ARTIFACT EVALUATED
USENIX
AVAILABLE

Nudge: A Private Recommendations Engine

Alexandra Henzinger
MIT

Emma Dauterman
MIT and Stanford

Henry Corrigan-Gibbs
MIT

Dan Boneh
Stanford

Abstract

Nudge is a recommender system with cryptographic privacy. A Nudge deployment consists of three infrastructure servers and many users, who retrieve/rate items from a large data set (e.g., videos, posts, businesses). Periodically, the Nudge servers collect ratings from users in secret-shared form, then run a three-party computation to train a lightweight recommender model on users' private ratings. Finally, the servers deliver personalized recommendations to each user. At every step, Nudge reveals nothing to the servers about any user's preferences beyond the aggregate model itself. User privacy holds against an adversary that compromises the entire secret state of one server. The technical core of Nudge is a new, three-party protocol for matrix factorization. On the Netflix data set with half a million users and ten thousand items, Nudge (running on three 192-core servers on a local-area network) privately learns a recommender model in 50 mins with 40 GB of server-to-server communication. On a standard quality benchmark (nDCG@20), Nudge scores 0.29 out of 1.0, on par with non-private matrix factorization and just shy of non-private neural recommenders, which score 0.31.

1 Introduction

Personalized recommendations are ubiquitous, powering everything from social-media feeds to video-streaming apps. The data that fuels these recommender algorithms betrays our every move—our clicks, likes, views, and browsing behaviors—as well as those of millions of other users. Given the scale and scope of this data collection, content recommendation services have become focal points of privacy failure: user data is resold to third parties [15, 160], exploited for profit [44, 53, 160], and stolen in data breaches [140].

To protect against such abuses, techniques for privately *fetching* content (via anonymizing proxies [18, 64] or cryptographic protocols [21, 52]) have advanced and seen deployment, but systems for private *recommendation* lag behind. Anonymity-based schemes extended with persistent identifiers to link queries and allow for recommendation break in the face of de-anonymization attacks [38, 78, 138]. Recommenders built on federated learning leak individual users' data [43, 86, 121, 176] or intermediate model versions [26, 128, 129, 152]; others built on trusted hardware [13, 63] are vulnerable to side-channel attacks [50, 136]. Cryptographically private recommenders [111, 139, 142, 166] run into a problem of scale: they operate on at most thousands of users fetching thousands of items. In sum, existing recommender methods fall short of offering strong privacy protection at large scale.

This paper presents Nudge, a privacy-protecting recommender system that operates on millions of users or items. Nudge splits trust across three servers run by independent parties. These servers collect secret-shared logs that encode many users' interactions with items from a large data set. In a secure three-party computation, the Nudge servers then aggregate this data to synthesize and infer retrieval patterns across queries and across users: in short, they learn a recommender model. Finally, the servers send secret-shared yet personalized recommendations back to each user. All along, Nudge provides a precise form of cryptographic privacy: even if an adversary compromises the state of one Nudge server (i.e., semi-honest security), no party—not the servers, not the users—learns anything about which user interacted with which item, beyond (a) what the aggregate recommender model reveals and (b) how many interactions each user submitted.

Inspired by non-private [113, 154] and private [111, 139, 142, 166] recommender systems, Nudge generates recommendations using matrix factorization. Nudge's core is a co-design of a matrix-factorization algorithm with multi-party computation techniques. Whereas prior cryptographic recommender protocols use general-purpose gradient descent [111, 139, 142, 166], Nudge instead implements the classic power-iteration algorithm [132] that extracts the top eigenvectors of a matrix in a sequence of algebraic steps. Using this new three-party protocol, Nudge factors matrices with tens of billions of entries (e.g., a Yelp data set) in hours of wall-clock time.

Our techniques. We show that power iteration fits into a class of algorithms efficiently computable in a three-party setting.

This is because it alternates between two steps: (a) applying a secret-shared matrix to a secret-shared vector and (b) evaluating simple, non-linear functions on the intermediate results. Replicated secret sharing [20, 134] lets three parties execute step (a) non-interactively and at high speed, while function secret sharing [36, 37] makes it possible to run step (b) with low communication costs. By casting matrix factorization in this high-level framework, we ensure that the communication between servers scales *linearly* with the number of users and items—unlike prior cryptographic recommender protocols with up to three parties [111, 139, 142].

Along the way, we contribute special-purpose three-party protocols for the non-linear functions that Nudge relies on: truncation and normalization. Building on a rich literature in privacy-preserving machine learning [34,42,104,150,163,167, 169], we improve the efficiency of these protocols in Nudge's three-party setting: e.g., reducing the communication by 6× over state-of-the-art truncation [104].

Evaluation. We implement Nudge in Go and C. Nudge runs faster than prior recommender systems with cryptographic privacy: it outperforms two-party approaches based on garbled circuits by four orders of magnitude in communication and compute [139, 142], and runs faster than a bespoke four-party computation while using 8× less communication and one fewer non-colluding party [166].

We evaluate both Nudge's performance and quality on the Netflix prize data set with half a million users and tens of thousands of items [7]. Running on three 192-core machines on a local-area network, Nudge privately learns a recommender model in 50 min and 40 GB of server-to-server communication. After learning the recommendation model, each Nudge server can serve secret-shared recommendations to a user in hundreds of kilobytes of communication (298 KB on Netflix) and seconds of latency (0.38 s on Netflix). Nudge's resulting recommendations match those of cleartext matrix factorization in recall and rank-based metrics: on a standard quality benchmark (nDCG@20), Nudge and non-private matrix-factorization systems both score 0.29 out of 1.0, just shy of non-private deep neural recommenders, which score 0.31.

Moreover, Nudge can scale larger yet: on a Criteo data set with millions of users and hundreds of ad campaigns [6], Nudge's private matrix factorization takes 1.5 h and 124 GB of server-to-server communication. On a Yelp data set with hundreds of thousands of users and hundreds of thousands of businesses [8], Nudge takes 8 h and 250 GB.

Extensions. Finally, we propose a number of extensions that enhance Nudge's performance, security guarantees, and functionality. To scale larger yet, we show how to compose Nudge with classic dimensionality-reduction techniques [105, 145]. To complement its privacy protections, Nudge integrates with well-studied mechanisms that make the model and the recommendations that it outputs differentially private [67, 131]. To defend against disruptions by malicious users, the Nudge servers run a lightweight, one-time well-formedness check on all user submissions, in a way that is almost free in our three-party setting [10, 55].

Lastly, while we present Nudge as a recommender system, algorithms for computing the top eigenvectors of a matrix form the backbone of a wide array of unsupervised learning and data-mining tasks [96]. Our new private power-iteration routine can extend to these same applications: principal component analysis [83], learning low-dimensional embeddings [33, 62, 145], link analysis using PageRank [90, 144], graph analysis and spectral clustering [120], and data compression [102, 107]. Nudge shows the possibility of performing these tasks on large secret-shared matrices, data sets, or graphs—without letting any party see the underlying data.

Limitations. Nudge comes with three main limitations. First, Nudge requires three servers and tolerates the semi-honest compromise of one. Second, Nudge's goal is to map user ratings into personalized recommendations; it relies on other means (e.g., Apple's private relay [18], Tor [64], or cryptographic private information retrieval [52]) to let users fetch data items in a private way.

Finally, Nudge's privacy property guarantees that no party learns *anything more* about users' behaviors than the final, aggregate recommender model encodes. As is inherent in any collaborative-filtering system, where users' inputs influence other users' outputs, the recommender model produced by Nudge reveals some information about user behaviors in a way that we precisely define in Section 3. As a standard (if imperfect) mitigation, we show how to make Nudge's recommender model and recommendations differentially private with respect to any user's data [67, 131].

2 Background: Matrix factorization

The goal of a recommender system is to give *users* recommendations for *items*. On a content-delivery website, items could be content to fetch; on a social network, items could be posts; on a restaurant-review app, items could be restaurants.

