FEDORA: Practical Federated Recommendation Model Learning Using ORAM with Controlled Privacy

Jinyu Liu
The Pennsylvania State University
State College, PA, USA
jzl6359@psu.edu

Wenjie Xiong
Virginia Tech
Blacksburg, VA, USA
wenjiex@vt.edu

G. Edward Suh
NVIDIA
Westford, MA, USA
Cornell University
Ithaca, NY, USA
esuh@nvidia.com

Kiwan Maeng
The Pennsylvania State University
State College, PA, USA
kvm6242@psu.edu

Abstract

Training high-quality recommendation models requires collecting sensitive user data. The popular privacy-enhancing training method, federated learning (FL), cannot be used practically due to these models' large embedding tables. This paper introduces FEDORA, a system for training recommendation models with FL. FEDORA allows each user to only download, train, and upload a small subset of the large tables based on their private data, while hiding the access pattern using oblivious memory (ORAM). FEDORA reduces the ORAM's prohibitive latency and memory overheads by (1) introducing ϵ-FDP, a formal way to balance the ORAM's privacy with performance, and (2) placing the large ORAM in a power- and cost-efficient SSD with SSD-friendly optimizations. Additionally, FEDORA is carefully designed to support (3) modern operation modes of FL. FEDORA achieves high model accuracy by using private features during training while achieving on average 5× latency and 158× SSD lifetime improvement over the baseline.

CCS Concepts: • **Security and privacy** → **Usability in security and privacy**; • **Information systems** → **Recommender systems**.

Keywords: Federated Learning; Differential Privacy; Privacy Preserving Machine Learning; Trusted Execution Environment; Oblivious Memory; Solid State Drives

ASPLOS '25, Rotterdam, Netherlands.

ACM ISBN 979-8-4007-1079-7/25/03
https://doi.org/10.1145/3676641.3716014

ACM Reference Format:
Jinyu Liu, Wenjie Xiong, G. Edward Suh, and Kiwan Maeng. 2025. FEDORA: Practical Federated Recommendation Model Learning Using ORAM with Controlled Privacy. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3676641.3716014

1 Introduction

Recommendation models are crucial in modern internet services, recommending music [76], videos [17, 105], news [123], and products [134] to people. These models are commonly trained by the service provider with sensitive *user features* (*e.g.*, demographic information or behavioral histories) [88, 135, 136], posing *privacy concerns*. Users are becoming reluctant to share these sensitive data, and governments and companies are strengthening regulations to limit the collection of such data [28, 112]. Past incidents have reported that the inability to collect these data degrades the quality of recommendation services [62]. Our evaluation (Section 6) also shows that sensitive data is crucial for high model quality.

Federated learning (FL) [77] is a popular technique to train models without collecting raw user data. Unfortunately, recommendation models cannot be trained with FL due to their large *embedding tables*, which convert discrete sparse user features into latent vectors. These tables are usually too large [58, 85, 130, 134] to train with FL because participants of FL cannot practically download and train them on their local devices (*e.g.*, smartphones) in their entirety.

In this paper, we propose FEDORA, a system that trains recommendation models with FL by allowing users to download and train only subsets of embedding tables relevant to their data. As naively doing so leaks private features via the entries downloaded [37, 87], FEDORA adopts ORAM [31], a primitive that obfuscates memory access patterns, to mitigate this leakage. The main challenges lie in overcoming

ORAM's high latency and memory overheads while still maintaining the privacy of user features.

First, we introduce *ϵ-feature-level differential privacy (ϵ-FDP)*, a variant of the popular ϵ-differential privacy (ϵ-DP) tailored for our privacy goal. When the strongest privacy is not necessary, ϵ-FDP enables FEDORA to trade off small privacy in a controlled manner to gain significant improvement in its ORAM. Any privacy degradation can be interpreted through its connection with the well-studied DP framework.

FEDORA effectively controls ϵ in ϵ-FDP to balance privacy, performance, and accuracy while supporting many popular operation modes of modern FL. To mitigate the prohibitive memory overheads of large embedding tables, FEDORA hosts the large ORAM on a power and cost efficient SSD with several SSD-friendly optimizations. We implement our prototype of FEDORA in software, assuming a trusted execution environment (TEE) with a small (4 KB) on-chip scratchpad. Below summarizes our contribution:

- To the best of our knowledge, FEDORA is the first to adopt ORAM to realize recommendation model training with FL. We show in our evaluation that FEDORA's ability to leverage sensitive user features in a privacy-preserving manner can significantly improve the final model accuracy. Through careful design, FEDORA supports many common operation modes in FL.
- We introduce a formal notion of privacy in federated recommendation training, ϵ-FDP. We show how an ORAM-based FL system can be designed to achieve ϵ-FDP while arbitrarily trading off performance, privacy, and accuracy. We show that existing heuristic ORAM optimizations are special cases of our general design.
- FEDORA introduces system optimizations that allow the large ORAM to be held in a cheap SSD while maintaining reasonable latency and SSD lifetime. Our evaluation shows that FEDORA realizes up to 1.8–24× latency improvement and 21–1000+× SSD lifetime improvement over other SSD baselines, and 6–22× and 1.9–23× hardware cost and energy improvement over a DRAM-based alternative.

2 Background and Motivation

2.1 Deep Learning Recommendation Models

Modern recommendation models [16, 33, 49, 85, 92, 110, 120, 121, 127, 130, 135, 136] use user features to predict items a user might like. When the feature values are categorical or sparse (*e.g.*, recently purchased items are not a vector-valued input), they are first translated into a dense vector through an *embedding table*. Each entry (row) of an embedding table holds a vector (64–256 bytes [58, 85]) that is a learned latent representation of the corresponding feature value. Translating with an embedding table is a lookup operation, where the *accessed row index directly leaks the feature value*. The translated vectors, along with dense features (naturally vector-valued inputs), go through a multi-layer perceptron (MLP) [85] or a Transformer-like [130] network to produce a prediction. The cardinality of the sparse feature's domain determines the height of an embedding table, and the table can be very large (*e.g.*, equal to the number of items on an e-commerce site that a user can purchase). Production-scale models often use tables with millions or billions of entries [70, 130, 134]. Each user only accesses a small subset of the table (*e.g.*, entries corresponding to a few items the user recently purchased), resulting in an extremely sparse memory access pattern.

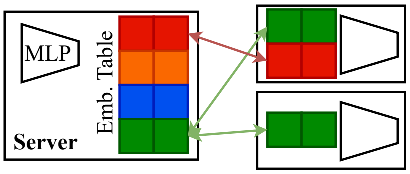

Figure 1. An insecure FL of a recommendation model, where users only partially download/train the embedding tables.

Sensitive user features. Many features used by modern recommendation models are sensitive and can cause a privacy issue if their *values* are leaked [37, 90, 115]. For example, modern models may use a user's demographic information, geographic location, or past behavioral histories (*e.g.*, items recently purchased or websites recently visited) [88, 135, 136]. The *number of feature values* can also be sensitive information. For example, the number of items a user recently purchased can reveal whether the user is a heavy shopper.

2.2 Federated Learning (FL)

Federated learning (FL) [77] is a popular method for training models without collecting raw user data. It has been adopted by major companies like Google [124, 133], Meta [34, 41], and LinkedIn [117].

In each round t of FL, the server selects a subset of users C to participate. Each selected user $c \in C$ downloads the current global model θ_t from the server and trains the model on their local devices with their private data. After training, users send their trained model (or, equivalently, the gradient[1]) back to the server. The server aggregates the received gradient ($\Delta\theta_t^c$) from each user c, and uses the aggregated gradient to update its global model. This process repeats for a predetermined number of rounds or until the model converges. The popular FedAvg [77] performs a weighted averaging of the gradients using the number of samples of each user (n_t^c):

$$\theta_{t+1} = \theta_t - \eta \Sigma_c \frac{n_t^c}{n_t} \Delta\theta_t^c, \tag{1}$$

where η is the learning rate and $n_t = \Sigma_c n_t^c$. FL is not entirely safe by itself [6, 7, 50, 137] and is often used with

[1] Here, gradient refers to the delta between the model before and after the local training. It is not necessarily the per-batch gradients in SGD.

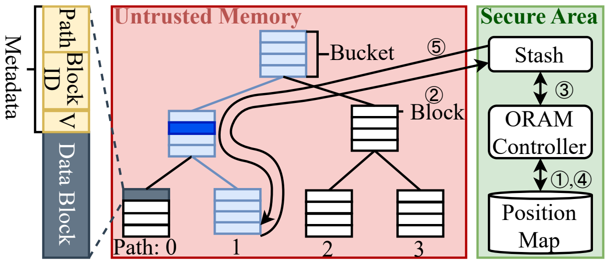

Figure 2. Path ORAM accessing the highlighted block.

secure aggregation (SecAgg) [8, 41, 114] and differential privacy (DP) [78, 124, 133]. SecAgg hides individual user gradients (n_t^c) from the service provider and only reveals the aggregated gradients. DP provides a mathematical guarantee on the information leakage through the aggregated gradient [78]. FEDORA is compatible with both SecAgg and DP.

FL cannot protect user features. The large embedding tables prohibit recommendation models from adopting FL. Prior work [87] suggested training the rest of the model with conventional FL while making users download, train, and upload *only subsets* of the embedding tables related to their data (Figure 1). When the rest of the model (MLP or Trasnformer-like *dense* layers) is relatively small [38, 85, 135, 136], this idea enables practical FL.

However, doing so is *not private* because the downloaded entries directly leak the user's feature values (Section 2.1). While FL commonly runs inside tamper-proof secure hardware on the server—called trusted execution environment (*TEE*; Section 5.1)—the accessed entries can still be discovered through various side channels [37]. A common workaround is to refrain from using any sensitive features during training [55, 58] and only use features the users agreed to share. However, not leveraging private features severely limits the model quality [58] as we will show in Section 6.

2.3 Oblivious RAM (ORAM)

ORAM [30, 31] is a security primitive that randomizes the memory access pattern and encrypts data values. Many popular ORAMs are variants of Path ORAM [108] (Figure 2). In Path ORAM, data are stored in fixed-size *blocks* in a binary tree (ORAM tree) where each node, called a *bucket*, has a fixed number of slots that can hold blocks. Each block is assigned to a particular *path* in the tree, and the *position map* holds the mapping between the blocks and the paths. The invariant is that blocks are always either in a bucket along the assigned path or in the *stash*, a fixed-size buffer that can hold a certain number of blocks [97]. Each slot also holds metadata, including a valid flag that indicates whether the slot contains a block. Typically, the stash and the position map are placed in an on-chip storage (*e.g.*, SRAM) that is hard to attack, while the ORAM tree is placed off-chip. On-chip storage is expensive [109] and small (usually a few kilobytes [26, 96]). If the position map is too large, it can also be stored off-chip in separate recursive ORAMs [108].

Figure 2 summarizes the operation of Path ORAM. When a request to a block is made, the ORAM controller (1) reads the position map to find on which path the block resides. (2) The entire path is brought to the stash and decrypted, and (3) the requested block is served. (4) Then, the block is randomly re-assigned to a new path, and the position map is updated. (5) Finally, to keep the stash size limited, as many blocks as possible are re-encrypted and evicted back to the same path that was read while maintaining the invariant. To an external observer, every ORAM access looks like reading and writing a random path.

3 Feature-level Differential Privacy (FDP)

During FL, we must prevent an adversary (*e.g.*, the FL server owner) from inferring each user's feature values (*e.g.*, recently purchased item). We introduce a variant of differential privacy (DP), *feature-level differential privacy (FDP)*, to achieve this privacy goal. We also explain how FDP can be extended to hide multiple values at the same time or hide the number of feature values (*e.g.*, the number of recently purchased items). Then, we explain why existing ORAMs cannot achieve FDP without significant overheads.

