1  Procedure initialize:
2      read_dataset()
3      start_network()
4      ecall_init(arguments)
5  Procedure on_receive:
       input : blob
6      ecall_input(blob)
7  Procedure ocall_send:
       input : destination
               blob
8      send(destination, blob)
```

Algorithm 2: Trusted code that runs inside SGX enclaves. It concerns both the attestation and REX protocols

```
1  Procedure ecall_init:
       input : args
   Enclaved
2      local_train_data, local_test_data ← extract(args)
3      initialize_data_structures(args)
4      rex_protocol(∅, ∅)                          // epoch 0
5  Procedure ecall_input:
       input : blob
   Enclaved
6      src, ciphertext ← extract(blob)
7      if attested(src) then
8          shared_key ← get_shared_key(src)
9          data ← decrypt(shared_key, ciphertext)
10         rex_protocol(src, data)
       else
11         attestation_protocol(src)
12 Procedure rex_protocol:
       input : src
               data
   Enclaved
13     if ready_to_train(src, data) then
14         alien_model, alien_train_data ← extract(data)
15         local_model.merge(alien_model)
16         local_train_data.append(alien_train_data)
17         local_model.train(local_train_data)
18         shareable_data ← sample(local_train_data)
19         shareable_model ← get_model(local_model)
20         share(shareable_data, shareable_model)
21         local_model.test(local_test_data)
```

(STL) provided in the Intel SGX software development kit (SDK) and libraries that do not need I/O (json serialization and linear algebra), whereas disk and network operations are kept in untrusted mode.

Once attested, REX nodes execute a typical event-based protocol that collects notifications from their neighbor nodes in the communication graph and perform specific tasks depending on a determined set of application-specific conditions. These tasks, in turn, may generate more events to be shared with fellow nodes. The high-level design of REX is summarized in Algorithms 1 and 2. Algorithm 1 lists the procedures executed in untrusted mode, *i.e.*, those related to the bootstrap and I/O, whereas Algorithm 2 presents the internal enclave structure.

At initialization, REX reads the input dataset, starts the network and initializes the enclave (Algorithm 1, lines 1-4). Upon receiving messages from the network, the untrusted code relays them to the enclave (Algorithm 1, lines 5-6). No privacy threat happens here as only attestation messages, which are not privacy-sensitive, are exchanged in clear text. Any attempt of an attacker to forge attestation messages would fail as it does not have access to secrets protected in the trusted environment. In the opposite direction, *i.e.*, for calls made from inside the enclave, the untrusted code relays encrypted output data to the network interface (Algorithm 1, lines 7-8).

There are two entry points to the enclave code: at initialization (*ecall_init*) and when a message arrives (*ecall_input*). The enclave bootstrap (Algorithm 2, lines 1-4) consists of copying the local partition of the dataset into protected memory, initializing data structures and triggering the first training on the initial data (epoch 0).

Upon reception of a message (Algorithm 2, lines 5-11), its source is identified. Along with the sender identifier, there is possibly a ciphertext that needs to be decrypted. In case the attestation procedure has already been successfully completed, a secret shared key, which is only accessible within the enclave, must have been established with the source node, in which case the message is deciphered and forwarded to the subroutine responsible for the REX protocol. Otherwise, the procedure that takes care of the attestation is called to manage the recognition of the sender.

When *rex_protocol* is called, it checks whether it can perform a training iteration (Algorithm 2, line 13). This happens either in the first training on the local initial data (*i.e.*, $src = \varnothing$ and $data = \varnothing$) or when it has received a message (possibly empty) from all its neighbors. In case one of these conditions is met, raw data and model (possibly empty) are extracted from the input data, and a series of operations take place (Algorithm 2, lines 14-21). We classify them into 4 steps:

- **merge** (lines 15-16):
  - if $alien\_model$ is not empty, it is merged with the local model (see Section III-C).
  - if $alien\_train\_data$ is not empty, all non-duplicate data items are appended to the local training data store.
- **train** (line 17): SGD iterations are performed on the local training data in order to improve the local model.
- **share** (lines 18-20):
  - if raw data sharing is activated, the local data store is sampled and the selected data items are shared with neighbors according to the sharing algorithm in place.
  - if model sharing is on, the local model is shared with neighbors following the selected sharing algorithm.
- **test** (line 21): predictions are made for data items contained in $local\_test\_data$, which was not used for training, and then compared to the ground truth in order to verify the quality of the current model.

C. Raw data sharing

REX speeds up convergence by sharing raw data with neighbors as opposed to FL and DLS that share model parameters. The amount of data is parametrizable (as a program argument) and randomly selected from the local raw data store (Algorithm 2, line 18), which is kept inside protected memory.

We support two algorithms which determine the set of neighbors that will receive the raw data: either a random one (RMW) or all of them (D-PSGD). This is inspired by the way

models are shared in two decentralized learning schemes, that we describe next.

1) Random model walk (RMW): In RMW, or Gossip learning [8], each node randomly selects one of its neighbors to send its current model. Upon receiving a model, a node averages it with its own and improves the model by training upon its local data. When RMW is active, REX sends the raw data (instead of the model) to the same randomly selected neighbor.

2) Decentralized parallel SGD (D-PSGD): In this approach [17], each node sends its model to all neighbors. Along with the model, it also sends an integer corresponding to its degree (*i.e.*, how many neighbors the sender has). Upon receiving a model, the destination node merges it into its own through a weighted average based on the degrees (we use Metropolis-Hastings weights [29]). When a node has no embedding for a given user or item, we consider only those of its neighbors. When D-PSGD is active, REX sends to all neighbors a sampling of the local raw data.

D. Parallelization

REX parallelization can be seen from both multi- and single-node perspectives. Being decentralized, REX nodes independently and concurrently run on multiple machines, periodically synchronizing with neighbors according to the network topology and sharing algorithms. Synchronization barriers are established when a given node receives a message from all its neighbors, thereby triggering subsequent iterations of the protocol. In the current version, we do not tackle fault tolerance aspects. We leave failure detection (*e.g.*, heartbeats and timeouts) for future work.

Within a single node, REX executes *merge-train-share-test* tasks sequentially. This is a requirement in model sharing schemes because each task depends on the result of the previous one. REX could however execute *share* in parallel with the other tasks, since raw data sharing is independent of computing steps. Although our implementation currently lacks this feature, it could only further increase the advantages of leveraging REX due to increased parallelism.