At the start of the recommendation process, each user holds *ratings* for a subset of the items. The interpretation of these ratings depends on the application: e.g., on a news site, a user's rating for item i could be "1" if the user has viewed article i, and "0" otherwise. For a restaurant-review app, the rating could be the user's 1–5 "star" rating for each restaurant.

A recommender system has two steps: training and serving recommendations. Training executes periodically; some fast-moving applications (e.g., Facebook ads [98]) retrain daily, while others (e.g., Spotify [69], VertexAI's recommenders for commerce [9]) retrain weekly. The resulting recommendations may be pushed to users' devices once training completes.

Matrix factorization. The algorithm at the core of our private recommender system is matrix factorization, which has been a key part of recommendation engines used at Netflix [113],

Yelp [25], Twitter [4], Spotify [3], and YouTube [56]. At the highest level, the matrix-factorization technique aims to explain users' ratings on a large collection of items by extracting a small number of latent "factors" that model each user and each item and capture the relationships between them [91]. The version of matrix factorization that we use is closely related to truncated singular value decomposition.

Even as neural-network-based recommenders grow popular [56, 97, 170], in many settings well-tuned matrix factorization techniques can still provide comparable quality and are appealingly simple, robust, and cheap to deploy [16, 75, 153, 154].

Training input: A matrix-factorization-based recommender system takes as input a user-item-interaction matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$, where m is the number of users, n is the number of items, and the entry at position (i, j) holds user i's rating for item j. (The entry is zero if user i did not rate item j.) The algorithm is parameterized by an embedding dimension $d \ll \min(m, n)$.

Training output: Matrix factorization computes a low-rank approximation of the matrix $\mathbf{U}$. More precisely, it outputs a matrix of d-dimensional user embeddings $\mathbf{A} \in \mathbb{R}^{m \times d}$ and a matrix of d-dimensional item embeddings $\mathbf{B} \in \mathbb{R}^{d \times n}$ such that $\mathbf{U} \approx \mathbf{A} \cdot \mathbf{B}$. The algorithm finds matrices $\mathbf{A}$ and $\mathbf{B}$ that minimize the Frobenius norm $\|\mathbf{U} - \mathbf{A} \cdot \mathbf{B}\|_F$ (i.e., the sum of the squared entries of matrix $\mathbf{U} - \mathbf{A} \cdot \mathbf{B}$). This can be computed in two steps: first compute a suitable matrix $\mathbf{B}$ that is row-orthonormal, then derive the associated matrix $\mathbf{A}$ as $\mathbf{A} := \mathbf{U} \cdot \mathbf{B}^\top$.

This output has a simple interpretation: the d-dimensional user and item embeddings model each user's preferences and each item's characteristics in terms of d shared latent factors.

Serving recommendations: To generate recommendations for user i, the system outputs the items that it estimates the user would rate the highest using its low-rank approximation $\mathbf{U} \approx \mathbf{A} \cdot \mathbf{B}$, and that user i has not already seen. In particular, denoting the rows of the user-embedding matrix $\mathbf{A}$ as $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)} \in \mathbb{R}^{1 \times d}$, the recommender outputs the items corresponding to the largest entries of $\mathbf{a}^{(i)} \cdot \mathbf{B}$ to user i.

When the matrix $\mathbf{B}$ is orthonormal (as described above), the matrix $\mathbf{A}$ is equal to $\mathbf{U} \cdot \mathbf{B}^\top$ and so this output is equivalent to finding the largest entries of $\mathbf{u}^{(i)} \cdot \mathbf{B}^\top \mathbf{B}$, where $\mathbf{u}^{(i)} \in \mathbb{R}^{1 \times n}$ is the ith row of the original matrix $\mathbf{U}$. This view has two appealing properties: first, the recommendations provided to user i depend only on its own ratings (i.e., vector $\mathbf{u}^{(i)}$) and on the learned item embeddings $\mathbf{B}$—that is, user i learns *no more* information about other users' ratings than is encoded in the item embeddings $\mathbf{B}$ [131]. Second, to generate recommendations for users who join after training completed (or who have since rated new items), it suffices to gather their ratings and use the already-computed item embeddings $\mathbf{B}$.

3 System design

A Nudge deployment consists of three infrastructure providers and many users and items. We refer to the infrastructure providers as "servers," though each provider might run on many physical machines. Much as in recent deployments of private telemetry [17, 70], we envision that, in practice, these Nudge servers could be run by one or multiple companies together with non-profit or governmental entities. A real-world example is the Prio deployment of exposure notification: CDC and ISRG were the non-colluding entities [2, 55].

3.1 Goals and threat model

The Nudge users and servers interact to aggregate all users' ratings with two goals: (1) training a recommender model that produces *useful* item recommendations for each user and (2) protecting the users' *privacy*—i.e., hiding which items each user rated and what its ratings are from the Nudge servers and from other users, in a cryptographic sense.

A Nudge deployment with m users and n items is parameterized by an integer $d \ll \min(m, n)$, representing the dimension of embeddings output by the training process. The Nudge servers then jointly implement the following functionality in a secure multiparty computation:

1. From each user $i \in [m]$, receive a rating vector $\mathbf{u}^{(i)} \in \mathbb{R}^n$. Assemble the m rating vectors into a matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$.
2. Set $(\mathbf{A} \in \mathbb{R}^{m \times d}, \mathbf{B} \in \mathbb{R}^{d \times n}) \leftarrow \mathsf{ApproxFactor}(\mathbf{U})$, where $\mathsf{ApproxFactor}$ is an approximate matrix-factorization algorithm (described in Section 5).
3. *Outputs:* Send the item embeddings $\mathbf{B}$ (i.e., the learned "model") to each of the three Nudge servers. Send the recommendation vector $\mathbf{u}^{(i)} \cdot \mathbf{B}^\top \mathbf{B}$ to user i.
4. *Additional leakage:* For each user $i \in [m]$, send each of the Nudge servers the number of non-zero entries (but not their location or values) in user i's rating vector $\mathbf{u}^{(i)}$.

As we discuss in Section 5, in reality, our multiparty computation represents real numbers with fixed-precision values.

Security goal: Protect user privacy against *one* honest-but-curious server colluding with malicious users. Against one honest-but-curious server and any number of malicious clients, Nudge should reveal *nothing* about the secret matrix $\mathbf{U}$ of users' ratings, apart from (1) what the learned item embeddings $\mathbf{B}$ reveal, and (2) how many items each user rated. (We could eliminate the latter leakage by requiring all users to submit a constant number of ratings.) Intuitively, this reveals the characterization of items in terms of d latent factors.

In some settings, when combined with sufficient side information, these recommendations themselves can be too revealing [39]. As a standard mitigation against such attacks, we show in Section 9 how to augment Nudge to provide differential privacy to protect each users' ratings vector—that is, the learned item embeddings $\mathbf{B}$ are differentially private with respect to any user's ratings vector—at modest cost in recommendation quality [67, 131].

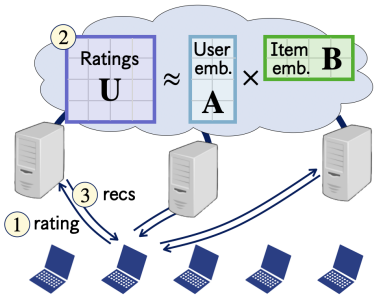

Figure 1: The three Nudge servers first collect secret-shared ratings from users, then run matrix factorization in a three-party computation to produce user and item embeddings, and finally serve secret-shared, personalized recommendations.

In addition, we show in Appendix A that the learned item embeddings $\mathbf{B}$ satisfy another desirable property: they are "anonymous" [55, Definition 2], in the sense that each Nudge server's view remains identical when any two users (with the same number of ratings) shuffle their rating vectors.