3.1 Definition of ϵ-FDP

DP is the de facto standard in quantifying ML privacy [124, 133]. We first reiterate the popular ϵ-DP [22] below:

Definition 3.1 (ϵ-DP)**.** Let $\mathcal{M}$ be a (randomized) algorithm and $\mathcal{D}$ be the input space of $\mathcal{M}$. For all subsets $\mathcal{S}$ of the output of $\mathcal{M}$ and all neighboring inputs $d, d' \in \mathcal{D}$, $\mathcal{M}$ is said to be ϵ-DP if:

$$\mathbf{Pr}[\mathcal{M}(d) \in S] \leq e^{\epsilon}\mathbf{Pr}[\mathcal{M}(d') \in S]. \tag{2}$$

ϵ-DP ensures that the output $\mathcal{S}$ only gives bounded information on whether the secret input was d or d'. When ϵ is small, the output distribution from d and d' becomes similar, making it hard for the adversary to guess the input (high privacy). When ϵ is zero, d and d' produces the exact same output distribution (perfect privacy).

The definition of *neighboring inputs* is application-specific, and choosing what is considered as neighboring inputs results in different privacy protections. DP-SGD [1] considers two datasets differing in one training sample as neighboring, which hides individual samples in the training set (item-level DP). DP-FedAvg [78] considers two datasets differing in one user as neighboring, entirely hiding the existence of each user during training (user-level DP). Our privacy goal is to hide each feature value of a user, which is different from both. Item-level DP cannot effectively protect feature values when users have several training samples with the same feature value (*e.g.*, demographic information will be the same for all the training samples from the same user). User-level DP can be too strong because it hides the existence of an entire user [4]. Instead, we define a new *feature-level differential privacy (FDP)* as a middle ground:

Definition 3.2 (ϵ-FDP)**.** If a system achieves ϵ-DP, where an input d is an arbitrary group of user feature values, and a neighboring input d' is defined by replacing any single feature value from d into an arbitrary different value in the feature value space, we say the system achieves ϵ-FDP.

Interpretation of ϵ-FDP. In an ϵ-FDP system with a low ϵ, replacing a feature value with a different value minimally influences the output (bounded by e^{ϵ}), making it hard for an adversary to guess a feature value of a user by looking at the (observable) output. Prior works on DP showed that ϵ can bound the *success rate of an adversary* [29, 48, 52, 84], which directly extends to ϵ-FDP. While the interpretation of ϵ varies between contexts, many works on DP assume $\epsilon \leq 10$ to be reasonably safe and $\epsilon \leq 1$ to be strongly safe [89]. We show in Section 6 that FEDORA works well in these regimes.

ϵ-FDP can be extended to hide multiple (correlated) values or the total number of feature values of a user. If we make all the users' number of feature values to be n (through thresholding and padding with dummy values) and hide the n values simultaneously, we can also hide the *number of feature values* of a user—as the attacker cannot distinguish between n real values and n dummy values (no real value). According to group privacy of DP [22], hiding n values simultaneously increases the privacy parameter by a factor of n. That is, hiding n features while achieving ϵ-FDP requires adding an equivalent noise of hiding each value with $\frac{\epsilon}{n}$-FDP.

3.2 Limitations of ORAM in Achieving ϵ-FDP

ORAM [26, 31, 108] can hide the embedding table access pattern and was used for recommendation model privacy in different contexts [90]. We show that applying vanilla ORAM achieves ϵ-FDP with $\epsilon = 0$ (perfect FDP) but severely degrades the performance, and a naive optimization similar to [25, 128, 132] results in $\epsilon = \infty$ (no FDP).

Strawman 1: Vanilla ORAM is private but inefficient. When ORAM is applied to embedding tables, access to a particular entry cannot be distinguished from access to any other entries. By definition, this achieves FDP with $\epsilon = 0$, the strongest privacy. However, vanilla ORAM incurs significant performance and memory overheads. ORAM amplifies each memory access into $O(logN)$ accesses with N entries. Storing N entries inside ORAM also amplifies the memory footprint (by 1.5–2× [69, 94, 96, 131] to 6–8× [96, 108]) because ORAM moves data blocks inside the tree and cannot work when the tree is full. In recommendation, N can go up to tens of millions to billions [58, 75, 134], making both the latency and memory amplification significant.

Strawman 2: Naive optimization is not private. Instead of accessing the ORAM on every user request, one might come up with an optimization where requests to the same entry are aggregated inside the ORAM controller, and only one read/write request per unique entry is sent to the ORAM to serve all duplicate requests. This optimization is similar to prior ORAMs that aggregate requests inside cache [25, 128] or stash [132]. Unfortunately, such a heuristic optimization does not achieve FDP. In such optimizations, the *total number of accesses* varies with feature values, which leaks information to the attacker (*e.g.*, FL server owner). Consider an extreme case where all the users access the exact same entry, resulting in only one ORAM read/write. The adversary will know that all the users have the same feature values, a valuable piece of information that would not have been leaked with Strawman 1. It can be seen that changing one feature within d to a different feature (d') in this extreme case will increase the number of ORAM accesses from one to two—changing the output distribution unboundedly ($\epsilon = \infty$).

3.3 Designing a Controllable ϵ-FDP ORAM

We introduce a generalized method to design an ORAM that can achieve ϵ-FDP with an arbitrary ϵ. We will subsequently show that Strawman 1 and 2 from Section 3.2 are special cases of our generalized method. Our method will be used to balance privacy, performance, and accuracy when anything other than the extreme designs (Strawman 1, 2) is desired.

Let C be the subset of users selected to participate in the FL round. Each user $c \in C$ has a set of embedding entries E_c corresponding to their private data that they must download and update to complete the training round. Let $K = \sum_{c \in C} |E_c|$ be the total number of entries required by the selected users and $k_{union} = |\bigcup_{c \in C} E_c|$ be the number of unique entries within K. Note that K is not necessarily a secret, while k_{union} is. Our goal is to send $k \leq K$ requests to the ORAM while ensuring that the number of ORAM requests (k) only gives bounded information about the feature values. This can be done by choosing k following the probability distribution:

$$p_i = \frac{Y_i e^{-\epsilon \frac{|k_{union} - i|}{2}}}{\sum_{j=1}^{K} Y_j e^{-\epsilon \frac{|k_{union} - j|}{2}}}, \qquad 1 \leq i \leq K \tag{3}$$

where p_i serves as the probability density function (PDF) of $k = i$ being selected. Y_is are predefined parameters that shape the PDF to balance performance and accuracy.

Proof. The proof directly follows that of the exponential mechanism (see [22] for its general form and [42] for the variant using Y_i). We briefly restate the proof below for completeness. From Equation 2, it suffices to show that $\frac{\mathbf{Pr}[\mathcal{M}(d) \in S]}{\mathbf{Pr}[\mathcal{M}(d') \in S]} \leq e^{\epsilon}$ for arbitrary d and d'. When the observable output S is the number of ORAM requests, the probability of an input d to result in i ORAM requests is directly p_i from Equation 3, *i.e.*, $\mathbf{Pr}[\mathcal{M}(d) = i] = \frac{Y_i e^{-\epsilon \frac{|k_{union} - i|}{2}}}{\sum_{j=1}^{K} Y_j e^{-\epsilon \frac{|k_{union} - j|}{2}}}$. If d results in k_{union} and d' results in k'_{union}, $|k_{union} - k'_{union}| \leq 1$ as changing one feature value in the input can change the union size at most 1. We can show that for all i,

$$\begin{aligned}\frac{\mathbf{Pr}[\mathcal{M}(d)=i]}{\mathbf{Pr}[\mathcal{M}(d')=i]} &= \frac{Y_i e^{-\epsilon\frac{|k_{union}-i|}{2}}}{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k_{union}-j|}{2}}} \cdot \frac{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k'_{union}-j|}{2}}}{Y_i e^{-\epsilon\frac{|k'_{union}-i|}{2}}} \\ &= e^{\frac{\epsilon(|k'_{union}-i|-|k_{union}-i|)}{2}} \cdot (\frac{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k'_{union}-j|}{2}}}{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k_{union}-j|}{2}}}) \\ &\leq e^{\frac{\epsilon}{2}} \cdot (\frac{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k'_{union}-j|}{2}}}{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k_{union}-j|}{2}}}) \\ &\leq e^{\frac{\epsilon}{2}} \cdot e^{\frac{\epsilon}{2}} \cdot (\frac{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k_{union}-j|}{2}}}{\Sigma_{j=1}^{K} Y_j e^{-\epsilon\frac{|k_{union}-j|}{2}}}) = e^{\epsilon}.\end{aligned}$$

□

Figure 3 shows some sample PDFs with different ϵ and Y_i, showing how the proposed method can trade-off between performance, privacy, and accuracy. The Y_is used are (a, c, e) uniform ($Y_i = 1$), (b) square ($Y_i = 1$ for $25 \leq i \leq 100$, else $Y_i = 0$), (d) pow ($Y_i = i^5$), and (f) delta ($Y_i = 1$ for $i = 100$, else $Y_i = 0$). Without considering security, for the best performance and accuracy, the ORAM should be accessed exactly $k_{union} = 30$ times (bars in blue). If the system accesses the ORAM less ($k < k_{union}$), some necessary entries cannot be read (lower accuracy; bars in orange). If accessed more ($k > k_{union}$), dummy accesses are being issued (lower performance; bars in green). We highlight several interesting observations:

- **Observation 1: ϵ-FDP improves performance.** Compared to the vanilla ORAM (Strawman 1), which will always read the main ORAM $K = 100$ times, Figures 3 (a–e) reads the ORAM much less, improving performance.
- **Observation 2: ϵ trades off privacy with accuracy and performance.** Figures 3 (a, c, e) shows that reducing ϵ (more privacy) increases the chances of incorrect (orange) and less efficient (green) executions.
- **Observation 3: Y_i trades off performance with accuracy.** Figures 3 (b, d, f) shows that choosing a nonuniform Y_i allows trading the chances of being inaccurate (orange) with being inefficient (green), whose effect is shown with the unbalanced orange and green regions.
- **Observation 4: Strawman 1/2 are special cases.** When an extreme Y_i is used to entirely eliminate the chance of being incorrect by always being inefficient, the system degenerates to $k = K$ (Strawman 1; Figure 3 (f)). In this case, the value of ϵ does not matter and can go down to 0 (perfect FDP). When very high ϵ is used (no FDP), the system degenerates to $k = k_{union}$ (Strawman 2; Figure 3 (a)).

4 FEDORA System Design

FEDORA is an FL system for recommendation models that uses ORAM to allow partial embedding table updates. It (1) adopts the ϵ-FDP ORAM from Section 3.3 to trade off performance, privacy, and accuracy, (2) supports popular operation modes in FL, and (3) places the large ORAM in a cheap SSD for reduced power and cost.

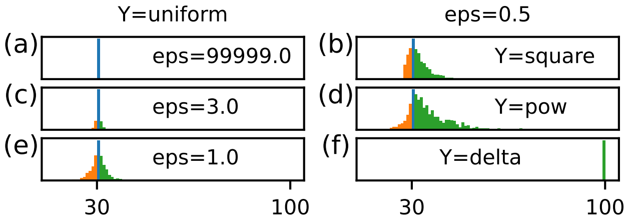

Figure 3. Sample PDFs with different ϵ and Y_i, with $k_{union} = 30$ and $K = 100$. Different Y_is explained in the main text.