E. Implementation

We implemented REX in about 4200 lines of code in C++. Additionally, we used Intel SGX SSL [30] for cryptographic algorithms, a json library [31] for serialization during attestation, ZeroMQ [32] for communication, and Eigen [33] for sparse matrices and vectors.

For the comparisons between SGX and native (*i.e.*, without SGX), we use the same code-base, but compiled with a different set of flags and linked to distinct libraries. Specific calls to the SGX SDK or routines that only make sense in enclave mode (such as attestation) are either filtered out with pre-processor directives or replaced by alternatives.

Sharing data brings the question of how much to share in every epoch. We treat this as another hyperparameter and experiment with several different values in order to pick one that fits well according to accuracy versus time comparisons.

TABLE I
DATASETS.

Dataset	Ratings	Items	Users	Last updated
MovieLens Latest	100 000	9000	610	2018
MovieLens 25M*	2 249 739	28 830	15 000	2019

*We capped the number of users (originally at 160 000), as our intent was to stay around the memory limits of our SGX servers. Ratings and items correspond to this truncated dataset.

By selecting a random sample of required data points to share, we make the data sharing a stateless procedure. Thus, nodes may send the same data points more than once, although the probability of duplicates decreases as the data size increases.

Another point to note when nodes share data is the amount of processing time required in every epoch, which would continually increase with the growth of input training data. This results in very long training times as the model begins to reach convergence. We solve this by fixing the number of batches taken into account in every epoch to a predefined value. Hence, each node takes a fixed number of SGD steps in every epoch regardless of the data available. As a result, the training time per epoch remains constant throughout the learning process.

IV. EVALUATION

We now present an extensive evaluation of REX. We start with simulated scenarios for DNN (50 nodes) and MF (610 and 50 nodes) to demonstrate the benefits of raw data sharing. Afterwards, we focus on a distributed setup of 8 nodes running on 4 SGX servers (2 processes per machine), where we evaluate the enclave overheads. We describe next the experimental setup followed by the results and corresponding assessment.

A. Experimental setup

Apart from the distinct decentralized learning schemes we presented in Section III-C, we use varied datasets in terms of size, and two network topologies. In this section, we also describe the metrics and experimental methodology we employed.

1) Datasets: We used MovieLens [12] datasets in our experiments, as shown in Table I. It consists of collections of movie ratings made by thousands of users in a website. Users' ratings correspond to how much they appreciated a given movie, on a scale that ranges from 0.5 to 5.0, graphically represented by 5 stars which may be fully or partially filled.

2) Network topologies: To assess our decentralized recommender system under different topologies, we chose Small World and random (Erdős-Rényi), which we briefly describe next.

a) Small World: This topology tries to mimic the relations that happen in real situations (*e.g.*, social networks), where nodes are connected to small groups, out of which some may have far-fetched connections [15], according to the topological distance in the network. Each node has then close connections and a few far-fetched ones. As a consequence,

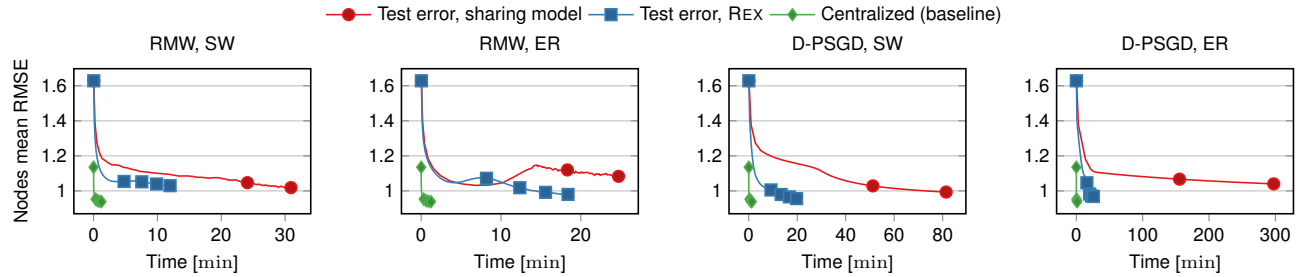

Fig. 1. One node per user — MF model. The figure charts evolution of test error with simulation elapsed time. REX converges much faster than MS across all four cases, while the centralized baselines remains fastest as expected. Markers on the plots are spaced 50 epochs.

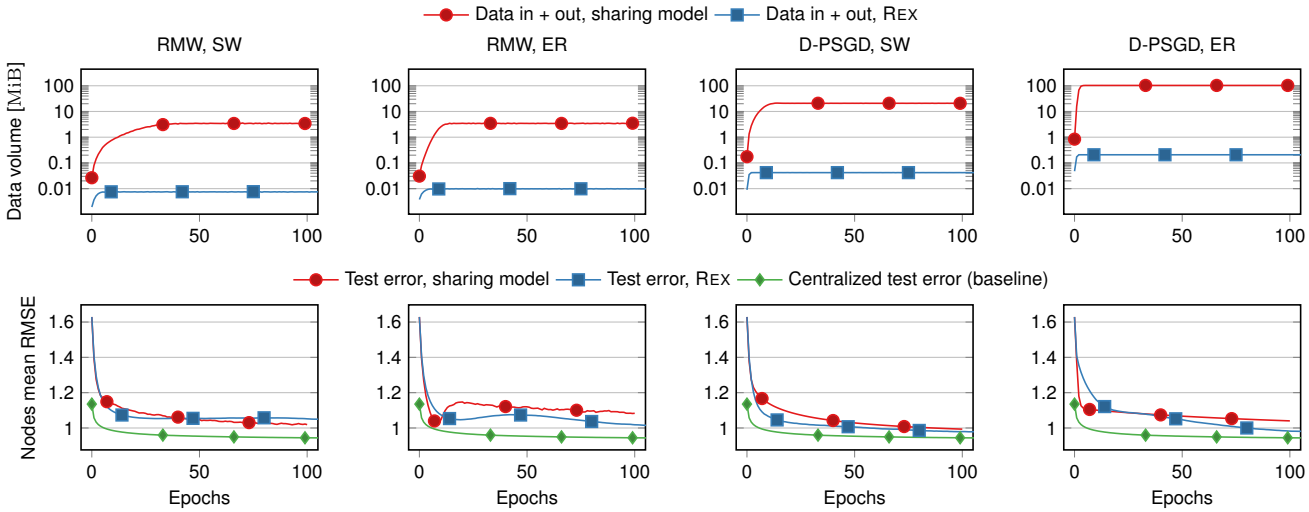

Fig. 2. One node per user — MF model. In correspondence to Figure 1, this figure charts network usage and evolution of test error across epochs. Data exchanged in REX is two orders of magnitude lower than MS across all four cases (Row-1). The test error in REX evolves similarly to MS across epochs (Row-2) but each epoch runs significantly faster since only data is shared. Finally, the centralized baselines remains fastest in all cases.