Security extension: Input validation. In some applications, the servers may want to enforce application-defined validity checks on the user-submitted rating vectors to ensure that no user can have an outsize influence on the computed recommendations (e.g., checking that each user rates at most k items, with ratings in a pre-determined range). These checks again protect client privacy against one honest-but-curious server.

Non-goals. Nudge does not protect against a malicious server who deviates from the prescribed protocol; such a server could disrupt the correctness of Nudge's outputs, the privacy of Nudge's users, or the availability of Nudge's service. Nudge also does not protect against two or more servers who collude. Finally, Nudge does not hide metadata: the servers learn when each user sends its messages.

Remark 3.1 (Differences to federated learning). Nudge differs from federated learning [130] in two ways: first, Nudge does not leak intermediate model versions to users or servers. Second, the Nudge servers train the recommendation model without involving users: users only need to submit their ratings and, later on, fetch their recommendations. At the same time, Nudge splits trust across three servers, while some federated-learning systems only require one or two [26, 129, 152].

3.2 Protocol flow

The Nudge protocol (Figure 1) takes place in three steps.

Step 1: Private data collection. Each user i succinctly secret shares their rating vector $\mathbf{u}^{(i)} \in \mathbb{R}^n$ and sends one share to each server. As we discuss in Section 3.1, the Nudge servers can run checks between the three of them to verify that all client submissions are well-formed.

Step 2: Private matrix factorization. Once the servers all hold a secret share of the matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$, they run the three-party matrix-factorization protocol, described in Section 5, to factor $\mathbf{U}$ into a secret-shared matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$ of user embeddings and a cleartext matrix $\mathbf{B} \in \mathbb{R}^{d \times n}$ of item embeddings.

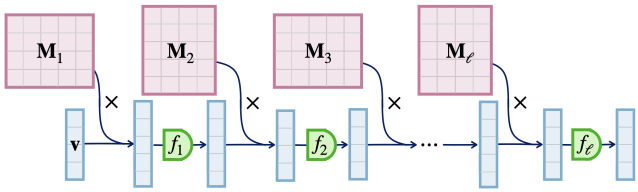

Figure 2: A matrix-vector program with functions $f_1, \ldots, f_\ell$, evaluated on input matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$, and input vector $\mathbf{v}$.

Step 3: Privately fetching recommendations. The recommendation scores for user i are just the values $\mathbf{a}^{(i)} \cdot \mathbf{B}$, where $\mathbf{a}^{(i)}$ is the ith user embedding, stored in the ith row of $\mathbf{A}$. For each client, the Nudge servers can compute secret shares of user i's recommendation scores and forward these shares to the user, who reconstructs the scores in plaintext.

Security analysis. Security of the overall system relies on the security of our new three-party matrix-factorization protocol, which we prove in Theorem 5.1.

4 Secure computation primitives

In this section, we introduce the specific multi-party computation protocols Nudge uses.

Our protocols work in the three-party honest-majority setting with an honest-but-curious adversary. That is, each party correctly executes the protocol, and we protect against an adversary that can compromise the state of a single party. We define security of a multiparty protocol using the standard real/ideal paradigm [122]. We assume that all parties have access to a cryptographic pseudorandom function (PRF) and that each pair of parties holds pairwise shared secret keys [20]. We refer to this setting as the "3PC with PRF" model.

Notation: 2-out-of-3 replicated secret sharing. A secret-sharing scheme makes it possible to split a value $x \in \mathcal{R}$ into shares, where each share individually hides x but together the shares encode x. A replicated sharing of x is a tuple of pairs $((x_1, x_2), (x_0, x_2), (x_0, x_1))$, where $x_0 + x_1 + x_2 = x$ and party i holds the ith pair. We denote a replicated sharing of x as $[\![x]\!]$.

We say that a function $f : \mathcal{R}^d \to \mathcal{R}^d$, on ring $\mathcal{R}$ and integer dimension $d \geq 1$, has a *3PC protocol* if three parties holding replicated shares $[\![\mathbf{v}]\!]$ of a vector $\mathbf{v} \in \mathcal{R}^d$ can compute replicated shares $[\![f(\mathbf{v})]\!]$, without learning anything about $\mathbf{v}$. Appendix B gives precise correctness and security definitions.

To measure computation costs over a ring $\mathcal{R}$, we count $\mathcal{R}$ operations and PRF invocations, each mapping a λ-bit seed to an element in $\mathcal{R}$, as having unit cost. For efficiency, we work with rings $\mathcal{R}$ of the form $\mathbb{Z}_{2^b}$ with integer bit length $b \geq 1$.

Function	Communication	Computation	Rounds
Any degree-two func. with m monomials	$3db$	$O(md)$	1
SIMD operation on each entry of the vector			
Zero-test [36]	$2db \cdot (\lambda+2)$	$\widetilde{O}(db)$	1
Integer compare [36]	$2db \cdot (\lambda+2b+2)$	$\widetilde{O}(db)$	1
Truncate t bits (§4.3)	$2dt \cdot (\lambda+4)+10db$	$\widetilde{O}(db)$	3
Normalize vector to have unit L_2-norm			
Approx. (§4.3)	$O(db\lambda+\lambda b^2+b^3)$	$O(db+b^2)$	$O(1)$

Table 3: Functions with constant-round 3PC protocols, mapping vectors in $\mathcal{R}^d$ to vectors in $\mathcal{R}^d$ on ring $\mathcal{R}=\mathbb{Z}_{2^b}$ with bit length b, dimension $d \geq 1$, and PRF-seed length λ. We hide logarithmic factors (in b and λ) in the notation $\widetilde{O}(\cdot)$. We highlight the new protocols from Section 4.3 in yellow.

4.1 Matrix-vector programs

We first define *matrix-vector programs*, which capture a class of computations that are particularly amenable to execution in a three-party computation (illustrated in Figure 2). We do not claim this model as a top-level contribution—our goal is to give a clean abstraction in which we can express our matrix-factorization algorithm.

Definition 4.1 (Matrix-vector program)**.** A *matrix-vector program* $\mathcal{P}$ is defined with respect to a ring $\mathcal{R}$ and a function class $\mathcal{F}$, that consists of a set of functions $f_1, \ldots, f_\ell \in \mathcal{F}$ mapping vectors in $\mathcal{R}$ to vectors in $\mathcal{R}$. The program $\mathcal{P}$ takes as input a vector $\mathbf{v} \in \mathcal{R}^d$ and a set of matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$ over $\mathcal{R}$. We define the program's output to be the vector, whose dimension is defined implicitly by the fs and $\mathbf{M}$s, computed as:

$$\mathcal{P}(\mathbf{v}, \mathbf{M}_1, \ldots, \mathbf{M}_\ell) := f_\ell\left(\mathbf{M}_\ell \cdot \ldots \cdot f_2\left(\mathbf{M}_2 \cdot f_1\left(\mathbf{M}_1 \mathbf{v}\right)\right)\right).$$

Looking ahead, in Section 5 we will cast iterative algorithms for approximate matrix decomposition [91] as matrix-vector programs. This matrix-vector-program abstraction may be useful in secure computation settings beyond the scope of this work; for example, evaluating a secret-shared neural network on secret-shared inputs (e.g., in private model fine-tuning); training a secret-shared network using gradient descent; computing the steady-state of a private Markov chain; or running belief propagation algorithms on secret-shared graphs.

4.2 Synthesizing prior work: Three-party computation of matrix-vector programs

The following theorem synthesizes multiple ideas in related work [20, 37, 134] for evaluating matrix-vector programs in a 3PC with concrete efficiency. It shows that, if a matrix-vector program is defined with respect to a set $\mathcal{F}$ of 3PC-friendly functions, it is possible to evaluate the program in a 3PC with communication *sublinear* in the input size. In particular, the parties' communication scales only with the largest size of an intermediate vector, not with the size of the input matrices. The number of rounds of interaction scales with the number of input matrices. When the matrices are large enough that the matrix-vector products dominate the overall compute, the per-party work in the 3PC protocol is only 3× larger than evaluating cleartext matrix-vector products of the same size.