4.1 Assumptions and Threat Model of This Work

This work assumes *on-chip* data in the FEDORA controller is safe, while the attacker can observe both the values and the access pattern (address, size, and timing) for data stored *off-chip* (DRAM or SSD). The security of the communication between the FEDORA controller and the users and the attestation of the FEDORA controller's integrity to the user is outside the scope of this work. We use constant-time logic inside the FEDORA controller to prevent timing side channels. We do not protect the FEDORA controller against other (*e.g.*, thermal) side channels. Such an FEDORA controller can be implemented in TEEs [43] or as custom hardware [108].

This work focuses on protecting user data against an honest-but-curious service provider that follows the FL protocol while trying to steal user data. The work does not consider malicious users colluding with the service provider (*i.e.*, Sybil attack [20]). The threat model is common and aligns with many prior FL literature [8, 98]. In general, FL cannot protect effectively against Sybil attacks [6], which is a known limitation of FL that FEDORA inherits. Defending against Sybil attacks in FL is an open problem orthogonal to this paper [27, 46, 47, 100]. We assume a predetermined minimum and a maximum number of clients participating in each round of FL [35, 41, 126, 133], and each client has a predetermined maximum number of features.

4.2 FEDORA Overview

Figure 4 explains how FEDORA works. FEDORA has two ORAMs, the large *main ORAM*, which holds the embedding table, and a smaller *buffer ORAM*, which holds the working set of entries of each round. Each embedding table entry (64–256 bytes) becomes a block in the main ORAM. FEDORA places the main ORAM inside an SSD, while the buffer ORAM resides in DRAM. Buffer ORAM serves two purposes: it provides support for a wide range of modern FL operation modes and serves as a DRAM cache for the main ORAM. FEDORA applies ϵ-FDP to the main ORAM.

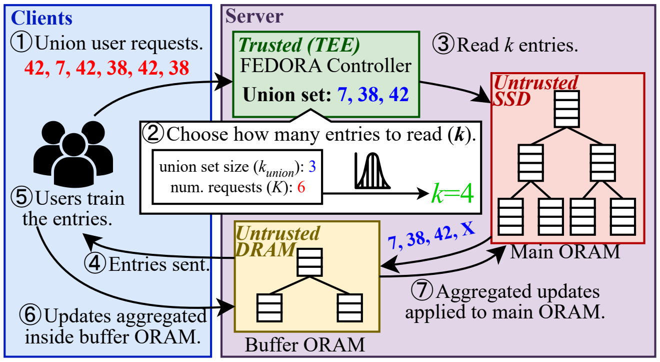

Figure 4. Overview of FEDORA.

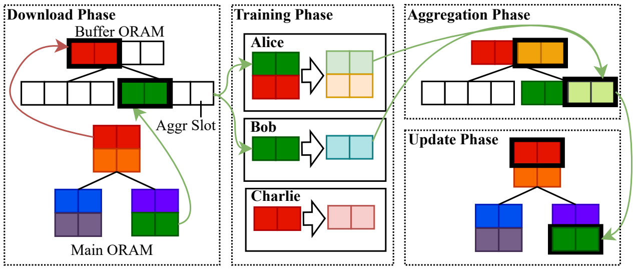

Figure 5. Downloaded entries are moved to the buffer ORAM (left, step ③). Users' gradients are aggregated in the aggregation slot inside the buffer ORAM (top right, step ⑥) and used to update the main ORAM (bottom, step ⑦).

Each FL round starts by selecting a subset of users to participate. ① The selected users send download requests to the FEDORA controller for embedding table entries that correspond to their feature values. The FEDORA controller securely calculates the union of requested entries across all participating users, and ② chooses the number of entries to read from the main ORAM (k) by sampling from the PDF of Equation 3 inside its secure controller. Then, ③ the FEDORA controller moves k entries from the main ORAM into the buffer ORAM. ④ Requested entries are served to the users by the FEDORA controller from the buffer ORAM, and ⑤ users train using the downloaded entries. ⑥ After training, users send the gradients of the embedding entries back to the FEDORA controller, which aggregates them inside the buffer ORAM. ⑦ Finally, the FEDORA controller moves k entries from the buffer ORAM back to the main ORAM, using the aggregated gradients to update them in the process.

At step ①, FEDORA controller goes through K user requests and calculates the union set of k_{union} elements. This is done without leaking any information by allocating an array to hold the resulting union set and simply linear scanning through both data structures (list of requests and result array) in a data-oblivious manner. The array is conservatively sized to make overflowing impossible. The union algorithm is $O(K^2)$ and can result in significant overheads when K is large. When K is too large, FEDORA splits the requests into evenly-sized *chunks* and performs steps ①–③ chunk by chunk to reduce the linear scanning overhead. This still achieves the same ϵ-FDP thanks to the parallel composition property of DP [22]. However, using smaller chunks can degrade accuracy as ϵ-FDP adds noise per chunk, and the noise accumulates with more chunks. It also degrades performance when there are duplicate entries across chunks. The chunk size is selected empirically (16K entries in our evaluation) to balance between performance and accuracy.

When reading from the main ORAM (steps ②–③), dummy accesses may be made (when $k > k_{union}$, X in the figure), or some necessary entries may fail to be read (when $k < k_{union}$). In the latter case, training accuracy can degrade. Several mitigation strategies can minimize the impacts of such errors. First, FEDORA has the liberty to choose which k entries to read. Some strategies include choosing the first k entries, choosing randomly, prioritizing popular entries or previously unseen entries, *etc.* Second, different strategies can be used for lost entries. Strategies include using a random/default value or simply dropping the corresponding training sample. Our prototype chooses the first options for both, which were simple and empirically worked well (Section 6.4).

4.3 Buffer ORAM for Different Operation Modes

Figure 5 summarizes how buffer ORAM works (steps ③–⑦). Blocks in the buffer ORAM are twice as large as the main ORAM blocks. Each entry read from the main ORAM is placed in the *first half* of the buffer ORAM blocks (Figure 5 (left)). Users' requests are served from the buffer ORAM (Figure 5 (middle)), and users' gradients are aggregated in the *second half* of the corresponding buffer ORAM blocks (Figure 5 (top right)). The aggregated gradients are used to update the original entry (first half) when the entry moves back to the main ORAM (Figure 5 (bottom right)).

The aggregation phase (Figure 5 (top right)) and the update phase (Figure 5 (bottom right)) can be customized to support a wide range of common operation modes in FL. FEDORA exposes a programmable pre-aggregation function (Pre) applied to each gradient before aggregation and a post-aggregation function (Post) applied to the aggregated gradient right before update. This allows system designers to implement a generalized version of Equation 1:

$$\theta_{t+1} = \theta_t - \eta\,\mathrm{Post}(\Sigma_c\,\mathrm{Pre}(\Delta\theta_t^c)). \tag{4}$$

The most popular FedAvg (Equation 1) can be achieved by choosing $\mathrm{Pre}(\Delta\theta_t^c) = n_t^c \Delta\theta_t^c$ and $\mathrm{Post}(x) = \frac{x}{n_t}$, which can be implemented by adding one additional 4-byte slot for each block to accumulate $n_t = \Sigma_c n_t^c$. This implementation naturally supports users dropping out during training by dynamically adjusting n_t. More complex algorithms, such as FedAdam [95], can be implemented by choosing $\mathrm{Post}(x)$ that involves the first- and second-order moments [54], with additional slots in each block to accumulate them. Implementing a differentially private FL (*e.g.*, DP-FedAvg) [78] is also possible[2]. A DP training method for recommendation models, EANA [86], can be adopted for FL by choosing $\mathrm{Pre}(x) = x/\max(1, \frac{||x||_2}{C})$ and $\mathrm{Post}(x) = x + \mathcal{N}(0, \sigma^2 C^2 I)$ for constants σ and C [1, 86]. Another recent method, LazyDP [68], can be implemented for FL by using $\mathrm{Post}(x) = x + \mathcal{N}(0, r\sigma^2 C^2 I)$, where r is the number of rounds since the entry was last updated. r can be tracked with an additional per-block counter.

Buffer ORAM privacy analysis. Introducing buffer ORAM does not degrade privacy. To the adversary, each round is simply seen as reading k entries from the main ORAM and writing them to the buffer ORAM (③), reading K entries from the buffer ORAM (④), updating K entries in the buffer ORAM (⑥), reading k entries to the buffer ORAM and writing them to the main ORAM (⑦). k and K are publicly known parameters, and both ORAMs protect which entries have been accessed. The buffer ORAM's capacity is sized to never overflow based on the maximum number of clients allowed per round and the maximum number of features per client. The capacity can be reconfigured in software when these parameters change.

4.4 Placing the Main ORAM in an SSD

The main ORAM must be 1.5–8× larger than the data it protects (Section 3.2). When the tables are already several gigabytes/terabytes [58, 85, 130, 134], placing the main ORAM in DRAM involves significant power, energy, and hardware cost. The costs are wasteful because user-side training and communication, not the server-side, are usually the main bottleneck in real-world FL [35, 57, 117, 126], leaving the large DRAM significantly underutilized. Instead, FEDORA places the main ORAM in a cheap, off-the-shelf SSD.

Limitations of existing SSD-based ORAMs. Prior SSD-based ORAMs [5, 11, 14, 69, 101, 106, 107, 113, 129] are not suitable for FL because the frequent reads/writes of FL quickly wear out the SSDs. As our evaluation shows (Section 6.2), using an existing SSD-friendly ORAM [101] based on Path ORAM for FL can quickly wear out the SSD only in 2–8 days, which is unreasonably short compared to the typical device lifetime in modern data centers (traditionally around 2–3 years, recently up to 5–6 years for lower carbon footprint [71]). The slow SSD reads/writes also add up to over 8× slowdown to the end-to-end FL latency (Section 6.3).

FEDORA's SSD-friendly ORAM design. Prior ORAMs based on Path ORAM [101] cannot leverage the unique data access pattern of FL where the first half (step ③) of the accesses are read-only and the second half (step ⑦) are write-only. Instead, FEDORA utilizes a custom variant of RAW ORAM [26] to benefit from FL's unique access patterns.

RAW ORAM [26] consists of access-only (AO) and eviction-only (EO) accesses. An AO access occurs on every memory read/write, which reads the entire path and places it in a *path buffer*. The path buffer is iterated through to find the requested block, and only the requested block moves to the stash. As the stash is not as populated after one AO access, RAW ORAM does not immediately write blocks back (unlike Path ORAM). Instead, only the valid flag of the metadata is updated to indicate that the block is pulled out. After A AO accesses (A, the *eviction period*, is a design parameter), an EO access occurs, which selects a path in a predetermined order and evicts blocks from the stash to the selected path.

Optimization 1: FL-friendly RAW ORAM. Adopting RAW ORAM immediately reduces the number of SSD writes because EO accesses (path read + write) only happen after A AO accesses (path read). On top of this benefit, FEDORA develops several additional optimizations. During step ③, the main ORAM is read-only. In the original RAW ORAM, even a series of read-only accesses need to be interspersed with EO accesses because blocks accumulate in the stash and can eventually cause overflow. However, stash overflow does not happen in FEDORA because all the blocks read from the main ORAM immediately move to the buffer ORAM, making the main ORAM's stash always empty. Leveraging the fact, FEDORA entirely eliminates EO accesses during step ③.

Similarly, AO accesses can be eliminated during step ⑦ as it is write-only. In normal RAW ORAM, updating a block incurs (1) an AO access, which brings in the target block to the stash, (2) the modification of the block inside the stash, and (3) an eventual EO access after A AO accesses, which puts the block back. In FEDORA, the block to update in step ⑦ is never in the main ORAM and is always inside the buffer ORAM. Thus, AO accesses can be removed, and only EO accesses occur after every A block update moving from the buffer ORAM to the main ORAM's stash.