most nodes can reach each other in a small number of hops. Technically, these graphs have low diameter and high clustering coefficient. In our experiments, we used a library called boost [34] to generate a SW topology taking as input 3 parameters: the size of the graph (610 and 50 here), the number of close connections (set to 6 in our experiments) and a probability of far-fetched connections (we set it to 3%).

b) Erdős-Rényi: This topology consists of a random graph, where each edge is included in the graph with a given probability p [16]. In comparison to Small World, these graphs may have larger diameters and lower clustering coefficients. Although its construction mechanism can result in a disconnected graph (*i.e.*, with multiple components), we ensure to make it connected by adding the missing edges. In our experiments, p is set to 5%.

3) Machine Learning models:

a) MF: In the matrix factorization experiments, we split the dataset into train (70%) and test (30%) sets. We set the learning rate to $\eta = 0.005$, the regularization parameter to $\lambda = 0.1$ and the embedding dimension to $k = 10$. These values were obtained by several trials in the centralized setup. The nodes share 300 data points per epoch.

b) DNN model: For the DNN model, we have a setup with 50 nodes where each one holds the data of 12 or 13 users. We use the Adam optimizer [35] with a learning rate of $\eta = 0.0001$, weight decay of 0.00001 and set the embedding dimension to $k = 20$. Following the embedding layer, the model has 4 hidden layers (linear + ReLU), dropout layers, and a final ReLU activation layer. The dropout rate for the embedding layer is 0.02 while for the first two hidden layers it is 0.15. The described DNN model has 215 001 model parameters in total. Finally, in each epoch, the nodes share 40 data points.

4) Metrics: We measure the benefits of REX over 3 dimensions: training time, network traffic and test error as the root mean square error (RMSE). Our goal is to show that raw data sharing renders better results than model sharing for all of them. With respect to test error, it reaches a given value in a shorter amount of time. We also evaluate REX on real SGX servers, where we measure the memory consumption as it represents a scarce resource in such environments.

5) Methodology: We start by evaluating the scenario where each node holds the data of one user. This represents the situation where users initially have only their own data, *i.e.*, what they produced. Even though we use item ratings for recommendation, this situation would similarly apply, *e.g.*, to text messages or pictures taken in a person's smartphone.

Next, we experiment with a setup where each node holds

TABLE II
ONE NODE PER USER. SPEEDUP IN TIME ACHIEVED BY REX COMPARED TO MODEL SHARING (MS) FOR A GIVEN TARGET ERROR.

Setup	Error target	REX [min]	MS [min]	REX speed-up
D-PSGD, ER	1.04	16.3	297.5	18.3×
RMW, ER	1.08	2.1	24.7	11.5×
D-PSGD, SW	0.99	10.8	81.4	7.5×
RMW, SW	1.03	12.0	27.4	2.3×

the data for several users, simulating a situation of distributed servers that are able to provide recommendations to these cohorts of users. For example, SGX servers in geographically-distributed data centers serving distinct clusters of users.

In our simulated experiments, we used servers with processor Intel Xeon E5-2630 v3 at 2.40GHz and and 128 GiB RAM running Ubuntu 20.04.2 LTS kernel 5.4.0-72. For the SGX ones, we used 4 servers with processor Intel Xeon E-2288G CPU at 3.70 GHz and 64 GiB RAM running Ubuntu 18.04.4 LTS kernel 4.15.0-117 and the Intel SGX SDK v. 2.9.1.

B. REX *versus model sharing*

We now present our experiments. They are organized according to the experimental setting: one or multiple users per node, and those conducted on SGX hardware.

a) One node, one user: Figure 1 presents the evolution of test error with respect to the simulation elapsed time. We use centralized execution as the baseline. Note that all scenarios converge to about the same error value, meaning that they are functionally equivalent.

Concerning the time to achieve a determined target error, we clearly observe that REX is always better than sharing models. To support this claim, we compile in Table II the values for an error target (chosen as the final value achieved by MS scheme), the times at which it was achieved and the ratio between timestamps. REX reaches speed-ups of up to 18.3× (D-PSGD, ER). Additionally, we observe that D-PSGD is much slower than RMW. Whereas it took 5 h to complete a simulation for D-PSGD ER, the longest RMW simulation took about 30 min for the same number of epochs. This is due to the broadcasting nature of D-PSGD in contrast to the random neighbor unicast of RMW (Section III-C).

The first line of charts in Figure 2 explains one reason why REX achieves the same results in less time. In terms of the volume of exchanged data, we observe that for all scenarios, sharing models was more than 2 orders of magnitude more expensive than REX. This happens because recommender systems are trained upon small data. A raw data item is represented by a triplet containing the user and item identifications, along with the rating. The model, on the other hand, is large. In the case of MF, a data item is associated to two feature vectors (or embeddings) related to the user and item in the triplet. Each of these vectors alone is already larger than the data item to which they are associated.

To evaluate the impact of the size of feature vectors, we ran the scenario with D-PSGD, SW for different lengths of embeddings and the equivalent REX execution. Results are shown in Figure 3. Each scenario is run for a fixed number of epochs (400). As expected, REX is not affected by feature vectors (2^{nd} row, 3^{rd} column) in terms of network load because it does not share models. When models are shared (1^{st} row), we observe that network load linearly increases at little benefit to convergence time upon increasing the size of feature vectors. Thus, in our experiments, we set them to a fairly small size, equal to 10. We found this to be a good compromise between having a reasonably accurate model and avoiding introducing bias towards our data sharing proposal by making models even bigger.