Theorem 4.2 (3PC for matrix-vector programs)**.** *Let $\mathcal{F}$ be the class of functions that have a constant-round 3PC protocol in the 3PC with PRF model. Let $\mathcal{P}$ be a matrix-vector program over function class $\mathcal{F}$ and ring $\mathcal{R}=\mathbb{Z}_{2^b}$ with bit length $b \geq 1$, consisting of functions $f_1, \ldots, f_\ell \in \mathcal{F}$ where each $f_j : \mathcal{R}^{d_j} \to \mathcal{R}^{d_j}$. Moreover, let the 3PC for computing f_j use $s_j \in \mathbb{N}$ bits of communication and $w_j \in \mathbb{N}$ computation.*

Then, three parties holding replicated shares of the inputs to $\mathcal{P}$ can securely compute replicated shares of the output of $\mathcal{P}$ in the 3PC with PRF model, using $O(\ell)$ rounds of interaction, $\sum_{i=1}^{\ell}(6bd_i+s_i)$ total bits of communication, and $\sum_{i=1}^{\ell}(6d_{i-1}d_i+8d_i+w_i)$ total operations on each party.

Proof idea. We prove Theorem 4.2 in Appendix B. The idea, used heavily in prior work [20, 134], is that three parties can non-interactively evaluate matrix-vector products on secret-shared data using properties of replicated sharing. □

In Table 3, we list the 3PC-friendly non-linear functions that we will need for our matrix-factorization algorithm, including new protocols that we describe next. Our new protocols build on function secret sharing [36, 37], which leverages PRFs to evaluate zero-test and comparison gates on secret-shared inputs with a single round of interaction (see Table 3). There are many other 3PC-friendly non-linear functions in the literature [12, 34, 87–89, 104, 156, 158, 163, 167, 173].

4.3 3PC protocols for non-linear functions

We now describe new protocols in the 3PC-with-PRF model for fixed-point truncation and vector normalization.

The need for these particular functions stems from a mismatch between 3PC techniques, which operate over a ring $\mathcal{R}=\mathbb{Z}_{2^b}$, and data mining, statistics, and machine-learning computations, which operate over the reals. Standard practice bridges this gap using fixed-point arithmetic [42, 80, 143]: given a number of fractional bits of precision $t < b$, represent the real value v as the ring element $\lfloor v \cdot 2^t \rfloor \in \mathcal{R}$. As long as the data values never grow too large (i.e., do not "wrap around" the modulus 2^b), adding two real values is just addition in $\mathcal{R}$. Multiplying two real values requires a multiplication in $\mathcal{R}$, then a truncation by t bits to return to the correct data format. As a result, the efficiency of two non-linear operations on secret-shared data is paramount: *truncation*, which follows every multiplication gate, and *normalization*, which prevents the represented data values from growing too large.

Truncating a secret-shared value. A long line of work investigates truncation on secret-shared data [34, 42, 72, 94, 134, 135, 169, 175, 177] with various functionalities and, in some cases, leakage [118]. We perform deterministic truncation with sign extensions and no leakage [34], assuming one bit of slack [59, 72]: that is, on input $v \in [-2^{b-2}, 2^{b-2})$, we evaluate the arithmetic-right-shift function $\mathsf{Trunc}_t(v) := (v \gg t)$.

Our technique builds on the three-round, three-party approach of Jawalkar et al. [104, Theorem 6]. At a high level, we implement truncation by dividing the share $[\![v]\!]$ by 2^t, then

1. using an integer comparison, shown in Table 3, to handle any erroneous carries in the lowest-order bits [34] and
2. using the one bit of slack to prevent any erroneous carries the highest-order bits [72].

Under the hood in step (1), our protocol performs a degree-two computation, that in turn allows the integer comparison to output as few bits as possible. As a result, when truncating by $t < b$ bits, our communication scales as $2t(\lambda+4)+10b$, i.e., dominated by the term λt, while prior state-of-the-art truncation scales as $2b \cdot (\lambda+6) - 6t$, i.e., dominated by the term λb [104]. In Nudge where $\lambda = b = 128$ and $t = 20$, this is a $\approx b/t = 6\times$ improvement in communication. We give our full protocol in Appendix C; applying it to a vector gives:

Lemma 4.3 (3PC for truncation)**.** *In the 3PC-with-PRF model over ring $\mathcal{R} = \mathbb{Z}_{2^b}$, with $t \in [0, b-1]$ bits to truncate and λ-bit PRF seeds, there exists a secure 3PC protocol for the function $\mathsf{Trunc}_t : \mathcal{R}^d \to \mathcal{R}^d$ using three rounds, $2dt \cdot (\lambda+4) + 10db$ bits of communication, and $O(db \log t)$ total operations.*

Normalizing a secret-shared vector. L2-Normalization in a 3PC requires mapping shares of a vector $[\![\mathbf{v}]\!]$ to shares of the normalized vector $[\![\mathbf{v}/\|\mathbf{v}\|]\!]$, which points in the same direction but has unit norm. To this end, the three parties must compute the scaling factor $[\![1/\|\mathbf{v}\|]\!]$. Three parties can compute the squared norm $[\![\|\mathbf{v}\|^2]\!]$ via a degree-two function, but computing its "inverse-square-root" is challenging.

Prior works take one of two routes: the first uses exact lookup tables [166, 167] or more granular piecewise approximations [34, 89, 163] in a way that is constant-round, but where the compute grows *exponentially* with the precision required—i.e., it is $2^{O(b)}$ on bit length b. These approaches have only been used on bit length $b \in \{16, 64\}$ [89, 163]. In contrast, to run with many users, Nudge uses $b = 128$. The second route resorts to iterative numerical methods that guarantee any desired level of accuracy [114], but must be seeded with a reasonable starting point [42, 51, 112, 127, 150, 169]. Finding this starting point is tricky: prior works use $O(b)$ rounds [22], large look-up tables [150], or leaky protocols that reveal the bit length of $\|\mathbf{v}\|^2$ (or a blinded version of it to achieve statistical security) [51, 169].

We show a simple way to handle the starting-point issue in the 3PC setting that is single-round, has low communication, and has no extra leakage: using function secret sharing to compute the "most significant non-zero bit" [34, Theorem 14] of $[\![\|\mathbf{v}\|^2]\!]$. (Under the hood, this just invokes $b+1$ simultaneous integer comparisons, shown in Table 3.) A linear combination then gives shares of the approximation $[\![x]\!] = [\![2^{-\lfloor \log \|\mathbf{v}\|^2 \rfloor/2}]\!] \approx [\![1/\|\mathbf{v}\|]\!]$, which we refine, as in prior works, with the standard Newton-Raphson iteration [42, 51, 112, 169]. This method evaluates a degree-three polynomial that doubles the number of correct digits in $[\![x]\!]$ each time; it can be run for a constant number of steps (as done by Nudge) for approximate results, or for $O(\log b)$ steps for full accuracy [114]. We define the function $\mathsf{ApproxNormalize}$ and detail a 3PC protocol for it in Appendix C, proving:

Lemma 4.4 (3PC for approximate normalization)**.** *In the 3PC with PRF model over a ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with λ-bit PRF seeds, there exists a secure 3PC protocol for the function $\mathsf{ApproxNormalize} : \mathcal{R}^d \to \mathcal{R}^d$ in $O(1)$ rounds, $O(db\lambda + \lambda b^2 + b^3)$ bits of communication, and $O(db + b^2)$ operations.*

5 Private matrix factorization

Next, we apply the three-party computation framework of Section 4 to design a protocol for private matrix factorization.