Optimization 2: Making AO access SSD-write-free. While AO accesses do not write to the path, they still update the valid flag of the main ORAM's metadata and cause SSD writes. FEDORA introduces a new data structure called *VTree* to make AO accesses entirely SSD-write-free. VTree is a small ORAM placed in DRAM, which only holds the valid flags extracted from the main ORAM. VTree is not independent, and blocks in VTree mirror the movements of the corresponding blocks in the main ORAM. VTree adds one bit per data block (64–256 bytes) plus additional metadata for secure encryption (Section 5.2), totaling around 2–112 MB.

Optimization 3: Minimizing EO accesses. With VTree, only EO accesses write to the SSD. As EO accesses happen only after A AO accesses, FEDORA maximizes A to minimize SSD writes. However, increasing A increases the bucket size, stash size, and path buffer size [96], degrading the latency.

[2] This DP is different from DP in ϵ-FDP. DP-FedAvg prevents training data leaking through the trained model [78, 124, 133], while ϵ-FDP prevents feature values leaking through the number of accesses to ORAM (k).

Also, more on-chip SRAM is needed if these are placed on-chip [26, 108]. For these issues, the original RAW ORAM used a small A (*e.g.*, 5 [26]).

FEDORA places the stash and the path buffer in off-chip DRAM, allowing the use of a fairly large stash and path buffer without needing a large on-chip SRAM. Placing these structures off-chip typically increases latency, but we observed that the decreased number of SSD writes cancels the additional overheads for FEDORA. Also, FEDORA can use a large bucket size without much downside because the SSD, which is a block device, can only access data in a large granularity (*e.g.*, 4 KB) anyways. FEDORA balances A, the bucket size, stash size, and path buffer size to maximize the SSD lifetime while not significantly adding latency. Through careful tuning, FEDORA was able to increase A significantly (up to 92), cutting down the number of EO accesses to 1.1%.

Privacy analysis. Our optimizations do not degrade the security properties of RAW ORAM [26] because it can be seen as simply moving $\frac{k}{A}$ EO accesses that should have occurred in between k AO accesses at step ③ to step ⑦. If we consider the state of the main ORAM at the end of each round, the stash occupancy will be equivalent to what it would have been when using the vanilla RAW ORAM; thus, the same proofs from [96] for stash overflow can be used. Within each round, the stash never overflows due to the buffer ORAM.

5 Implementation Details

We implement our prototype in software, assuming running in a trusted execution environment (TEE) with a small (4 KB) on-chip SRAM (scratchpad). TEEs are already widely available in FL systems [41, 117, 133], which makes building FEDORA in them appealing. Prior works [67, 81, 101] also implemented ORAM in a TEE, and our implementation is similar to them. It is possible to implement FEDORA as specialized hardware instead [26, 96, 108].

5.1 TEE-based Implementation Overview

Background on TEEs. A TEE [2, 43, 59, 60, 109] is a specialized hardware that provides data confidentiality and integrity verification. TEEs encrypt data stored off-chip and only decrypt data inside its secure hardware. Academic proposals [109] and earlier versions of Intel SGX [43] used strong counter-based memory encryption, where an incrementing counter and an authentication tag were allocated for every cache line for freshness and tamper detection [32]. Additionally, tampering with counters was detected through an expensive Merkle tree [32, 109]. Other TEEs [44] do not provide such strong security and are prone to certain attacks [64, 65]. While TEEs can provide data confidentiality when used for FL [41] for the rest of the model, they cannot hide the memory access patterns [37] for embedding tables.

Commercial TEEs do not offer on-chip scratchpads that are safe from external attacks, and any programmer-controlled data must be placed in insecure off-chip DRAM. The lack of a scratchpad makes ORAM implementation inefficient [67, 81, 101] on them because typical ORAMs assume a safe on-chip scratchpad [26, 96, 108].

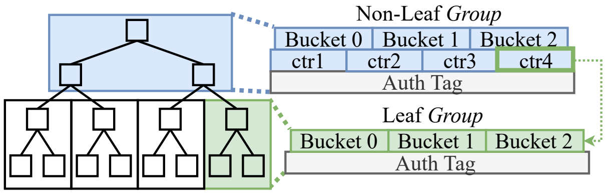

Figure 6. Group-based encryption for tree-structured data.

TEE-based FEDORA. Our FEDORA prototype is built in software, assuming it runs in TEEs that FL systems commonly have [41, 117, 133]. Specifically, our implementation assumes a TEE with a strong counter-based memory encryption support [43, 109] and a small amount (4 KB) of scratchpad. We assume a small scratchpad (although existing commercial products do not have it [43, 44]) because it makes the entire system a lot more efficient and is feasible to add—many embedded CPUs [3] and hardware [53, 83] have scratchpads, and on-chip cache can be repurposed as a scratchpad through cache locking [82] or specialized features [12, 19, 40]. FEDORA can also be implemented without the small scratchpad with additional overheads (Section 6.6).

As we use minimal (to no) scratchpad, we place most of the data structures off-chip. The on-chip scratchpad holds the key and the root counter for encryption (Section 5.2), and a small scratch space for efficient EO access implementation. FEDORA places all the other components other than the main ORAM in off-chip DRAM and the main ORAM in the SSD. Following [101], we implemented FEDORA in C++ with a best-effort constant-time, data-independent logic to avoid timing side channels.

5.2 Encrypting Off-chip Data Structures

Existing TEEs' [43, 109] hardware engines for counter-based encryption [32] are efficient for small data structures but incur high computation/memory overheads for larger structures. FEDORA uses existing hardware for small data structures but uses a tailored algorithm for larger structures (position map, buffer ORAM, VTree, and the main ORAM).

When encrypting these tree structures in off-chip DRAM, FEDORA groups multiple nodes and encrypts/decrypts them together (Figure 6). Nodes in the same group share the counter and the tag, and the group size balances the counter/tag overhead and the encryption/decryption latency. We empirically group 512 bytes of nodes together, which gives an analytical 8× memory overhead improvement over a TEE that allocates the counter/tag per cache line (64 bytes) [32]. FEDORA groups nodes in a subtree together (Figure 6) to minimize the number of encrypted groups in a path.

FEDORA protects counters without Merkle tree by placing the counter for each group in its parent group (Figure 6 (right)). The counter for the root group (root counter) is stored in the scratchpad. Decrypting a path starts from the root: each group is decrypted, the integrity of the values and the counter are verified, and the counter is used to decrypt the next group. Encryption happens in the opposite direction. The main ORAM is encrypted without any counters other than the root counter, using a similar idea to [26]. Using the fact that writes to the main ORAM only happen during EO accesses in a predetermined order (Section 4.4), FEDORA tracks the number of writes to each bucket with a single root counter that tracks the total number of EO accesses [26].

6 Evaluation

We aim to answer the following questions on FEDORA:

- How much longer is the SSD lifetime?
- How much faster is FEDORA?
- How much accuracy is improved with private features?
- How much cost benefit does using an SSD bring over a DRAM-based system?

6.1 Evaluation setup

System setup. We evaluated FEDORA on Intel i7-13700K CPU with Samsung PM9A1 1 TB SSD. We did not use a commercial TEE because they do not have any on-chip scratchpad. Our result can be interpreted as a close estimate of a setup with a TEE with a scratchpad as TEE-related overheads are expected to be small compared to SSD overheads.

Datasets. We studied three popular datasets, MovieLens-20M [36] (MovieLens), Taobao Ads Click and Display [88] (Taobao), and Criteo Kaggle [18] (Kaggle). MovieLens and Taobao reveal the user ID of each datapoint and allow FL simulation [74]. We used them both for performance and FL accuracy evaluation. Kaggle does not reveal user ID and cannot simulate data heterogeneity between users in FL. We used Kaggle only for performance evaluation.

For MovieLens and Taobao, we considered the behavioral history (recently liked movies for MovieLens, recently purchased items for Taobao) private. We studied two modes of protection: (1) only hiding individual feature values (*hide priv val*) and (2) hiding the number of feature values (*hide # of priv vals*). For the latter, we made every user have 100 real or dummy values through padding or random subsampling. For Kaggle, we assumed the largest table as private and only evaluated the former mode.

Performance study. The open-source datasets are much smaller-scale than production environments [58]. To study scenarios closer to production, we scaled up these datasets with a synthetic dataset generation technique [85], scaling up the table size, number of users, and number of feature values per user. We evaluated three table sizes. The *Small* table has 10 million entries, 64 bytes each. The *Medium* table has 50 million entries, 128 bytes each. The *Large* table has 250 million entries, 256 bytes each. The sizes follow open-source models and industry papers [58, 75, 85, 134, 136].

We scaled up the number of requests (K) to 10K, 100K, and 1M. These numbers represent the number of users per round multiplied by the number of feature values of each user. The number of users per round ranges from 100 [35, 126] to 6,500 [124], and the number of feature values per user is usually several tens [58], but can go up to hundreds [135, 136]. The studied K values represent a wide range of real-world use cases; for example, 10K requests can represent 100 users per round (similar to [35, 126]) with each having 100 feature values (similar to [135, 136]), or 1,000 users per round (similar to [41]) with each having 10 values (similar to [58]).

For end-to-end FL latency, we assumed the communication through the network and the user-side training altogether take roughly 2 minutes per round, following the reported numbers from Google [126]. While the setups significantly differ from each other, other real-world studies reported similar numbers [35, 41, 57, 117, 125]. When estimating SSD lifetime, we assumed 5.4 PB can be written per TB capacity [103]. As the lifetime of an SSD depends on the size (increasing the size arbitrarily prolongs the lifetime), we report the expected lifetime when the SSD is the same size as the ORAM.

FL accuracy study. We used RF^2 [74], an FL simulator for recommendation models. We used MovieLens and Taobao datasets to train the DLRM [85] model. We used the default setup of RF^2 from its Github repository [73] except for: (1) we did not use an ℓ_2 regularizer for the embedding tables as it becomes impractical for large tables and (2) added a dropout at the end of the MLP layer with $p = 0.5$ for MovieLens for improved accuracy. Taobao did not benefit from dropout.

Baseline. We implemented *Path ORAM+*, which follows the general structure of FEDORA (Figure 4) but uses an SSD-friendly variant of Path ORAM as the main ORAM. We implemented the SSD-friendly Path ORAM by adopting several optimizations from prior work [101], achieving competitive performance with the numbers reported by the prior work [101]. Path ORAM+ always accessed the main ORAM for each user request (Strawman 1 from Section 3.2) for perfect privacy.

6.2 FEDORA Increases the SSD Lifetime

Figure 7 summarizes the SSD lifetime of different setups. Path ORAM+ and FEDORA with perfect FDP ($\epsilon = 0$) only have one bar (All) because the behavior does not change with users' private data distribution (both always accesses the main ORAM for each user request). When $\epsilon > 0$, FEDORA is able to skip some duplicate accesses (Figure 3), gaining more improvements when the data distribution is skewed.