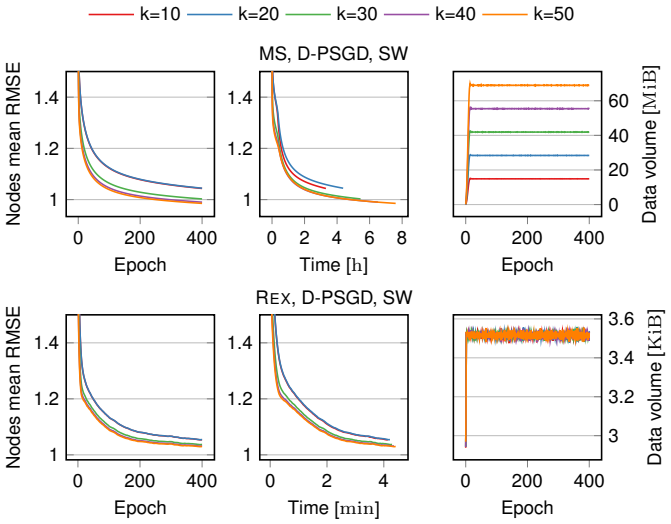

Fig. 3. Effect of varying feature vector size for D-PSGD, SW — MF model. All scenarios ran for fixed 400 epochs. Columns 1 and 2 chart test loss while column 3 charts data exchanged per node per round. For the MS case (row-1), increasing feature vector size provides little benefit in convergence time for the corresponding linear increase in network load. The impact on convergence of REX (row-2) also remains little while the network load remains constant since only data is shared.

In terms of epochs, the charts in the second line of Figure 2 show that decentralized settings need more iterations in order to achieve the same error target as in the centralized equivalent. This is inherent to their lack of global knowledge. While the global model can uniformly improve for all the dataset at each iteration, decentralized alternatives can only count on local data plus the interactions with closest neighbors, thus delaying the progress of information dissemination. In any case, even though REX and model sharing take roughly the same number of epochs to converge, REX is much better in terms of network and time.

b) Multiple users per node: Our following experiment tested our system in a second scenario: when the data of multiple users is initially partitioned across a number of servers (Section IV-A5). In this setup, simulation times were much faster due to the fewer nodes through which information had to propagate. We partitioned the ratings of the 610 users through 50 nodes and got similar results with respect to model and raw data sharing. This time, however, ratios were more modest. The results displayed on Figure 4 and Table III summarizes

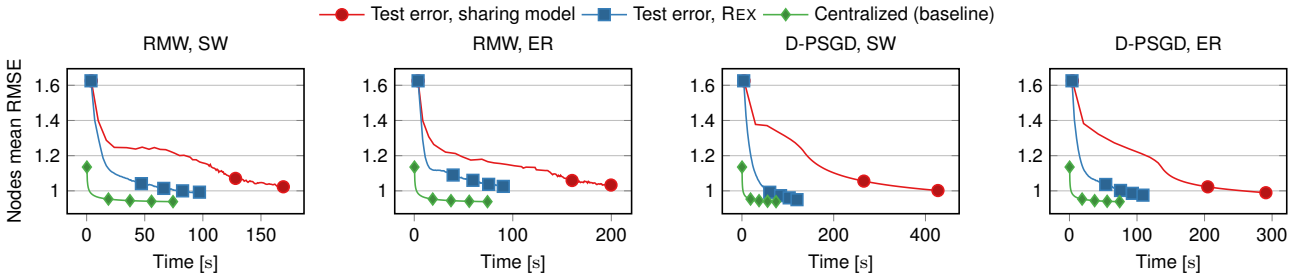

Fig. 4. Multiple users per node — MF model. The figure charts evolution of test error with simulation elapsed time. Similar to one node per user scenario, REX converges much faster than MS across all four cases, while the centralized baselines remains fastest as expected. (Plots markers are spaced 50 epochs)

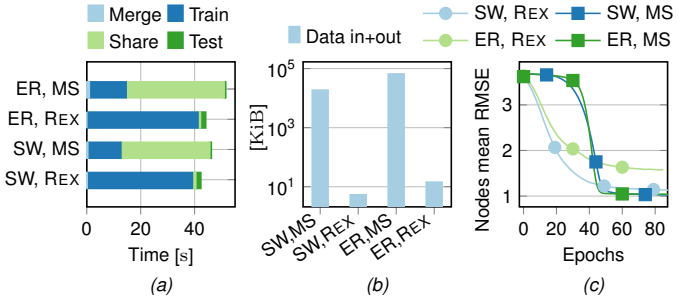

Fig. 5. Multiple users per node — DNN model. *(a)* Time breakout of stages within an epoch (average across all nodes) - REX is slightly faster. *(b)* Data volume exchanged per epoch - REX exchanges significantly less data than MS. *(c)* Test error evolution per epochs - For SW, REX converges faster than MS while achieving similar test error whereas for ER, REX performs slightly worse.

TABLE III
MULTIPLE USERS PER NODE. SPEED-UP IN TIME ACHIEVED BY REX COMPARED TO MODEL SHARING (MS) FOR A GIVEN TARGET ERROR.

Setup	Error target	REX [s]	MS [s]	REX speed-up
D-PSGD, ER	0.99	87.8	292.5	3.3×
RMW, ER	1.03	82.9	200.6	2.4×
D-PSGD, SW	1.00	57.0	430.4	7.5×
RMW, SW	1.02	61.1	170.1	2.8×

them. The reason why speedup is lower for multiple users per node is due to data concentration. As each node holds more data, we need less iterations to achieve a given target error, lowering the impact of network load, and hence REX.

We then experimented with our DNN recommender. It was developed in 1495 lines of Python and uses PyTorch [36] for the DNN, ZeroMQ [32] for communication and D-PSGD (Section III-C) as sharing scheme. Figure 5 displays the results.

DNN results match the previous ones and show a lower epoch duration for REX (Figure 5*(a)*), even though the difference is slightly smaller. Similarly, with respect to the amount of data exchanged, we observe that volumes are orders of magnitude larger for model sharing (Figure 5*(b)*) in comparison to REX. Concerning the test error (Figure 5*(c)*), we observe that results vary according to the topology. While small world (SW) achieves very similar results between the two sharing schemes, the random graph (ER) performs slightly worse for REX, *i.e.*, it achieves a larger error after a fixed number of epochs. We conjecture the reason to be related to the sparsity of the random graph, less connected than small world in this 50-node scenario. While MS encapsulates and propagates more information by training on entire local data, DS exchanges limited knowledge (contained only in the shared data points).

C. SGX experiments

Next, we measure REX in SGX-capable machines in a distributed setup. We used a 4-node network and ran 2 processes per machine in a fully connected setup, *i.e.*, 8 nodes and 28 pair-wise connections. Results are shown on Figs. 6 (low memory usage) and 7 (memory beyond EPC).