Recall from Section 2 that a matrix-factorization algorithm takes in a matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ and a dimension $d \le \min(m, n)$ and then outputs a pair of matrices $\mathbf{A} \in \mathbb{R}^{m \times d}$ and $\mathbf{B} \in \mathbb{R}^{d \times n}$ that factor $\mathbf{U} \approx \mathbf{A} \cdot \mathbf{B}$. In Section 5.1, we describe how to use power iteration to factor a matrix. In Section 5.2, we show how to run this algorithm with repeated invocations of a matrix-vector program, letting us instantiate it in the 3PC framework. The proof sketch in Section 5.2 then gives:

Theorem 5.1 (3PC for matrix factorization)**.** *In the 3PC with PRF model, on seed length $\lambda \in \mathbb{N}$, ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with bit length $b \ge 1$, and public parameter $\ell \in \mathbb{N}$ denoting the number of iterations of the power-iteration algorithm, for any $m, n \in \mathbb{N}$ and $d \le \min(m, n)$, three parties that hold secret shares of a matrix $[\![\mathbf{U}]\!] \in \mathcal{R}^{m \times n}$ can securely evaluate the power-iteration algorithm to compute a secret-shared matrix $[\![\mathbf{A}]\!] \in \mathcal{R}^{m \times d}$ and a cleartext matrix $\mathbf{B} \in \mathcal{R}^{d \times n}$ in $O(d\ell)$ rounds, $O\left(\lambda d\ell b \cdot (m+n) + \lambda d\ell b^2 + d\ell b^3\right)$ communication, and $O\left(d\ell mn + d\ell b \cdot (m+n) + d\ell b^2\right)$ operations.*

5.1 Background: power iteration

We compute matrix factorizations using a classic algorithm from the 1920s: power iteration, also known as "vector iteration" or the "power method" [82, 132, 161]. We give pseudocode for our repeated application of power iteration in Figure 4. At its core, this algorithm leverages the linear-algebra fact that, to find a pair of matrices $\mathbf{A} \in \mathbb{R}^{m \times d}$ and $\mathbf{B} \in \mathbb{R}^{d \times n}$

Algorithm: Matrix factorization via power iteration.
Parameterized by $\ell \geq 1$ iterations and dimension $d \geq 1$.

// Compute $\mathbf{A}, \mathbf{B}$ *to be a rank-d approximation of* $\mathbf{U}$

$\mathsf{ApproxFactor}(\mathbf{U} \in \mathbb{R}^{m \times n}) \to \mathbf{A} \in \mathbb{R}^{m \times d}, \mathbf{B} \in \mathbb{R}^{d \times n}$

- Initialize $\mathbf{B}$ to be the all-zeros matrix in $\mathbb{R}^{d \times n}$.
- For i from 1 to d:
  - Initialize $\mathbf{v} \in \mathbb{R}^n$ to a random vector.
    Run $\mathbf{v} := \mathsf{Normalize}(\mathsf{SetOrthogonal}(\mathbf{v}, \mathbf{B}))$.
  - For j from 1 to ℓ:
    - Update $\mathbf{v} := \mathsf{Mul}(\mathbf{U}^\top, \mathsf{Mul}(\mathbf{U}, \mathbf{v}))$.
    - Update $\mathbf{v} := \mathsf{SetOrthogonal}(\mathbf{v}, \mathbf{B})$.
    - Update $\mathbf{v} := \mathsf{Normalize}(\mathbf{v})$.
  - Set the ith row of $\mathbf{B}$ equal to $\mathbf{v}$.
- Compute $\mathbf{A} := \mathbf{U} \cdot \mathbf{B}^\top$. Output $\mathbf{A}$, $\mathbf{B}$.

// Set $\mathbf{v}$ *orthogonal to all rows in* $\mathbf{B}$ *(Gram-Schmidt)*

$\mathsf{SetOrthogonal}(\mathbf{v} \in \mathbb{R}^n, \mathbf{B} \in \mathbb{R}^{d \times n}) \to \mathbf{v}' \in \mathbb{R}^n$

- For i from 1 to d:
  - Let $\mathbf{b} \in \mathbb{R}^n$ be the ith row of $\mathbf{B}$.
  - Set $\mathbf{v} := \mathbf{v} - \langle \mathbf{v}, \mathbf{b} \rangle \cdot \mathbf{b}$.
- Output $\mathbf{v}$.

Figure 4: Matrix factorization via power iteration [132].

that minimize the Frobenius norm $\|\mathbf{U} - \mathbf{A} \cdot \mathbf{B}\|_F$, it suffices to set the rows of $\mathbf{B}$ to be the top-d eigenvectors with the largest eigenvalues of the square matrix $\mathbf{U}^\top\mathbf{U}$. After this, finding the associated $\mathbf{A}$ is a linear operation: $\mathbf{A} := \mathbf{U} \cdot \mathbf{B}^\top$.

Finding dominant eigenvectors. To find the eigenvectors with the largest eigenvalues, the algorithm in Figure 4 extracts eigenvectors one at a time, in order of decreasing eigenvalue. To do so, the algorithm applies the square matrix $\mathbf{M} := \mathbf{U}^\top\mathbf{U}$ to itself many times. The eigenvector with the largest eigenvalue of $\mathbf{M}$ is its "direction of maximum stretch". For a large enough power $\ell \geq 1$ and a random vector $\mathbf{v}$, computing the vector $\mathbf{v}' := \mathbf{M}^\ell \cdot \mathbf{v}$ stretches $\mathbf{v}$ the most in the direction of the dominant eigenvector, until $\mathbf{v}'$ essentially equals matrix $\mathbf{M}$'s dominant eigenvector. (During this computation, we normalize the vector at each step to avoid overflow/underflow.)

After finding an eigenvector, the algorithm starts with a vector orthogonal to the eigenvector(s) previously recovered. It then repeats the powering step to extract the next eigenvector. Powering a random vector $\mathbf{v}$, that is orthogonal to all already-computed eigenvectors, ensures that $\mathbf{v}$ will not get stretched in the direction of the already-computed vectors—so, this recovers the eigenvector with the next largest eigenvalue.[1] When using fixed-point arithmetic, we re-orthogonalize the vectors at every step.

Remark 5.2 (Why power iteration?). Power iteration has a number of appealing properties: it has few hyperparameters to tune; it is relatively numerically stable [91]; it is robust to (even adversarially generated) noise [92]; it can be made differentially private [67, 131], as we discuss in Section 9; and there are theoretical guarantees on its convergence [157]. For example, letting γ be the ratio between the matrix $\mathbf{M}$'s largest and second-largest eigenvalues, power iteration converges to a vector that is within L_2-distance ϵ from the top eigenvector in $\ell = O(\log(n/\epsilon)/\gamma)$ iterations [137].

5.2 Power iteration as a matrix-vector program

Proof sketch of Theorem 5.1. We securely evaluate power iteration across three parties that hold shares of the matrix $[\![\mathbf{U}]\!]$ to be factored. At the end of the protocol, each party learns secret shares of the matrix $[\![\mathbf{A}]\!]$ and the cleartext matrix $\mathbf{B}$.

To do so, we cast the powering step in Figure 4 as a matrix-vector program (Definition 4.1) over the ring $\mathcal{R} = \mathbb{Z}_{2^b}$ and the function class $\mathcal{F}$ containing three functions:

- $\mathsf{normalize}(\mathbf{v})$: L_2-normalize the vector $\mathbf{v}$,
- $\mathsf{truncate}_t(\mathbf{v})$: perform an arithmetic right shift (i.e., with sign extension) by t bits on each element of $\mathbf{v}$, and
- $\mathsf{orthogonalize}_{\mathbf{B}}(\mathbf{v})$: compute the component of $\mathbf{v}$ that is orthogonal to all vectors in $\mathbf{B}$.