Overall lifetime. *Path ORAM+* suffers from an extremely short SSD lifetime (2 days–1.2 months for Small, 8 days–3.9 months for Medium, and 2–23 months for large) that

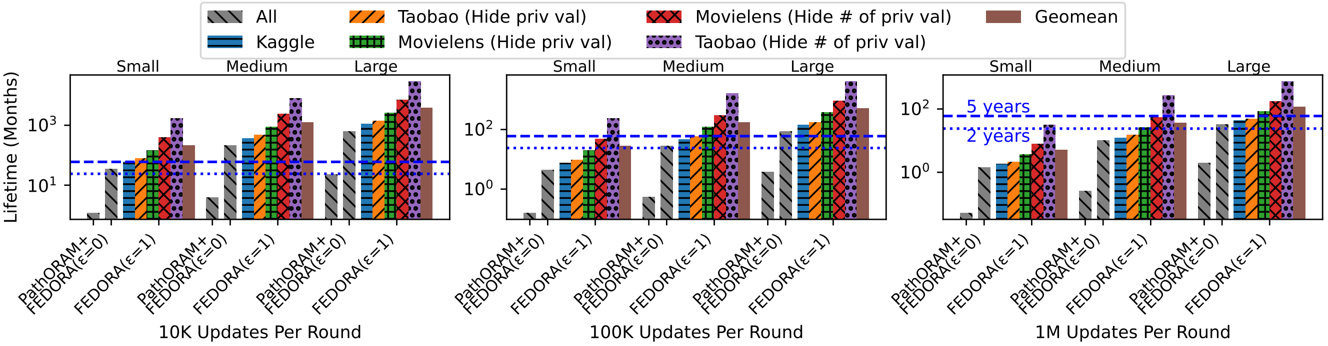

Figure 7. FEDORA significantly improves the expected SSD lifetime over the baseline.

makes using SSD unrealistic. FEDORA's SSD-friendly optimizations significantly increase the lifetime. Even with $\epsilon = 0$, our SSD-friendly optimizations improve the SSD lifetime to become 1.4 months–3 years for Small, 10 months–10+ years for Medium, and over 2.7 years for Large. With $\epsilon = 1$, the lifetime improves even more. The improvement depends on the users' private feature distributions, achieving an average lifetime improvement over the $\epsilon = 0$ case ranging from **1.56**× (Kaggle) to **38**× (Taobao, Hide # of priv vals). Compared to Path ORAM+, FEDORA improves SSD lifetime by **21**× to over **1000**× when $\epsilon = 1$. In many cases, FEDORA achieves an SSD lifetime near or over the typical SSD replacement period (2–5 years).

Reduction in ORAM accesses. The increased lifetime of the $\epsilon = 1$ case is mainly due to the reduced number of main ORAM accesses. Table 1 summarizes the reduced number of accesses (*Reduced Accesses* column) with different ϵ compared to the perfectly privacy case ($\epsilon = 0$). The table shows that trading off a small amount of privacy ($\epsilon > 0$) cuts down the number of accesses to the main ORAM by **51.06–97.69%**. The benefit depends on the skewness of the datasets. *Taobao (hide # priv vals)* enjoys the most benefit as the secret (number of items a user purchased) is extremely skewed—heavy shoppers have hundreds of items in their purchase histories, while many others have empty histories.

The reduction does not vary a lot when decreasing ϵ because both the additional dummy accesses and the lost accesses increase, canceling out each other (as in Figure 3(c, e)). When the reduction is extreme (over 90%, hide # priv vals), increasing privacy increases the number of total accesses. This is because the number of dummy accesses grows much faster than lost accesses as the left side of the distribution, *e.g.*, the orange part of Figure 3 (c, e), cannot grow beyond 0.

6.3 FEDORA Improves the End-to-end Latency

Figure 8 shows the additional overheads introduced to the assumed 2-minute latency of FL. When the number of updates per round is small (10K), even Path ORAM+ adds less than 5% overhead as the per-round latency of FL (2 minutes) is already slow. However, when the number of updates is larger (100K–1M), Path ORAM+ starts to add non-negligible latency overheads of 39–75% and 354–838% for 100K and 1M updates, respectively. FEDORA with $\epsilon = 0$ improves the latency overheads into 32–42% and 322–446% through its SSD-friendly optimizations. With $\epsilon > 0$, the overheads again improve due to the reduced main ORAM accesses. FEDORA with $\epsilon = 1$ only adds on average 10–15% for 100K updates and 104–155% for 1M updates, which is 1.6–6.2× and 1.6–6× improvements over $\epsilon = 0$. Adding everything together, the latency improvement over Path ORAM+ becomes **2–24**× (10K), **1.9–9.1**× (100K), and **1.8–9.7**× (1M) when $\epsilon = 1$. The improved numbers make using an SSD immediately practical with 10K—100K updates per round. When larger numbers of updates are desired (1M), FEDORA with $\epsilon = 1$ adds around 2–3 minutes of latency, which may be tolerable depending on the use case.

6.4 Private Features Improve the Model Quality

Table 1 summarizes the final model quality (*AUC* column) in terms of receiver operating characteristic area under the curve (ROC-AUC or AUC, higher is better), a common metric for recommendation models [74, 85]. The first two rows (*pub*) show the AUC when not using private features during training (0.7104, 0.5902). These represent scenarios without FEDORA. Rows with $\epsilon = \infty$ represent FEDORA without FDP (Strawman 2 in Section 3.2). The corresponding AUC numbers (0.7931 and 0.7972 for MovieLens, 0.5972 for Taobao) show that using private features indeed improves the model quality significantly. As $\epsilon = \infty$ is not private, these numbers represent an upper bound of what FEDORA can achieve.

The rest of the rows with $\epsilon = \{0.1, 1.0\}$ show the achieved accuracy of FEDORA with different levels of privacy. In general, higher privacy ($\epsilon = 0.1$) achieves less accuracy. However, all setups achieve significantly higher accuracy than the baseline, which only uses non-private features (*pub*). Note that in recommendation models, even a small improvement in accuracy (0.1% [134] or 0.001 AUC [136]) is considered significant. FEDORA improves the accuracy even when some

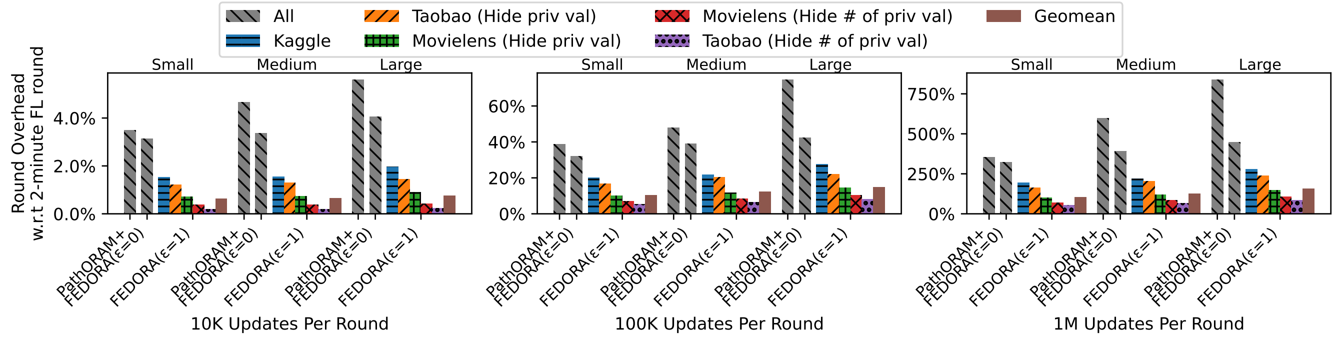

Figure 8. FEDORA improves the overhead ORAM adds to each round of FL.

Table 1. ORAM access reduction and model quality under different ϵ-FDP settings. *Reduced Accesses* is the percentage of accesses saved compared to the perfect privacy ORAM ($\epsilon = 0$). *Dummy* and *Lost* are the percentage of dummy and lost accesses compared to the optimal number of accesses ($\epsilon = \infty$). *pub* is trained without using private features.

	Dataset	ϵ	**Reduced Accesses**	Dummy	Lost	**AUC**
pub	MovieLens	-	-	-	-	0.7104
	Taobao	-	-	-	-	0.5902
hide priv val	MovieLens	∞	**52.5%**	0%	0%	**0.7972**
		1.0	**52.5%**	0.1%	0.1%	**0.7955**
		0.1	**52.5%**	1.1%	1.1%	**0.7944**
	Taobao	∞	**51.11%**	0%	0%	**0.5972**
		1.0	**51.11%**	1.01%	1.01%	**0.5972**
		0.1	**51.06%**	9.89%	9.77%	**0.5970**
hide # of priv vals	MovieLens	∞	**91.13%**	0%	0%	**0.7931**
		1.0	**91.02%**	11.65%	10.47%	**0.7924**
		0.1	**78.37%**	156.0%	12.16%	**0.7929**
	Taobao	∞	**99.05%**	0%	0%	**0.5972**
		1.0	**97.69%**	156%	12.89%	**0.5967**
		0.1	**80.78%**	1924%	2.42%	**0.5970**

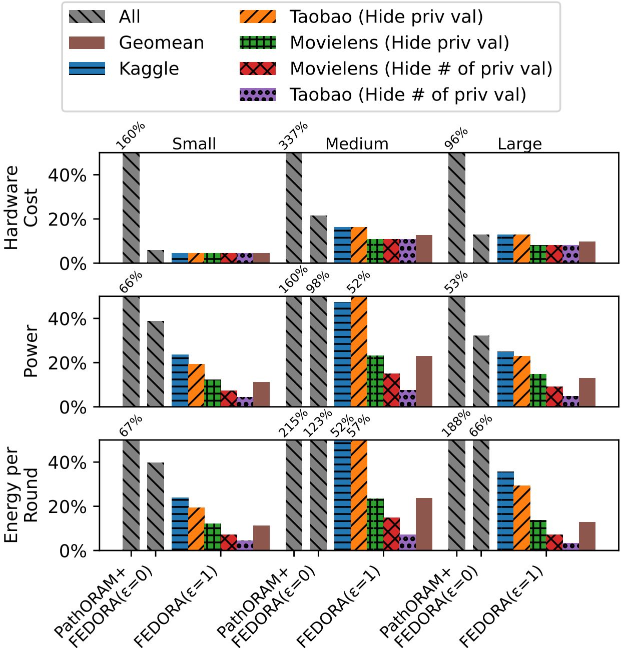

Figure 9. Estimated hardware cost, power, and energy consumption. The values are normalized with the DRAM-based.

entries are lost due to ϵ-FDP because the lost number of entries is reasonably small (Table 1, *Lost* column).

6.5 FEDORA Reduces Power, Energy, and Cost

Figure 9 plots the estimated hardware cost, power, and energy consumption of FEDORA and Path ORAM+, normalized by the numbers from a DRAM-based alternative that uses a large DRAM to hold the main ORAM. To calculate the hardware cost, we assume all hardware is replaced every five years or when the SSD wears out, whichever happens first. We assume DRAM cost of \$3.15/GB and SSD cost of \$0.1/GB [79]. We assumed DRAM draws a constant power of 375 mW/GB [61] and SSD draws its rated power of 6.2 W [99] when it is actively reading or writing. For energy calculation, we projected an optimistic DRAM-based design's latency by subtracting the SSD-related latencies from FEDORA. We used the analytical numbers for the DRAM-based design as we did not have enough DRAM to run the actual design.

While SSD is much cheaper than DRAM, Path ORAM+ is not cheaper in terms of hardware cost, as the SSD wears out too quickly and must be replaced too frequently. By significantly increasing the SSD lifetime, FEDORA ($\epsilon = 1$) reduces the hardware cost by **6-22**× over the DRAM-based design. Similarly, Path ORAM+ avoids the idle power of the large DRAM but incurs frequent high-power SSD reads/writes. FEDORA ($\epsilon = 1$) again reduces the power and the energy consumption by **1.9–23**× compared to the DRAM-based design through fewer SSD reads/writes.