In Figures 6*(a)* and 7*(a)*, we see the time breakdown according to each step of the distributed training process: merge, train, share and test. The values correspond to the mean time that each step took per epoch across all nodes. We observe that sharing data (REX and Native, DS) is always faster in comparison to exchanging models (MS). The reason is the extra time needed for merging and sharing models. As discussed in Section III-C, the contributions of a group of neighbors are averaged together when models are exchanged. While this procedure obviously takes some time, it is completely bypassed by REX. Although we still need to check for duplicates, new data items are simply dumped into the local store with no further processing. This is much faster than locating the relevant embeddings, attributing weights for neighbor contributions and performing the average.

When it comes to the duration of sharing, the difference is explained by the size of raw data and models (see Section IV-B). In Figures 6*(b)* and 7*(b)*, we see the average data volume exchanged per node per epoch. As previously shown, the data volume when sharing models is orders of magnitude higher, which justifies the extra overhead in sharing times for MS when compared to REX. Charts *(c)* and *(d)* of both Figures 6 and 7, that present wall-clock time versus test error, were obtained in a non-simulated environment, *i.e.*, with real network exchanges. They confirm the same pattern between MS and REX found previously in our simulations.

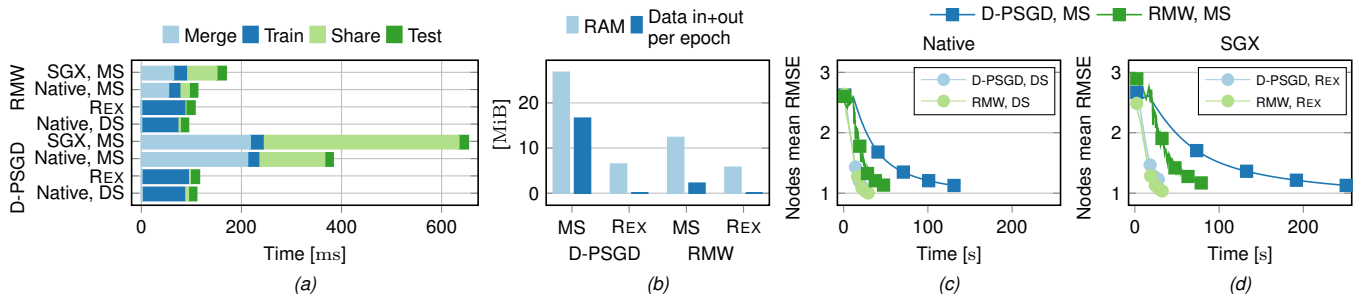

Fig. 6. Performance comparison with and without SGX for low memory usage (MovieLens Latest with 610 users) — MF model. *(a)* Time breakdown of steps in an epoch - Duration of merging and sharing is very low for REX as compared to MS since only data is exchanged while the training duration remains high given the SGX procedures (Section II-C). However, altogether REX is faster than MS. The native equivalent runs faster for both DS and MS as expected. *(b)* Memory and network usage - REX exchanges much less data and requires less memory than MS. *(c)* and *(d)* Convergence speed (marker each 50 epochs) - REX converges faster than MS similar to previous simulated experiments with very little overhead.

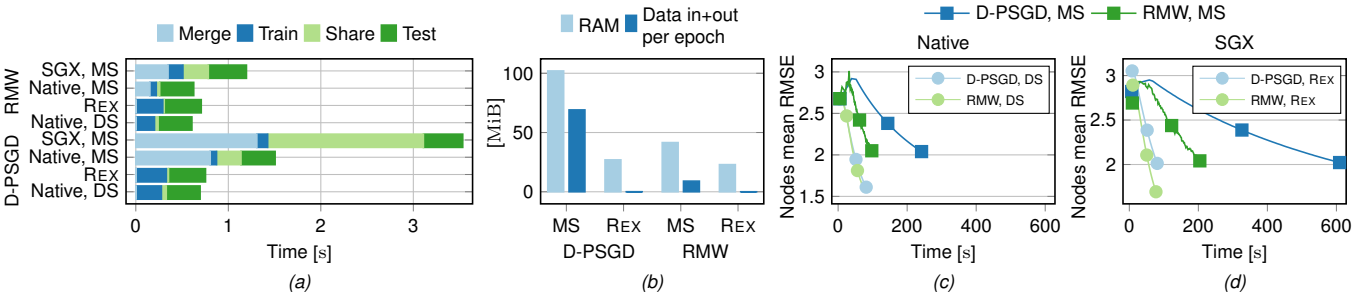

Fig. 7. Performance comparison with and without SGX for memory usage beyond EPC limit (MovieLens 25M with 15k users) — MF model. The observed trends for *(a)* Time breakdown of steps in an epoch, *(b)* Memory and network usage, *(c)* and *(d)* Convergence speed remain very similar to the scenario of low memory usage in Figure 6.

D. SGX and memory usage

When putting into perspective the native (*i.e.*, without SGX) and SGX experiments, we notice some slowdown in execution times for the latter. Note that in native executions, data transmissions are in plaintext and there is no hardware protection. Both raw data and models are therefore vulnerable in this case. In the experiments of Figure 6, we use the same dataset as in Section IV-B, *i.e.*, MovieLens with 610 users. For those, the SGX overhead in terms of execution time varies from 5 % (REX) to 70.5 % (model sharing).

The reasons for the difference between SGX and native executions lie in the intrinsic way enclaves are designed (see Section II-C), specially with respect to memory usage, transitions between the trusted and untrusted environments and all cryptographic and integrity operations involved in the process. This is why the sharing step presents the biggest difference when we compare its times for SGX and native, *i.e.*, because it simultaneously involves I/O, cryptographic operations and intensive memory usage.

Interestingly enough, we consistently observed a puzzling exception to this pattern. For REX, the sharing step was slightly faster in the enclave execution. We investigated the reason and found that the data sampling function was the source for this time difference. The reason lies in the way memory is allocated. While all enclave memory pages are obtained at initialization time, the native execution asks for more pages on-demand, therefore incurring in extra system calls to be serviced by the operating system during the sharing step, rather than in the bootstrap. This behavior was one constraint of the first version of SGX (*i.e.*, the version of our machines), which was latter alleviated with the introduction of hardware support for dynamic memory allocation inside an enclave [37].

In Figure 6*(b)*, along with network data volume exchanged, we see the amount of memory used throughout the execution. The memory usage was measured with valgrind [38] for the native implementation. As we use identical codes in the native and SGX implementations (Section III-E), the same measurements apply to the enclave. The values correspond to the average heap usage sampled after the initial dataset input (when there is a peak of memory usage), after which it remains fairly constant. Since the dataset input happens in untrusted mode, this initial peak does not affect the SGX execution and was therefore discarded for the sake of estimating the enclave memory usage.