At the end of each powering step, our parties reveal and learn the computed row of matrix $\mathbf{B}$ in the clear. This design leverages power iteration's structure: it iteratively computes a portion of the matrix $\mathbf{B}$ at each step. (In contrast, gradient descent iteratively refines the *entire* recommender model at each step.) So, the three parties can learn $\mathbf{B}$ in cleartext as it is computed: aiding the computation while incurring no additional leakage.

To complete our protocol and apply Theorem 4.2, we must specify three things: (1) how to embed the real numbers into the finite ring $\mathbb{Z}_{2^b}$, (2) how to write the inner loop of Figure 4 as a matrix-vector program, and (3) how to implement the functions in $\mathcal{F}$ in a concretely-efficient 3PC.

1. Embedding the reals in $\mathbb{Z}_{2^b}$. We use the standard fixed-point techniques [134] described in Section 4.3. As in prior work [19, 110], we perform the truncations needed after each multiplication lazily: we replace any occurrences of "truncate-then-add" by "add-then-truncate". If the bit length b has sufficient space, we instead truncate by $2t$ bits after every

[1] This process achieves the same functionality as "blocked power iteration" or "subspace iteration" [157, Section 5.1], but we unroll the blocks here to take advantage of the fact that, when we will run in a 3PC, each successive row of the matrix $\mathbf{B}$ will be revealed in cleartext.

two multiplications. With these optimizations, running power iteration only requires truncating vectors—never matrices.

2. Inner loop as a matrix-vector program. The inner loop of power iteration (Figure 4) has precisely the structure of a matrix-vector program: it alternates between (a) applying the matrices $\mathbf{U}^\top$ and $\mathbf{U}$ to a vector $\mathbf{v}$, and (b) truncating, orthogonalizing, and normalizing the intermediate vectors. In our 3PC, the honest-but-curious parties jointly sample a cleartext initial vector $\mathbf{v}$, from which they can locally derive $[\![\mathbf{v}]\!]$.

3. Implementing non-linear layers in $\mathcal{F}$. Beyond vector normalization and fixed-point truncation (described in Section 4.3), the only remaining operation is orthogonalization. Orthogonalization is a linear function of the shares of vector $[\![\mathbf{v}]\!]$ since each party knows the matrix $\mathbf{B}$ in cleartext; as such, it can be performed locally, without any communication.

6 Implementation

Our Nudge prototype consists of 7,296 lines of Go code, using C to implement the matrix-multiplication core, along with 2,736 lines to implement the function-secret-sharing tools (as per Section 4.3). Nudge uses AES as a cryptographic PRF. We open-source our implementation at `https://github.com/NudgeArtifact/private-recs`.

Collecting ratings from users. Each user secret shares each of its ratings using a distributed point function that outputs replicated shares [11, 36]. (Using distributed point functions here reveals to the servers how many items each user rated.)

The Nudge servers then run checks to verify that all submissions are well-formed. In theory, Nudge could check arbitrary predicates on users' submissions using zero-knowledge proofs on secret-shared data [30, 55]. As detailed in Appendix D, we instead implement a simple check that ensures that all secret-shared ratings lie in $\{0, 1\}$. This check follows prior work [10] and is almost free in our three-party setting: the Nudge servers check all submissions at once by exchanging $O(\lambda)$ bits, on security parameter λ. If the check fails, the servers can binary search to identify malformed submissions.

Concrete parameters. Nudge takes the bit length of the ring $b = 128$, the number of fractional bits $t = 20$, and the seed length to a cryptographic PRF $\lambda = 128$. During normalization, Nudge increases the precision to 40 fractional bits and performs 5 iterations of Newton-Raphson approximation.

Picking hyperparameters for power iteration. The embedding dimension d should be set using application-dependent context. Following common practice on the data sets we evaluate on, we run Nudge with d in the range 8–50 [75]. The larger d, the more accurate the recommendations, but also the more user information they encode. The number of power iterations ℓ (Section 5.1) is a fixed constant that trades off between quality and cost: the larger ℓ, the more accurate the matrix factorization, but the more expensive it is to compute. By default, Nudge takes $\ell = 10$. (In our evaluation, we experiment with various choices of ℓ on the validation set and see that ℓ in the range 5–100 perform comparably.)

7 Evaluation

In our evaluation, we answer the following questions:

- Section 7.1: How good are Nudge's recommendations, compared to non-private recommendation algorithms?
- Section 7.2: What is the cost of Nudge's private matrix factorization, compared to prior private approaches?
- Section 7.3: How do Nudge's costs scale to large collections of items and users?
- Section 7.4: What is the end-to-end performance of serving private recommendations in Nudge?

Experimental setup. We run each Nudge server on a memory-optimized `r7a.48xlarge` AWS instance with 192 cores and 1,536 GB of RAM (costing $14.6064 per hour). We run on these large-RAM machines because of the computation's memory demands: the secret-shared matrix is over 200 GB on the Netflix dataset and over 1.3 TB on the Yelp dataset. On the smaller MovieLens data sets, we instead run each server on a `r7a.large` instance (2 cores, 16 GB of RAM) or a `r7a.4xlarge` instance (16 cores, 128 GB of RAM). Between the Nudge servers, we simulate two network settings: a local area network (LAN), with 50 GBps of bandwidth and a 0.25 ms RTT, and a wide area network (WAN), with 5 GBps of bandwidth and a 10 ms RTT. We count the cost of AWS data transfer as $0.01 per GB. Finally, we run a user on a `r7a.medium` instance (1 core, 8 GB of RAM). To measure throughput, we run many users on a `r7a.4xlarge` instance (16 cores, 128 GiB of RAM). We simulate the link between a user and the Nudge servers as 100 Mbps with a 50 ms RTT.

We measure Nudge's quality and performance on the Netflix data set [7], released as part of the Netflix prize competition (2006–2009). Following the recommendations literature [75, 119], we preprocess the data set to keep users that rated at least 5 items and to binarize the ratings (i.e., ratings on a scale of 1–5 are mapped to "1" if they are four or larger, and "0" otherwise). To compare to prior work, we also measure Nudge's costs on the smaller MovieLens data sets [5]. Finally, we report Nudge's costs on matrices whose sizes match a range of large-scale recommender data sets described in Section 7.3.

7.1 Nudge's recommendation quality

In Figure 5, we compare the quality of the recommendations produced by Nudge and a suite of standard recommender algorithms (run in cleartext). These algorithms include

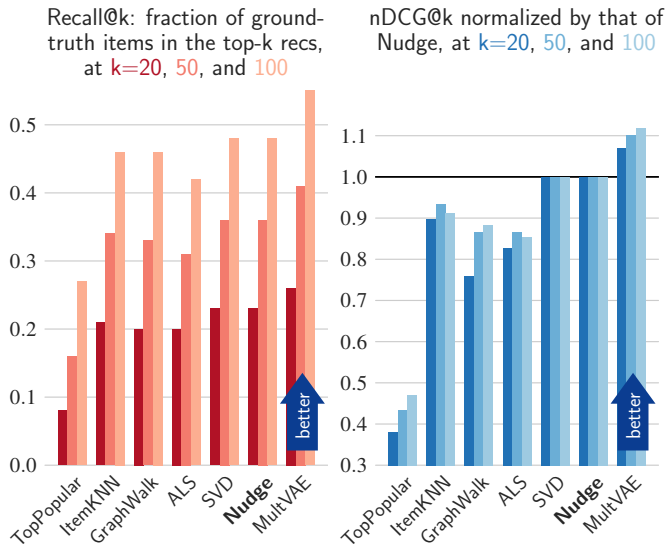

Figure 5: Nudge's recommendation quality on the Netflix data set [7] matches that of the cleartext matrix-factorization algorithms (ALS, SVD), exceeds the textbook recommender algorithms (TopPopular, ItemKNN, GraphWalk), and is within 12% of a deep-neural-network approach (MultVAE). We report numbers besides Nudge's from the survey of Dacrema et al. [75]; their SVD/ALS baselines use embedding dimension $d = 55/d = 146$, while Nudge uses $d = 50$.