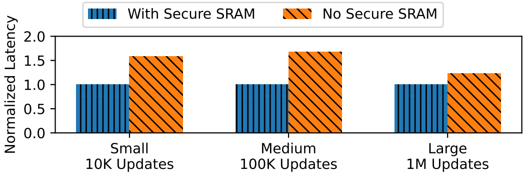

Figure 10. Normalized training latency of FEDORA with or without the 4 KB scratch space in on-chip SRAM.

6.6 Additional Ablation Studies

Importance of a small on-chip SRAM. Our prototype assumes a small (4 KB) on-chip scratchpad in the TEE, which is used as a scratch space to accelerate path eviction. Alternatively, it is possible to implement FEDORA on a TEE without such scratch space, placing everything in off-chip DRAM. This comes at the cost of increased linear scanning during the eviction process. Figure 10 shows the slowdown when the scratch space is not available. A small scratch space greatly (~1.5×) helps FEDORA when block (entry) size is small (Figure 10, Small and Medium). As the block size gets larger (Figure 10, Large), the on-chip SRAM becomes less helpful.

Choosing the bucket sizes. In general, we found it to be helpful to make the bucket size a multiple of 4 KB to match the SSD read/write granularity. Within the constraint, choosing the bucket size is a balancing act of trading between the SSD lifetime and the latency. As described in Section 4.4, increasing the bucket size increases the SSD lifetime by allowing a larger eviction period A. At the same time, the larger bucket size increases the read/write latency as more data moves on each read and write. For all the evaluations, we used a bucket size of 4 KB. 4 KB is already quite large (we can achieve up to $A = 92$), and increasing the bucket further had diminishing returns. For example, increasing the bucket size of the Small table from 4 KB to 16 KB increases the SSD lifetime by only 18% but increases latency by 67%. Larger buckets can be used when a longer lifetime is desired over lower latency.

7 Additional Related Work

Secure federated submodel learning (SFSL) [87] is a prior work on FL for recommendation models. SFSL tried to hide the private feature values of users by directly perturbing them with DP noise. However, SFSL does not achieve FDP and provides much weaker protection because it uses a neighbor definition different from Definition 3.2. Instead of replacing a feature value from d to an *arbitrary value* to get d', SFSL replaces feature values to a *value that some other users requested* in the same round, narrowing down the possible feature values to a small number of candidates. This unintuitive choice with much weaker privacy was because using the neighbor definition from Definition 3.2 did not give good final accuracy for SFSL, as the authors acknowledged [87]. Unlike SFSL, FEDORA uses ORAM to hide the accesses and additionally uses DP (ϵ-FDP) to control the leakage through the number of ORAM accesses. FEDORA can achieve much better accuracy with the correct neighbor definition (Definition 3.2) because FEDORA adds DP noise to an aggregate statistic (number of ORAM accesses), while SFSL adds noise to individual feature values. It is well-known that DP works much better on aggregated statistics [22].

Other ORAMs. Most ORAMs either use a binary tree [9, 11, 13–15, 26, 56, 67, 81, 90, 91, 93, 94, 96, 101, 118, 119, 128, 131, 132] or oblivious shuffling [5, 30, 69, 106, 107, 113, 129]. FEDORA uses the former due to its lower read/write overheads. The latter incurs frequent and large writes to storage, making them unsuitable for FL. LAORAM [90] is a variant of ORAM that also targets recommendation model training. However, LAORAM optimizes for a centralized GPU training [90], and the requirements and optimizations totally differ from FEDORA. FEDORA's ϵ-FDP may be applicable to systems like LAORAM as well.

FL for recommendation models. Other than SFSL [87], recommendation models haven't been studied a lot in FL due to their large embedding tables. Most existing works use simple models with tiny embedding tables [10, 21, 39, 45, 51, 63, 66, 74, 80, 117]. Singhal et al. [102] does not communicate the large table and locally approximate the table on user devices. Using tiny tables or approximations inevitably degrades the model's accuracy, especially when the datasets are complex.

Hiding embedding table accesses. Several different methods have been explored to hide the embedding table access pattern of recommendation models in different contexts. LAORAM [90] used ORAM in the context of centralized training. Lam et al. [58] used two-server private information retrieval (PIR) for on-device inference. Others [24, 42, 72] added statistical noise, similar to SFSL [87]. These approaches all optimize for different application use cases and not directly comparable to FEDORA.

Recommendation models with an SSD. Several works ran recommendation model training or inference with embedding tables placed inside an SSD [23, 104, 111, 116, 122]. These works did not focus on privacy and did not use ORAM.

8 Conclusion

Recommendation model training currently requires the service provider to collect sensitive user data. Existing federated learning solutions cannot be adopted for recommendation models due to the large embedding tables. We present FEDORA, an ORAM-based FL system for recommendation models with large embedding tables. FEDORA introduces ϵ-FDP to control the privacy and efficiency of ORAM-based FL and places the large ORAM inside an SSD with several SSD-friendly optimizations. FEDORA maintains high training accuracy by privately using sensitive features while improving the SSD lifetime and latency significantly. Compared to a DRAM-based system, FEDORA uses much less hardware cost, power, and energy.

A Artifact Appendix

A.1 Abstract

We present two pieces of artifacts for evaluation. One is a modified version of RF^2 [74] that performs federated learning simulations (FLSim) that produce Table 1. The other is C++ code that implements the proposed FEDORA system (ORAMSim) and is used to evaluate the system's performance (Figures 7 and 8).

A.2 Artifact check-list (meta-information)

- **Compilation:**
  - FLSim: None
  - ORAMSim: g++ and make
- **Run-time environment:**
  - FLSim: PyTorch Environment
  - ORAMSim: Ubuntu 22.04 or later
- **Hardware:**
  - FLSim: a CUDA-capable GPU
  - ORAMSim: x86-64 CPU with AVX2 support, NVME SSD formatted in ext4
- **Output:**
  - FLSim: reduced accesses, wasted reads, lost reads, and AUC similar to those presented in Table 1.
  - ORAMSim: Plots similar to Figures 7 and 8.
- **Experiments:**
  - FLSim: Federated learning simulation
  - ORAMSim: FEDORA system performance evaluation
- **How much disk space required (approximately)?:**
  - FLSim: 10GB
  - ORAMSim: 100GB
- **How much time is needed to prepare workflow (approximately)?:** 2 hours
- **How much time is needed to complete experiments (approximately)?:** 12 hours
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:**
  - FLSim: Creative Commons Attribution-NonCommercial 4.0 International
  - ORAMSim: BSD 3-Clause
- **Workflow automation framework used?:** Python and Bash scripts.
- **Archived (provide DOI)?:**
  - FLSim: 10.5281/zenodo.14812749
  - ORAMSim: 10.5281/zenodo.14812528

A.3 Description

A.3.1 How to access.

- FLSim: https://github.com/psu-paws/FEDORA-FLSim
- ORAMSim: https://github.com/psu-paws/FEDORA-OramSim

A.3.2 Hardware dependencies.

A CUDA-capable GPU is required to run FLSim. To run ORAMSim, an x86-64 multicore CPU with AVX2 support and an NVME SSD formatted in ext4 are required. We used an Intel i7-13700K and a Samsung PM9A1 1TB SSD. About 100GB of free disk space is required. Preferably, an SSD can be dedicated to the experiments.

A.3.3 Software dependencies.

To run FLSim, a PyTorch environment is needed. We ran our ORAMSim experiments on Ubuntu 22.04.5 LTS. Please see the following list of software dependencies:

- GCC 11.4 or higher
- CMake 4.3 or higher
- Libsodium 1.0.19-stable or higher
- LibAIO
- Python 3.10.12 or higher

A.3.4 Data sets.

The datasets used in the experiment are downloaded as part of the installation process. Refer to Section 6.1 for detailed description.

A.4 Installation

A.4.1 FLSim.

- Clone repository.

```
git clone \
git@github.com:psu-paws/FEDORA-FLSim.git
```

- Change directory into the project.
- Setup a Python environment with PyTorch. Refer to https://pytorch.org/ for instructions. Using a Conda or a Python virtual environment is recommended.
- Install dependencies specified in `requirements.txt`. `pip install -r requirements.txt`
- Run `./download_dataset.sh` to download and unpack Taobao Ads Click and Display[88] and MovieLens 20M[36] datasets.

A.4.2 ORAMSim.

- Install GCC, CMake, LibAIO, and other dependencies. On Ubuntu 24.04, run the following command:

```
sudo apt install build-essential \
python3-dev cmake libaio1t64 libaio-dev
```

- Install Libsodium following instructions at https://doc.libsodium.org/installation
- Clone repository. Note `--recurse-submodules` flag.

```
git clone --recurse-submodules \
git@github.com:psu-paws/FEDORA-OramSim.git
```

- Change directory into the project.
- run `./build.sh`. This should configure and build the project, placing the outputs in a directory named `build`.

- Download trace files from https://doi.org/10.5281/zenodo.14818427. Decompress the zip archive into the project's root directory. This should create a directory named `input-traces`.
- Setup a Python environment. Using a Conda or a Python virtual environment is recommended.
- Install Python dependencies specified in `requirements.txt`. `pip install -r requirements.txt`

A.5 Experiment workflow

A.5.1 FLSim.

- Optionally select the GPU you wish to use by setting `CUDA_VISIBLE_DEVICES`.
- run `./run_tests.sh`. This could take several hours, depending on the speed of your GPU.
- run `python process_logs.py` to generate a CSV file similar to Table 1.

A.5.2 ORAMSim.

- Set `ORAM_WORKING_DIR` environment variable to point to the NVME drive you want to use for the experiment. For example, if the drive is mounted at `/data`, then you should run `export ORAM_WORKING_DIR=/data` . You can leave it unset to use the current directory, although this is not recommended.
- Run `python generate_orams.py` to create the ORAMs. This is slow and can take multiple hours. You should find the created ORAMs in the `orams` directory.
- Run `python recsys_sim.py` to run the experiments. This is slow and can take multiple hours. You should find the run statistics in the `experiments` directory
- Run `python get_latency_recsys_results.py` to collect the results into a CSV file.
- Run `python make_graphs_for_paper.py` to generate plots similar to Figures 7 and 8 in the paper.

A.6 Evaluation and expected results

A.6.1 FLSim. The AUC, reduced accesses, dummy, and lost values should be close to the corresponding ones for MovieLens in Table 1.

A.6.2 ORAMSim. The code should produce graphs similar to Figures 7 and 8, except without data from the "Large" configuration. `Recsys_lifespan_months.pdf` should be similar to Figure 7. `Recsys_latency.pdf` should be similar to Figure 8. The absolute latency numbers may vary depending on hardware, but overall trends should remain similar.

A.7 Experiment customization

A.7.1 FLSim. We have omitted evaluation on the Taobao Ads Click and Display dataset[88] to make the runtime reasonable. If you want to run the Taobao-based experiments, uncomment lines 11-17 in `run_experiments.sh`.

A.7.2 ORAMSim. We have shortened the experiment and excluded the "Large" configuration to make the runtime reasonable. If you want to run the full-length experiment, please make the following modifications.

- Uncomment line 99-100 in `generate_orams.py`, this will enable the "Large" configuration.
- Change line 22 in `recsys_sim.py` from 1 million to 10 million. This will increase the number of accesses performed in each test case.

Note this requires 500GB of additional disk space and might take over a day to complete.

A.8 Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae

References

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 308–318. ACM, 2016.

[2] Arm. Trustzone for cortex-a. https://www.arm.com/technologies/trustzone-for-cortex-a, 2023.

[3] ARM Ltd. Arm Cortex-M series processors. https://developer.arm.com/ip-products/processors/cortex-m, 2021.