In order to evaluate REX in a more challenging setup, we took the MovieLens 25M dataset and limited the amount of users to 15 000. This number was chosen because of the memory usage footprint it caused in our experiment. More precisely, we wanted to have a condition where the enclave page cache (EPC) is overcommitted. The SGX machines at our disposal have an EPC of 128 MiB, out of which only 93.5 MiB are available for all enclaves running in each machine [39]. In these experiments (Figure 7), we reach more than twice the EPC for D-PSGD MS and roughly the EPC limit for RMW

TABLE IV
THE TABLE PRESENTS OVERHEAD IN EXECUTION TIME FOR SGX W.R.T NATIVE. PRESENTED ALONGSIDE IS THE MEMORY USAGE WHICH EXPLAINS THE OVERHEAD. FOR THE MS CASE, THE OVERHEAD IS SIGNIFICANT (UP TO 135%) BUT REMAINS LOW FOR REX (UP TO 17%).

Setup	610 users		15 000 users	
	RAM [MiB]	Overh. [%]	RAM [MiB]	Overh. [%]
RMW, REX	11.5	14	45.9	17
RMW, MS	24.7	51	83.1	91
D-PSGD, REX	12.9	5	53.9	8
D-PSGD, MS	53.6	70	204.0	135

MS. Both of them had considerable increase in the overhead when compared to the previous scenario. Table IV summarizes these results, which were obtained by comparing average time per epoch of SGX over native.

E. Discussion

In this section, we recall REX motivations and justify the absence of scalability experiments in this paper. We address next the limitations of REX, namely, SGX vulnerabilities, poisoning attacks, vendor lock-in and memory constraints. Additionally, we list some future research avenues we would like to pursue.

a) Recap on REX *motivations:* With REX, we demonstrated our three-dimensional goal of achieving at once: *i*) privacy, with SGX enclaves; *ii*) accuracy, shown through test loss in terms of RMSE; and *iii*) scalability, as a consequence of shorter convergence times and lower network usage (see *b* below). Our exciting results pave the way for further investigation on privacy-preserving raw data sharing in decentralized systems. From a broader perspective, this would enable independent users to collaboratively train ML models in a secure and scalable manner, as no dependence on centralized service providers is necessary. This comes as an alternative to current recommender systems, which belong to giant tech companies who have access to private data of billions of users.

b) Scalability evaluation: Although crucial in decentralized systems, scalability is a direct consequence of network topology and sharing algorithm [40]. A fully connected topology scales poorly due to excessive connections, whereas RMW scales better than D-PSGD because of frugal network usage. This is however orthogonal to REX, whose positive impact on scalability is secondary, *i.e.*, it is a side-product of savings on network transfer. Consequently, in this paper, we chose to evaluate metrics directly impacted by the distinguishing design principles of REX, *i.e.*, data sharing and SGX. Given that we consistently achieve shorter convergence times and lower network usage, REX can only improve decentralized recommender systems in terms of scalability. In addition, our resource limitations in terms of SGX hardware currently deters a proper scalability study of REX.

c) SGX weaknesses: Hardware-enforced attestation at the application granularity is currently available only with SGX. Although it guarantees that only trustworthy code runs inside enclaves, it does not prevent Byzantine users from subverting the system through poisoned input data, for instance. Such attacks, along with those based on denial of service and side-channels, are not covered by the SGX threat model and therefore out the scope of this work. Despite a few published attacks to SGX [41]–[44] and the mitigations that followed them, manufacturers keep investing and improving TEEs, which is a sign that such technology will keep evolving and hopefully will reach a maturity point when the feasibility of attacks will be very limited and swiftly neutralized.

d) Vendor lock-in and memory constraints: Given that Intel Xeon platforms are *full steam ahead* with SGX [45], we believe that REX represents a viable solution for the future. We hope however that multi-vendor groups, such as the trusted computing group (TCG) [46], will eventually come up with standardized inter-operable TEEs, so that vendor lock-in will no longer be an issue. With respect to memory limits, Intel recently announced their new line of server processors with EPC capacity of up to 512 GiB, expandable to 1 TiB when using two chips in one machine [47]. This will likely allow this technology to be widely used for memory-eager applications.

Despite the technological infrastructure that enables REX, the key takeaway of our proposal lies on the volume of raw data in perspective to models. Apart from this, speed-ups can also come from the fact that sharing data can happen in parallel with the training, unlike model sharing which requires costlier aggregation and synchronization.

e) Recommenders versus other ML applications: Our choice on recommender systems was not incidental. Among the reasons why we obtained considerable time and network gains is the high degree of sparsity in user profiles in such applications as well as the small size per data sample. Because of that, we are interested in evaluating to which extent the same applies to other ML applications (*e.g.*, image classification, sentiment analysis, natural language processing).

Concerning model sharing, one could further reduce the amount of data that is exchanged as models by using gradient compression [48]–[50]. Since recommendation systems are based on ratings that can take very few values (only 10 in the case of MovieLens, *i.e.*, from 0.5 to 5.0 in steps of 0.5), data sharing in this area is also highly compressible. In other domains, where data may already be compressed at the origin (*e.g.*, pictures in JPG format), the compression rates would shrink. For these reasons, we leave for future work the assessment of the impact of gradient compression with respect to the choice of model or raw data sharing in decentralized training. Moreover, data non-iidness is well-known to have a significant impact on the convergence of models in DLS. We also plan on studying the impact of raw data sharing in the context of pathological non-iid datasets.

V. RELATED WORK

To the best of our knowledge, we are the first to use SGX in a decentralized secure recommender system and leverage raw data sharing as a way to speed up training. Nevertheless, privacy in recommender and decentralized learning systems was previously tackled, which we cover next.

a) Differential privacy and homormorphic encryption: Until the rise of TEEs, most practical approaches involved

differential privacy or homomorphic encryption (HE). In this sense, Bellet et al. [51] propose differentially-private algorithms for decentralized systems, where a privacy budget ϵ is set in order to determine how much noise is added to data in order to prevent the disclosure of privacy-sensitive information. Boutet et al. [52] propose the design of a decentralized recommender ensuring differential privacy through randomized protocols and a profile obfuscation mechanism. Danner et al. [53], in turn, combine heavy compression and a tree-based homomorphic encryption scheme to make a group of nodes jointly compute gradient sums in the context of a mini-batch SGD. Nikolaenko et al. [54] propose an approach to perform privacy-preserving matrix factorization through garbled circuits. Common to this line of work, one needs to handle the trade-off between accuracy, efficiency, and privacy. This is precisely what REX avoids by using TEEs.