- three textbook approaches: *TopPopular*, which recommends the most popular items [58], *ItemKNN*, a nearest-neighbor search on the items [159], and *GraphWalk*, a random walk on the user-item-interaction graph [54],
- two matrix-factorization techniques: *ALS*, matrix factorization via alternating least squares [100], and *SVD*, the classic singular value decomposition [58], and
- a recent, neural-network-based approach: *MultVAE*, which uses variational autoencoders [119].

Experiment. To evaluate recommender quality, we run the following experiment, common in the recommendations literature [75]: we hold out a fraction of a user's ratings (i.e., "1"s in the binarized matrix), and then measure how often each algorithm recommends the held-out items that the user actually rated. In more detail, we split the data set into training, validation, and test sets by selecting 10,000 users for each of the validation and test sets and, for each such user, including 80% of their ratings in the training set and using the remaining 20% as "ground truth" ratings in the validation and test sets respectively, as per prior work [75, 119]. (All other users' ratings are added to the training set.) We report performance on the test set, measuring the following standard metrics:

- *Recall at k*: what fraction of the "ground-truth" ratings (i.e., items that the user rated highly, but this was held out from training) are in the top-k recommended items?
- *Normalized discounted cumulative gain (nDCG) at k*: how many of the "ground-truth" ratings are surfaced in the top-k recommended items, weighted by each item's rank in the top-k list (with logarithmic discount) and normalized by the best-possible ranking [103]?

For each metric, higher scores indicate better recommendations and the maximum score is 1.

Results. Figure 5 shows that Nudge's recommendation quality matches the matrix-factorization baselines and is within 12% of the deep-neural-network approach. In particular, at cutoff $k = 20$, the textbook algorithms achieve a nDCG score of 0.11–0.26, the matrix-factorization algorithms achieve a nDCG score of 0.24–0.29, and the neural algorithm achieves a nDCG score of 0.31. Nudge achieves a nDCG of 0.29. Moreover, at $k = 20$, Nudge's recall score is 0.23—that is, 23% of the held-out, "ground-truth" items appear in the list of top-20 recommendations produced by Nudge, on average.

These results show that Nudge's differences to a cleartext matrix-factorization algorithm (namely, performing a fixed number of steps of power iteration and using fixed-point arithmetic with deferred truncations and approximate normalizations) do not noticeably affect its recommendation quality.

Finally, we measure the quality of Nudge's recommender algorithm using 10-fold cross-validation (by partitioning the users into 10 folds and performing the evaluation experiment on each fold); each of Nudge's nDCG and recall scores at $k \in \{20, 50, 100\}$ remain the same as in Figure 5.

7.2 Nudge's private matrix factorization

In Table 6, we benchmark the main performance overhead in Nudge: private matrix factorization (Section 5). Asymptotically, for a fixed embedding dimension d and with m users and n items, Nudge's compute scales as $O(mn)$ and its communication scales as $O(m+n)$. Concretely, Nudge factors the full Netflix data set—which contains 76× more users, 4.5× more items, and 50× more ratings than the largest data set previously used for private recommendation—in 50 min of wall-clock time, with 40 GB of communication between the three Nudge servers, running on a local-area network. In total, this costs $37 on AWS. On a wide-area network, the wall-clock time increases to 1h 13 mins and the communication stays the same (costing $54 on AWS).

Nudge outperforms prior private-matrix-factorization systems and 3PC baselines, as we describe next.

Prior private recommender systems. A range of prior works perform private matrix factorization with semi-honest security: Nikolaenko et al. [142] use garbled circuits to run gradient descent between two non-colluding parties. The GraphSC system [139] parallelizes these garbled-circuits computations.

System	Non-colluding servers	Algorithm	Dim. d	Rounds	Server-to-server comm. (GB)	Runtime		
						LAN	WAN	Cores
Subset of MovieLens100K: 940 users, 40 movies, and 14,683 ratings.								
Nikolaenko et al.♣ [142]	2	gradient descent	-	1	1,330	29 hr	-	32
Nudge	3	power iteration	8	7,080	**0.1**	**2.8 sec**	**34.1 sec**	6
MovieLens100K: 943 users, 1,682 movies, and 100K ratings.								
PIRSONA♠ [166]	4	gradient descent	8	-	4.6	-	11.7 mins	32
Nudge	3	power iteration	8	7,080	**0.5**	**10.7 sec**	**51.8 sec**	6
MovieLens1M: 6,040 users, 3,883 movies, and 1 million ratings.								
GraphSC♣ [139]	2	gradient descent	10	1	78,125	est. 11 days	-	128
Nudge in MP-SPDZ [110]	3	power iteration	10	11,057,010	1.5	52.4 min	est. 1.3 days	48
Nudge	3	power iteration	10	9,150	**1.4**	**32.8 sec**	**1.8 min**	48
Netflix: 463,435 users, 17,769 movies, and 56 million ratings.								
Nudge	3	power iteration	50	75,750	**38.9**	**50.2 min**	**1h 13 min**	576

Table 6: Nudge runs faster and incurs less communication than prior systems for private matrix factorization. We bold the best performance on each data set, running Nudge on fewer cores than prior work. All systems provide semi-honest security. We report the performance and configurations specified in each paper, using - to indicate unknown values. We scale the costs for Nikolaenko el al. [142] and PIRSONA [166], who report a single iteration of gradient descent, linearly according to the fact that gradient descent requires 10 iterations on the subset of MovieLens100K (as used by Kim et al. [111]) and 20 iterations on the full MovieLens100K (as used by recommender libraries [101]). Column 8 gives the total number of cores used across all parties. ♣We estimate communication based on microbenchmarks and the linear scaling described. ♠We compute PIRSONA's communication using Table 1. PIRSONA's WAN setup diverges from ours: it runs on 4 servers in different availability zones.

Unlike Nudge, these protocols require only *two* non-colluding servers, use a single round of interaction, and have asymptotically better computation (that scales with the number of ratings, rather than the product of users and items mn). At the same time, however, they incur asymptotically larger server-to-server communication costs (that scale with the number of ratings, rather than the sum of users and items $m+n$). In concrete terms, these systems have over *four* orders of magnitude higher communication and compute costs than Nudge.

In particular, Nikolaenko et al. [142] run on a subset of the MovieLens100K data set: they report 29 hrs (using 32 cores) and over 1 TB of server-to-server communication to run gradient descent; Nudge factors the same data set in 2.8 sec (using 6 cores) and 0.1 GB. GraphSC [139] run on the MovieLens1M data set: they report 11 days of compute (using 128 cores) and over 76 TB of communication; Nudge factors the same data set in 33 seconds (using 48 cores) and 1.4 GB. We expect very recent, state-of-the-art techniques for SIMD garbling [85] to shave an order of magnitude off of GraphSC's computation and communication, though they do not evaluate on the matrix-factorization task. Other new techniques for communication-efficient garbling [125] could shrink the communication, but would further increase the compute.

Again using two non-colluding servers, Kim et al. [111] improve the cost of garbled-circuit-based matrix factorization with homomorphic encryption, at the cost of leaking which users rated which items. (The authors propose to obfuscate this leakage by having each user log some fake ratings.) They require 15 mins (using 6 cores) and 5.7 GB on the same subset of MovieLens100K. Nudge remains 320× faster while incurring 57× less communication.

Finally, PIRSONA [166] performs private matrix factorization by running gradient descent in a bespoke multi-party computation between *four* semi-honest, non-colluding parties. Like Nudge, PIRSONA's communication scales as $O(m+n)$. PIRSONA reports 11.7 mins (using 32 cores) and 4.6 GB to run full gradient descent on the MovieLens100K data set on a WAN setup. Nudge factors the same data set in 51.8 seconds (using 6 cores) and 0.5 GB on our WAN setup—saving 8× in communication and requiring one fewer non-colluding party.