[4] Hilal Asi, John Duchi, and Omid Javidbakht. Element level differential privacy: The right granularity of privacy. *arXiv preprint arXiv:1912.04042*, 2019.

[5] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 837–849. ACM, 2015.

[6] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. Reconstructing individual data points in federated learning hardened with differential privacy and secure aggregation. In *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*, pages 241–257. IEEE, 2023.

[7] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. In *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*, pages 175–199. IEEE, 2023.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[9] Dingyuan Cao, Mingzhe Zhang, Hang Lu, Xiaochun Ye, Dongrui Fan, Yuezhi Che, and Rujia Wang. Streamline ring ORAM accesses through spatial and temporal optimization. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021, Seoul, South Korea, February 27 - March 3, 2021*, pages 14–25. IEEE, 2021.

[10] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Secure federated matrix factorization. *IEEE Intelligent Systems*, 36(5):11–20, 2020.

[11] Anrin Chakraborti, Adam J. Aviv, Seung Geol Choi, Travis Mayberry, Daniel S. Roche, and Radu Sion. roram: Efficient range ORAM with o(log2 N) locality. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019.

[12] Prasenjit Chakraborty, Preeti Ranjan Panda, and Sandeep Sen. Partitioning and data mapping in reconfigurable cache and scratchpad memory–based architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 22(1), September 2016.

[13] Yuezhi Che, Yuan Hong, and Rujia Wang. Imbalance-aware scheduler for fast and secure ring ORAM data retrieval. In *37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, November 17-20, 2019*, pages 604–612. IEEE, 2019.

[14] Yuezhi Che and Rujia Wang. Multi-range supported oblivious RAM for efficient block data retrieval. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*, pages 369–382. IEEE, 2020.

[15] Hao Chen, Ilaria Chillotti, and Ling Ren. Onion ring ORAM: efficient constant bandwidth oblivious RAM from (leveled) TFHE. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 345–360. ACM, 2019.

[16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In Alexandros Karatzoglou, Balázs Hidasi, Domonkos Tikk, Oren Sar Shalom, Haggai Roitman, Bracha Shapira, and Lior Rokach, editors, *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 7–10. ACM, 2016.

[17] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells, editors, *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, pages 191–198. ACM, 2016.

[18] Criteo AI Labs. Criteo Research Datasets. https://ailab.criteo.com/ressources/, 2024.

[19] Ian Cutress. A few notes on intel's knights landing and mcdram modes from sc15. https://www.anandtech.com/show/9794/a-few-notes-on-intels-knights-landing-and-mcdram-modes-from-sc15, 2015.

[20] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[21] Erika Duriakova, Elias Z Tragos, Barry Smyth, Neil Hurley, Francisco J Peña, Panagiotis Symeonidis, James Geraci, and Aonghus Lawlor. Pdmfrec: a decentralised matrix factorisation with tunable user-centric privacy. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 457–461, 2019.

[22] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[23] Assaf Eisenman, Maxim Naumov, Darryl Gardner, Misha Smelyanskiy, Sergey Pupyrev, Kim M. Hazelwood, Asaf Cidon, and Sachin Katti. Bandana: Using non-volatile memory for storing deep learning models. In Ameet Talwalkar, Virginia Smith, and Matei Zaharia, editors, *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019.

[24] Oluwaseyi Feyisetan, Borja Balle, Thomas Drake, and Tom Diethe. Privacy- and utility-preserving textual analysis via calibrated multivariate perturbations. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 178–186. ACM, 2020.

[25] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pages 3–8, 2012.

[26] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, Emil Stefanov, Dimitrios N. Serpanos, and Srinivas Devadas. A low-latency, low-area hardware oblivious RAM controller. In *23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, Vancouver, BC, Canada, May 2-6, 2015*, pages 215–222. IEEE Computer Society, 2015.

[27] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[28] GDPR.EU. A guide to GDPR data privacy requirements. https://gdpr.eu/data-privacy/, 2024.

[29] Arpita Ghosh and Robert Kleinberg. Inferential privacy guarantees for differentially private mechanisms. *arXiv preprint arXiv:1603.01508*, 2016.

[30] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 182–194. ACM, 1987.

[31] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[32] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptol. ePrint Arch.*, page 204, 2016.

[33] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1725–1731. ijcai.org, 2017.

[34] Ian Hamilton. Oculus quest keyboard option sends 'aggregate modeling data' to facebook. https://www.uploadvr.com/facebook-quest-keyboard-data/, 2021.

[35] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[36] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[37] Hanieh Hashemi, Wenjie Xiong, Liu Ke, Kiwan Maeng, Murali Annavaram, G Edward Suh, and Hsien-Hsin S Lee. Private data leakage via exploiting access patterns of sparse features in deep learning-based recommendation systems. In *Workshop on Trustworthy and Socially Responsible Machine Learning, NeurIPS 2022*, 2022.

[38] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 173–182. ACM, 2017.

[39] István Hegedűs, Gábor Danner, and Márk Jelasity. Decentralized recommendation based on matrix factorization: A comparison of gossip and federated learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 317–332. Springer, 2019.

[40] Ing-Jer Huang, Chun-Hung Lai, Yun-Chung Yang, Hsu-Kang Dow, and Hung-Lun Chen. A reconfigurable cache for efficient use of tag ram as scratch-pad memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(4):663–670, 2018.

[41] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. PAPAYA: practical, private, and scalable federated learning. In Diana Marculescu, Yuejie Chi, and Carole-Jean Wu, editors, *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. mlsys.org, 2022.

[42] Jacob Imola, Shiva Kasiviswanathan, Stephen White, Abhinav Aggarwal, and Nathanael Teissier. Balancing utility and scalability in metric differential privacy. In *Uncertainty in Artificial Intelligence*, pages 885–894. PMLR, 2022.

[43] Intel. Intel® software guard extensions. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, 2023.

[44] Intel. Intel® trust domain extensions (intel® tdx). https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html, 2024.

[45] Amir Jalalirad, Marco Scavuzzo, Catalin Capota, and Michael Sprague. A simple and efficient federated recommender system. In *Proceedings of the 6th IEEE/ACM international conference on big data computing, applications and technologies*, pages 53–58, 2019.

[46] Yupeng Jiang, Yong Li, Yipeng Zhou, and Xi Zheng. Mitigating sybil attacks on differential privacy based federated learning. *arXiv preprint arXiv:2010.10572*, 2020.

[47] Yupeng Jiang, Yong Li, Yipeng Zhou, and Xi Zheng. Sybil attacks and defense on differential privacy based federated learning. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 355–362. IEEE, 2021.

[48] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1376–1385. JMLR.org, 2015.

[49] Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 197–206. IEEE Computer Society, 2018.

[50] Sanjay Kariyappa, Chuan Guo, Kiwan Maeng, Wenjie Xiong, G. Edward Suh, Moinuddin K. Qureshi, and Hsien-Hsin S. Lee. Cocktail party attack: Breaking aggregation-based privacy in federated learning using independent component analysis. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 15884–15899. PMLR, 2023.

[51] Eugene Kharitonov. Federated online learning to rank with evolution strategies. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 249–257, 2019.

[52] Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Trans. Database Syst.*, 39(1), January 2014.

[53] Seah Kim, Jerry Zhao, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Aurora: Virtualized accelerator orchestration for multi-tenant workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 62–76, New York, NY, USA, 2023. Association for Computing Machinery.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[55] Walid Krichene, Nicolas Mayoraz, Steffen Rendle, Shuang Song, Abhradeep Thakurta, and Li Zhang. Private learning with public features. In *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pages 4150–4158. PMLR, 2024.

[56] Jinxi Kuang, Minghua Shen, Yutong Lu, and Nong Xiao. Exploiting data locality in memory for oram to reduce memory access overheads. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, page 703–708, New York, NY, USA, 2022. Association for Computing Machinery.

[57] Fan Lai, Yinwei Dai, Sanjay Sri Vallabh Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 11814–11827. PMLR, 2022.

[58] Maximilian Lam, Jeff Johnson, Wenjie Xiong, Kiwan Maeng, Udit Gupta, Minsoo Rhu, Hsien-Hsin S. Lee, Vijay Janapa Reddi, Gu-Yeon Wei, David Brooks, and G. Edward Suh. Gpu-based private information retrieval for on-device machine learning inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2024*. ACM, 2024.

[59] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, pages 38:1–38:16. ACM, 2020.

[60] Ruby B. Lee, Peter C. S. Kwan, John Patrick McGregor, Jeffrey S. Dwoskin, and Zhenghong Wang. Architecture for protecting critical secrets in microprocessors. In *32st International Symposium on Computer Architecture (ISCA 2005), 4-8 June 2005, Madison, Wisconsin, USA*, pages 2–13. IEEE Computer Society, 2005.

[61] Seunghak Lee, Ki-Dong Kang, Hwanjun Lee, Hyungwon Park, Younghoon Son, Nam Sung Kim, and Daehoon Kim. Greendimm: Os-assisted dram power management for dram with a sub-array granularity power-down state. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 131–142, New York, NY, USA, 2021. Association for Computing Machinery.

[62] Kif Leswing. Facebook says apple ios privacy change will result in $10 billion revenue hit this year. https://www.cnbc.com/2022/02/02/facebook-says-apple-ios-privacy-change-will-cost-10-billion-this-year.html, 2022.

[63] Dongsheng Li, Chao Chen, Qin Lv, Li Shang, Yingying Zhao, Tun Lu, and Ning Gu. An algorithm for efficient privacy-preserving item-based collaborative filtering. *Future Generation Computer Systems*, 55:311–320, 2016.

[64] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A systematic look at ciphertext side channels on AMD SEV-SNP. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 337–351. IEEE, 2022.

[65] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX*

Security 2021, August 11-13, 2021, pages 717–732. USENIX Association, 2021.

[66] Mu Li, Ziqi Liu, Alexander J Smola, and Yu-Xiang Wang. Difacto: Distributed factorization machines. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 377–386, 2016.

[67] Xiang Li, Yunqian Luo, and Mingyu Gao. Bulkor: Enabling bulk loading for path oram. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4258–4276, 2024.

[68] Juntaek Lim, Youngeun Kwon, Ranggi Hwang, Kiwan Maeng, G. Edward Suh, and Minsoo Rhu. Lazydp: Co-designing algorithm-software for scalable training of differentially private recommendation models. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2024.* ACM, 2024.

[69] Liang Liu, Rujia Wang, Youtao Zhang, and Jun Yang. H-ORAM: A cacheable ORAM interface for efficient I/O accesses. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 33. ACM, 2019.

[70] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. Understanding capacity-driven scale-out neural recommendation inference. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2021, Stony Brook, NY, USA, March 28-30, 2021*, pages 162–171. IEEE, 2021.

[71] Jialun Lyu, Jaylen Wang, Kali Frost, Chaojie Zhang, Celine Irvene, Esha Choukse, Rodrigo Fonseca, Ricardo Bianchini, Fiodar Kazhamiaka, and Daniel S. Berger. Myths and misconceptions around reducing carbon embedded in cloud platforms. In George Porter, Tom Anderson, Andrew A. Chien, Tamar Eilam, Colleen Josephson, and Jonggyu Park, editors, *Proceedings of the 2nd Workshop on Sustainable Computer Systems, HotCarbon 2023, Boston, MA, USA, 9 July 2023*, pages 7:1–7:7. ACM, 2023.

[72] Kiwan Maeng, Chuan Guo, Sanjay Kariyappa, and G Edward Suh. Bounding the invertibility of privacy-preserving instance encoding using fisher information. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023.