b) TEE-based decentralized systems: Using SGX enclaves in decentralized systems was tackled in the domain of web-search relay networks. Given the shielding and attestation capabilities of SGX, enclaves were used to conceal user queries in such a way that adversaries are not able to inspect or subvert the behavior of relays. SGX-Tor [55] shows that this is achieved with low overheads. In addition, Cyclosa [56] provides obfuscation mechanisms to frustrate Web-search engines attempts of re-identifying users. These however do not involve gossip protocols to jointly compute ML models as REX does.

c) TEE and ML: When it comes to ML, many works use TEEs for security and privacy [57]–[59]. Slalom [60] uses the TEE to keep the secrecy of linear layers in DNNs before leveraging hardware accelerators on concealed data. Vessels [61] focuses on memory efficiency within enclaves. In the context of FL, PPFL [62] uses SGX on the server-side and ARM TrustZone on worker nodes, whereas ShuffleFL [63] protects the transmission of gradients with hardware enclaves along with a randomized scheme to prevent side-channel attacks. Unlike REX, they are not targeted to decentralized systems.

VI. CONCLUSION

In this paper, we addressed privacy in distributed collaborative filtering systems and proposed the design, evaluation and implementation of REX, the first SGX-based decentralized recommender. In the process, we have debunked the myth that there is an inescapable trade-off between accuracy or efficiency and privacy in collaborative filtering-based recommenders.

Effectively, by leveraging TEEs, REX enables raw data sharing among participants of decentralized systems without compromising user's privacy. This contrasts with the traditional parameter sharing of federated learning and decentralized gossip-based approaches, which may yield privacy breaches.

We evaluated REX across two network topologies, two model merging schemes and datasets of distinct sizes. Moreover, we tried scenarios with one and multiple users per node. We presented results for both native setting (without SGX) and in a (4 machine) distributed SGX environment. Our results over all these settings consistently demonstrate that REX improves up to $18.3\times$ the training time, mostly due to network traffic which is significantly reduced. Our implementation also demonstrates that the overhead of using TEEs remains negligible. At a time where SGX is starting to be available in major cloud providers, we believe that REX is a credible approach to provide efficient, accurate recommendations in a wide range of applications without sacrificing on users' privacy.

REFERENCES

[1] C. A. Gomez-Uribe and N. Hunt, "The Netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems*, vol. 6, no. 4, 2016. DOI: 10.1145/2843948.

[2] U. Gupta, C. Wu, X. Wang, *et al.*, "The architectural implications of facebook's DNN-based personalized recommendation," in *IEEE HPCA*, 2020. DOI: 10.1109/HPCA47549.2020.00047.

[3] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. Artif. Intell.*, vol. 2009, 2009. DOI: 10.1155/2009/421425.

[4] A. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: Scalable online collaborative filtering," in *WWW*, Banff, Alberta, Canada, 2007, pp. 271–280. DOI: 10.1145/1242572.1242610.

[5] K. Lauter, M. Naehrig, and V. Vaikuntanathan, *Can homomorphic encryption be practical?* Crypt. ePrint Archive, 2011. eprint: https://eprint.iacr.org/2011/405.

[6] X. Zhu and Y. Sun, "Differential privacy for collaborative filtering recommender algorithm," in *ACM IWSPA*, New Orleans, Louisiana, USA, 2016. DOI: 10.1145/2875475.2875483.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *PMLR AISTATS*, vol. 54, 2017. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html.

[8] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, 2013. DOI: 10.1002/cpe.2858.

[9] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients - how easy is it to break privacy in federated learning?" In *NeurIPS*, 2020, pp. 16 937–16 947. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf.

[10] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, "White-box vs black-box: Bayes optimal strategies for membership inference," in *ICML*, 2019, pp. 5558–5567. [Online]. Available: http://proceedings.mlr.press/v97/sablayrolles19a/sablayrolles19a.pdf.

[11] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *ACM CCS*, Toronto, Canada, 2018, 619–633. DOI: 10.1145/3243734.3243834.

[12] Grouplens, *Movielens datasets*, 2021. [Online]. Available: https://grouplens.org/datasets/movielens/.

[13] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, 1998.

[16] P Erdös and A Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[17] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel SGD," in *NIPS*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf.

[18] M. D. Ekstrand, J. Riedl, and J. A. Konstan, "Collaborative filtering recommender systems," *Found. Trends Hum. Comput. Interact.*, vol. 4, no. 2, pp. 175–243, 2011. DOI: 10.1561/1100000009.

[19] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized recommendation based on matrix factorization: A comparison of gossip and federated learning," in *ECML PKDD*, Würzburg, Germany, 2020. DOI: 10.1007/978-3-030-43823-4_27.

[20] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A. Kermarrec, "WHATSUP: A decentralized instant news recommender," in *IEEE IPDPS*, 2013, pp. 741–752. DOI: 10.1109/IPDPS.2013.47.

[21] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich, "A unified theory of decentralized SGD with changing topology and local updates," in *ICML*, PMLR, 2020. [Online]. Available: https://proceedings.icml.cc/paper/2020/file/6c2e49911b68d315555d5b3eb0dd45bf-Paper.pdf.

[22] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, "Decentralized collaborative learning of personalized models over networks," in *AISTATS*, vol. 54, PMLR, 2017. [Online]. Available: http://proceedings.mlr.press/v54/vanhaesebrouck17a/vanhaesebrouck17a.pdf.

[23] G. Nadiradze, A. Sabour, P. Davies, S. Li, and D. Alistarh, "Asynchronous decentralized SGD with quantized and local updates," in *NeurIPS*, 2021. [Online]. Available: https://papers.nips.cc/paper/2021/file/362c99307cdc3f2d8b410652386a9dd1-Paper.pdf.

[24] M. Jelasity, S. Voulgaris, R. Guerraoui, A. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM TOCS*, vol. 25, no. 3, p. 8, 2007. DOI: 10.1145/1275517.1275520.

[25] V. Costan and S. Devadas, *Intel SGX explained*, Cryptology ePrint Archive, Report 2016/086, 2016. eprint: https://eprint.iacr.org/2016/086.

[26] M. A. Strangio, "Efficient Diffie-Hellmann two-party key agreement protocols based on elliptic curves," in *ACM SAC*, Santa Fe, New Mexico, 2005. DOI: 10.1145/1066677.1066755.

[27] F. Gregor, W. Ozga, S. Vaucher, *et al.*, "Trust management as a service: Enabling trusted execution in the face of Byzantine stakeholders," in *IEEE/IFIP DSN*, 2020. DOI: 10.1109/DSN48063.2020.00063.

[28] G. Chen and Y. Zhang, "MAGE: Mutual attestation for a group of enclaves without trusted third parties," in *31st USENIX Security*, 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/chen-guoxing.

[29] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *Journal of Parallel and Distributed Comp.*, vol. 67, no. 1, 2007. DOI: 10.1016/j.jpdc.2006.08.010.

[30] Intel, *Intel software guard extensions ssl*, 2021. [Online]. Available: https://github.com/intel/intel-sgx-ssl.

[31] N. Lohmann, *JSON for modern C++*, 2021. [Online]. Available: https://github.com/nlohmann/json.

[32] L. Boccassi *et al.*, *Zeromq: An open-source universal messaging library*, 2021. [Online]. Available: https://zeromq.org.

[33] G. Guennebaud *et al.*, *Eigen: C++ template library for linear algebra: Matrices, vectors, numerical solvers, and related algorithms*, 2021. [Online]. Available: https://eigen.tuxfamily.org.

[34] J. Siek *et al.*, *The boost graph library (BGL)*, 2021. [Online]. Available: https://www.boost.org/doc/libs/1_76_0/libs/graph/doc/index.html.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[36] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[37] B. C. Xing, M. Shanahan, and R. Leslie-Hurd, "Intel software guard extensions (intel SGX) software support for dynamic memory allocation inside an enclave," in *HASP*, Seoul, Republic of Korea, 2016. DOI: 10.1145/2948618.2954330.

[38] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *ACM PLDI*, San Diego, California, USA: Association for Comp. Machinery, 2007. DOI: 10.1145/1250734.1250746.

[39] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "SGX-aware container orchestration for heterogeneous clusters," in *IEEE ICDCS*, Vienna, Austria, 2018. DOI: 10.1109/ICDCS.2018.00076.

[40] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04, Portland, Oregon, USA, 2004, 367–378. DOI: 10.1145/1015467.1015508.

[41] J. Van Bulck, M. Minkin, O. Weisse, *et al.*, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security*, Baltimore, USA, 2018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[42] V. Crăciun, P. A. Felber, A. Mogage, E. Onica, and R. Pires, "Malware in the SGX supply chain: Be careful when signing enclaves!" *IEEE TDSC*, 2020. DOI: 10.1109/TDSC.2020.3024562.

[43] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "Crosstalk: Speculative data leaks across cores are real," in *IEEE Symposium on Security and Privacy (SP)*, 2021. DOI: 10.1109/SP40001.2021.00020.

[44] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "Voltpillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface," in *USENIX Security*, 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai.

[45] A. Rao, *Rising to the challenge — data security with intel confidential computing*, 2022. [Online]. Available: https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141.

[46] TCG, *Trusted computing group (TCG)*, 2022. [Online]. Available: https://trustedcomputinggroup.org/.

[47] Intel, *3rd gen intel® xeon® scalable platform technology preview*, 2021. [Online]. Available: https://newsroom.intel.com/wp-content/uploads/sites/11/2021/04/3rd-Gen-Intel-Xeon-Scalable-Platform-Press-Presentation-281884.pdf.

[48] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *ICML*, vol. 97, Long Beach, California, USA: PMLR, 2019. [Online]. Available: http://proceedings.mlr.press/v97/koloskova19a.html.

[49] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *International Conference on Learning Representations*, ser. ICLR '18, 2018. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W.

[50] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *NeurIPS*, Montréal, Canada: Curran Associates Inc., 2018. [Online]. Available: https://papers.nips.cc/paper/7992-communication-compression-for-decentralized-training.pdf.

[51] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, "Personalized and private peer-to-peer machine learning," in *PMLR AISTATS*, vol. 84, Playa Blanca, Spain, 2018. [Online]. Available: http://proceedings.mlr.press/v84/bellet18a.html.

[52] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A. Kermarrec, "Privacy-preserving distributed collaborative filtering," *Computing*, vol. 98, no. 8, pp. 827–846, 2016. DOI: 10.1007/s00607-015-0451-z.

[53] G. Danner, A. Berta, I. Hegedűs, and M. Jelasity, "Robust fully distributed minibatch gradient descent with privacy preservation," *Security and Communication Networks*, vol. 2018, 2018. DOI: 10.1155/2018/6728020.

[54] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *ACM SIGSAC CCS*, Berlin, Germany, 2013, pp. 801–812. DOI: 10.1145/2508859.2516751.

[55] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Enhancing security and privacy of Tor's ecosystem by using trusted execution environments," in *USENIX NSDI*, Boston, MA, Mar. 2017. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/kim-seongmin.

[56] R. Pires, D. Goltzsche, S. Ben Mokhtar, *et al.*, "Cyclosa: Decentralizing private web search through SGX-based browser extensions," in *IEEE ICDCS*, Vienna, Austria, 2018. DOI: 10.1109/ICDCS.2018.00053.

[57] A. Law, C. Leung, R. Poddar, *et al.*, "Secure collaborative training and inference for XGBoost," in *PPMLP Workshop*, Virtual Event, USA, 2020. DOI: 10.1145/3411501.3419420.

[58] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *IEEE SP*, 2019. DOI: 10.1109/SP.2019.00045.

[59] C. Zhang, J. Xia, B. Yang, *et al.*, "Citadel: Protecting data privacy and model confidentiality for collaborative learning," in *ACM SoCC*, 2021, 546–561. DOI: 10.1145/3472883.3486998.

[60] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *ICLR*, 2019. [Online]. Available: https://openreview.net/pdf?id=rJVorjCcKQ.

[61] K. Kim, C. H. Kim, J. J. Rhee, *et al.*, "Vessels: Efficient and scalable deep learning prediction on trusted processors," in *ACM SoCC*, Virtual Event, USA, 2020. DOI: 10.1145/3419111.3421282.

[62] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "PPFL: Privacy-preserving federated learning with trusted execution environments," in *MobiSys '21*, Virtual Event, Wisconsin, 2021. DOI: 10.1145/3458864.3466628.

[63] Y. Zhang, Z. Wang, J. Cao, R. Hou, and D. Meng, "ShuffleFL: Gradient-preserving federated learning using trusted execution environment," in *18th ACM International Conference on Computing Frontiers*, New York, NY, USA, 2021. DOI: 10.1145/3457388.3458665.