[73] Kiwan Maeng, Haiyu Lu, Luca Melis, John Nguyen, Mike Rabbat, and Carole-Jean Wu. Rf2. https://github.com/facebookresearch/RF2.

[74] Kiwan Maeng, Haiyu Lu, Luca Melis, John Nguyen, Mike Rabbat, and Carole-Jean Wu. Towards fair federated recommendation learning: Characterizing the inter-dependence of system and data heterogeneity. In Jennifer Golbeck, F. Maxwell Harper, Vanessa Murdock, Michael D. Ekstrand, Bracha Shapira, Justin Basilico, Keld T. Lundgaard, and Even Oldridge, editors, *RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022*, pages 156–167. ACM, 2022.

[75] Peter Mattson, Christine Cheng, Gregory F. Diamos, Cody Coleman, Paulius Micikevicius, David A. Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debo Dutta, Udit Gupta, Kim M. Hazelwood, Andy Hock, Xinyuan Huang, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. Mlperf training benchmark. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020.* mlsys.org, 2020.

[76] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson, and Rishabh Mehrotra. Explore, exploit, and explain: personalizing explainable recommendations with bandits. In Sole Pera, Michael D. Ekstrand, Xavier Amatriain, and John O'Donovan, editors, *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, pages 31–39. ACM, 2018.

[77] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017.

[78] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, 2018.

[79] Matthew Mead. History of costs of ram vs. hard drives vs. ssds. https://azrael.digipen.edu/ mmead/www/Courses/CS180/ram-hd-ssd-prices.html, 2022.

[80] Xuying Meng, Suhang Wang, Kai Shu, Jundong Li, Bo Chen, Huan Liu, and Yujun Zhang. Personalized privacy-preserving social recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[81] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296, 2018.

[82] Sparsh Mittal. A survey of techniques for cache locking. *ACM Trans. Des. Autom. Electron. Syst.*, 21(3), May 2016.

[83] Jianqiao Mo, Jayanth Gopinath, and Brandon Reagen. Haac: A hardware-software co-design to accelerate garbled circuits. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[84] Milad Nasr, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlini. Adversary instantiation: Lower bounds for differentially private machine learning. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 866–882. IEEE, 2021.

[85] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[86] Lin Ning, Steve Chien, Shuang Song, Mei Chen, Yunqi Xue, and Devora Berlowitz. EANA: reducing privacy risk on large-scale recommendation models. In Jennifer Golbeck, F. Maxwell Harper, Vanessa Murdock, Michael D. Ekstrand, Bracha Shapira, Justin Basilico, Keld T. Lundgaard, and Even Oldridge, editors, *RecSys '22: Sixteenth ACM Conference on Recommender Systems, Seattle, WA, USA, September 18 - 23, 2022*, pages 399–407. ACM, 2022.

[87] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[88] Pavan Sabnagapati. Ad Display/Click Data on Taobao.com. https://www.kaggle.com/datasets/pavansanagapati/ad-displayclick-data-on-taobaocom, 2020.

[89] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H. Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ML: A practical

guide to machine learning with differential privacy. *J. Artif. Intell. Res.*, 77:1113–1201, 2023.

[90] Rachit Rajat, Yongqin Wang, and Murali Annavaram. LAORAM: A look ahead ORAM architecture for training large embedding tables. In Yan Solihin and Mark A. Heinrich, editors, *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023, Orlando, FL, USA, June 17-21, 2023*, pages 76:1–76:15. ACM, 2023.

[91] Rachit Rajat, Yongqin Wang, and Murali Annavaram. Pageoram: An efficient dram page aware oram strategy. In *Proceedings of the 55th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '22, page 91–107. IEEE Press, 2023.

[92] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. *CoRR*, abs/2305.05065, 2023.

[93] Mehrnoosh Raoufi, Jun Yang, Xulong Tang, and Youtao Zhang. Aboram: Constructing adjustable buckets for space reduction in ring oram. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 361–373, 2023.

[94] Mehrnoosh Raoufi, Youtao Zhang, and Jun Yang. Ir-oram: Path access type based memory intensity reduction for path-oram. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 360–372, 2022.

[95] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[96] Ling Ren, Christopher W. Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Ring ORAM: closing the gap between small and large client storage oblivious RAM. *IACR Cryptol. ePrint Arch.*, page 997, 2014.

[97] Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious ram in secure processors. *SIGARCH Comput. Archit. News*, 41(3):571–582, jun 2013.

[98] Holger Roth, Michael Zephyr, and Ahmed Harouni. Federated learning with homomorphic encryption. https://developer.nvidia.com/blog/federated-learning-with-homomorphic-encryption/, 2021.

[99] Samsung. Samsung nvme™ ssd 980 pro data sheet. https://download.semiconductor.samsung.com/resources/data-sheet/Samsung-NVMe-SSD-980-PRO-Data-Sheet_Rev.2.1_230509_10129505081019.pdf, 2021.

[100] Ahmed E Samy and Šarūnas Girdzijauskas. Mitigating sybil attacks in federated learning. In *International Conference on Information Security Practice and Experience*, pages 36–51. Springer, 2023.

[101] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. Zerotrace : Oblivious memory primitives from intel SGX. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[102] Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, John Rush, and Sushant Prakash. Federated reconstruction: Partially local federated learning. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11220–11232, 2021.

[103] Solidigm. D7-p5620. https://www.solidigm.com/products/data-center/d7/p5620.html.

[104] Mohammadreza Soltaniyeh, Veronica Lagrange Moutinho dos Reis, Matthew Bryson, Xuebin Yao, Richard P. Martin, and Santosh Nagarakatte. Near-storage processing for solid state drive based recommendation inference with smartssds®. In Dan Feng, Steffen Becker, Nikolas Herbst, and Philipp Leitner, editors, *ICPE '22: ACM/SPEC International Conference on Performance Engineering, Bejing, China, April 9 - 13, 2022*, pages 177–186. ACM, 2022.

[105] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. Deep learning for recommender systems: A netflix case study. *AI Mag.*, 42(3):7–18, 2021.

[106] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious cloud storage. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 253–267. IEEE Computer Society, 2013.

[107] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. Towards practical oblivious RAM. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[108] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 299–310. ACM, 2013.

[109] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: architecture for tamper-evident and tamper-resistant processing. In Utpal Banerjee, Kyle A. Gallivan, and Antonio González, editors, *Proceedings of the 17th Annual International Conference on Supercomputing, ICS 2003, San Francisco, CA, USA, June 23-26, 2003*, pages 160–171. ACM, 2003.

[110] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In Wenwu Zhu, Dacheng Tao, Xueqi Cheng, Peng Cui, Elke A. Rundensteiner, David Carmel, Qi He, and Jeffrey Xu Yu, editors, *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 1441–1450. ACM, 2019.

[111] Xuan Sun, Hu Wan, Qiao Li, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. RM-SSD: in-storage computing for large-scale recommendation inference. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2-6, 2022*, pages 1056–1070. IEEE, 2022.

[112] The White House. Executive Order on Preventing Access to Americans' Bulk Sensitive Personal Data and United States Government-Related Data by Countries of Concern. https://www.whitehouse.gov/briefing-room/presidential-actions/2024/02/28/executive-order-on-preventing-access-to-americans-bulk-sensitive-personal-data-and-united-states-government-related-data-by-countries-of-concern/, 2024.

[113] Shruti Tople, Yaoqi Jia, and Prateek Saxena. PRO-ORAM: practical read-only oblivious RAM. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2019, Chaoyang District, Beijing, China, September 23-25, 2019*, pages 197–211. USENIX Association, 2019.

[114] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In Lorenzo Cavallaro, Johannes Kinder, Sadia Afroz, Battista Biggio, Nicholas Carlini, Yuval Elovici, and Asaf Shabtai, editors, *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019*, pages 1–11. ACM, 2019.

[115] Muhammad Umar, Akhilesh Parag Marathe, Monami Dutta Gupta, Shubham Jogprakash Ghosh, G. Edward Suh, and Wenjie Xiong. Efficient memory side-channel protection for embedding generation in machine learning. In *2025 International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2025.

[116] Hu Wan, Xuan Sun, Yufei Cui, Chia-Lin Yang, Tei-Wei Kuo, and Chun Jason Xue. Flashembedding: storing embedding tables in ssd for large-scale recommender systems. In *Proceedings of the 12th ACM*

SIGOPS Asia-Pacific Workshop on Systems, APSys '21, page 9–16, New York, NY, USA, 2021. Association for Computing Machinery.

[117] Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. Flint: A platform for federated learning integration. In *Proceedings of Machine Learning and Systems 2023, MLSys 2023, Miami Beach, FL, USA, June 4 - June 8, 2023.* mlsys.org, 2023.

[118] Rujia Wang, Youtao Zhang, and Jun Yang. Cooperative path-oram for effective memory bandwidth sharing in server settings. In *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, pages 325–336. IEEE Computer Society, 2017.

[119] Rujia Wang, Youtao Zhang, and Jun Yang. D-ORAM: path-oram delegation for low execution interference on cloud servers with untrusted memory. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pages 416–427. IEEE Computer Society, 2018.

[120] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13 - 17, 2017*, pages 12:1–12:7. ACM, 2017.

[121] Ruoxi Wang, Rakesh Shivanna, Derek Zhiyuan Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1785–1797. ACM / IW3C2, 2021.

[122] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. Recssd: near data processing for solid state drive based recommendation inference. In Tim Sherwood, Emery D. Berger, and Christos Kozyrakis, editors, *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, pages 717–729. ACM, 2021.

[123] X Xie, J Lian, Z Liu, X Wang, F Wu, H Wang, and Z Chen. Personalized recommendation systems: Five hot research topics you must know. *Microsoft Research Lab-Asia*, 2018.

[124] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A Choquette-Choo, Peter Kairouz, H Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. Federated learning of gboard language models with differential privacy. *arXiv preprint arXiv:2305.18465*, 2023.

[125] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 935–946. ACM / IW3C2, 2021.

[126] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[127] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed H. Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In Toine Bogers, Alan Said, Peter Brusilovsky, and Domonkos Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 269–277. ACM, 2019.

[128] Xiangyao Yu, Syed Kamran Haider, Ling Ren, Christopher W. Fletcher, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Proram: dynamic prefetcher for oblivious RAM. In Deborah T. Marr and David H. Albonesi, editors, *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, pages 616–628. ACM, 2015.

[129] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root ORAM: efficient random access in multi-party computation. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 218–234. IEEE Computer Society, 2016.

[130] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024.* OpenReview.net, 2024.

[131] Xian Zhang, Guangyu Sun, Peichen Xie, Chao Zhang, Yannan Liu, Lingxiao Wei, Qiang Xu, and Chun Jason Xue. Shadow block: Accelerating oram accesses with data duplication. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 961–973, 2018.

[132] Xian Zhang, Guangyu Sun, Chao Zhang, Weiqi Zhang, Yun Liang, Tao Wang, Yiran Chen, and Jia Di. Fork path: improving efficiency of ORAM by removing redundant memory accesses. In Milos Prvulovic, editor, *Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015*, pages 102–114. ACM, 2015.

[133] Yuanbo Zhang, Daniel Ramage, Zheng Xu, Yanxiang Zhang, Shumin Zhai, and Peter Kairouz. Private federated learning in gboard. *arXiv preprint arXiv:2306.14793*, 2023.

[134] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. Distributed hierarchical GPU parameter server for massive scale deep learning ads systems. In Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze, editors, *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020.* mlsys.org, 2020.

[135] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 5941–5948. AAAI Press, 2019.

[136] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1059–1068. ACM, 2018.

[137] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14747–14756, 2019.