Baseline: Three-party arithmetic computation. To evaluate the impact of Nudge's three-party protocols for truncation and normalization (Section 4.3), we re-implement and benchmark Nudge's same power iteration algorithm, written as a matrix-vector program, using the MP-SPDZ framework [110]. Like Nudge, MP-SPDZ performs secure computation using replicated secret sharing over a power-of-two ring; however, where Nudge uses function secret sharing for non-linear op-

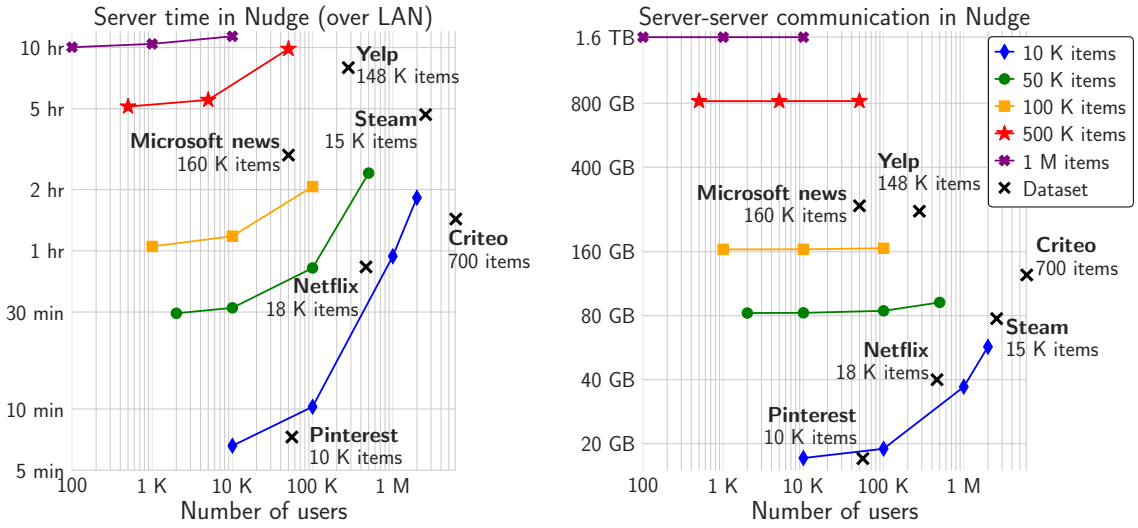

Figure 7: Nudge privately factors matrices with millions of users or items. We measure Nudge's wall-clock time and server-to-server communication on matrices of increasing size, keeping the number of items constant (up to 1 TB in size after secret sharing, which translates to $mn \leq 3.4 \times 10^{10}$ with m users and n items). For context, we show Nudge's performance on matrices of the size of public recommendation data sets. In each case, Nudge produces embeddings of dimension $d = 50$, requires 75,750 rounds, and runs on three 192-core machines. For large m or n, we measure costs with $d = 5$ or $d = 2$ and extrapolate linearly.

erations, MP-SPDZ instead converts to binary secret shares (as we disallow leaky probabilistic truncations). As a result, on the MovieLens1M data set, the MP-SPDZ baseline has approximately the same communication cost as Nudge, but it requires three orders of magnitude more rounds of interaction. This increased interaction translates to 95× slower runtime in a LAN setting or 1040× slower runtime in a WAN setting.

Baseline: No privacy. Dacrema et al. [75] report the cost of running non-private SVD as 50.53 s on the Netflix data set (compared to Nudge's 50 mins); they use one `p3.2xlarge` AWS instance with 8 vCPUs, 30 GB of RAM, and one GPU. As a result, their non-private matrix factorization of Netflix costs roughly $0.05 on AWS, while Nudge's costs $37.

7.3 Scaling Nudge to larger data sets

Popular recommender systems handle many millions to billions of users/items. In Figure 7, we scale Nudge's private matrix factorization to data sets with millions of users or items. As mentioned earlier, given m users and n items, Nudge's compute costs grow as $O(mn)$ and its communication costs grow as $O(m+n)$. The constant factors within these asymptotics play an important role: because of lazy truncation (Section 5.2), the constant in front of the "m" term in the communication is dwarfed by that in front of the "n" term, causing the concrete communication to be dominated by the number of items n as soon as n grows sufficiently large.

In addition to synthetic benchmarks, we measure Nudge's runtime and server-to-server communication costs when privately factoring matrices whose sizes match those of public recommendation data sets. On a LAN setup, we measure that:

- Nudge factors a Criteo ads data set [6] (with $m = 6.1$M users and $n = 700$ ad campaigns) in 1.5 h and 125 GB.
- Nudge factors a Yelp data set [8] (with $m = 279$K users and $n = 148$K businesses) in 8h and 248 GB.
- Nudge factors a Steam data set [146] (with $m = 2.5$M users and $n = 15$K video games) in 4.7 hr and 78 GB.
- Nudge factors a Pinterest data set [68] (with $m = 55$K users and $n = 10$K photos) in under 10 mins and 20 GB.
- Nudge factors a Microsoft-News data set [172] (with $m = 50$K users and $n = 160$K articles) in 3 hr and 263 GB.

7.4 Nudge's end-to-end performance

In Table 8, we break down the end-to-end costs of serving private recommendations with Nudge, both analytically and as measured on the Netflix data set.

With distributed point functions, the user-to-server communication to log a rating scales logarithmically with the number of items n (concretely, 694 bytes). Each of the three Nudge servers then performs $O(n \log n)$ operations to log the secret-shared submission, done in tens of milliseconds of latency and at a throughput of hundreds of submissions per second. The servers perform a one-time check that the final secret-shared matrix $\mathbf{U}$ is well-formed, which requires scanning over the matrix (done in 7.7 min) and exchanging 64 bytes. Finally,

	Analytical cost	As implemented
Logging a rating.		
User work	$O(\lambda \log n + b)$ ops	
User-server comm.	$2\log n(\lambda+8)+2b$ bits	694 bytes
Server work	$O(n \log n)$ ops	0.03 s
Server throughput		>270 queries/s
Checking that a matrix of ratings is well-formed.		
Server-server comm.	$4b$ bits	64 bytes
Server work	$O(mn)$ ops	7.7 min
Fetching recommendations.		
User-server comm.	$O(nb)$ bits	298 KB
User-perceived latency		0.38 s
Server work	$O(nd)$ ops	0.04 s
Server throughput		>110 queries/s

Table 8: Nudge's costs for data collection (step 1) and fetching recommendations (step 3) scale with the number of items n, dimension d, bit length b of the ring, and PRF seed length λ. We report concrete costs on the Netflix data set (Table 6).

fetching recommendations requires a user to download $O(n)$ bits from each server (concretely, 298 KB), which takes 0.38s. The servers again handle hundreds of requests per second.

8 Extension: Scaling up Nudge with dimensionality reduction

While Nudge can privately factor million-user data sets in hours, Nudge's compute grows as the product of users, items, and embedding dimension dmn—which becomes a bottleneck when applied to sufficiently massive data sets. (This issue does not befall cleartext matrix-factorization systems: they can take advantage of the matrix $\mathbf{U}$'s inherent sparsity, whereas the Nudge servers who only see secret shares of $[\![\mathbf{U}]\!]$ cannot.)

Nudge can overcome this scalability bottleneck with *dimensionality reduction* via random projection [123]. Johnson and Lindenstrauss [105] showed that projecting a matrix onto a random, low-dimensional sketch $\mathbf{S}$ approximately preserves the distances between its rows. Papadimitriou, Raghavan, Tamaki, and Vempala [145, Theorem 4] showed that compressing a matrix in this way before factoring it incurs only a small loss in accuracy: for an error parameter $0 < \epsilon < 0.5$, the two-step process of (a) projecting the matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ down to a smaller, sketched version $\hat{\mathbf{U}} := \mathbf{U} \cdot \mathbf{S} \in \mathbb{R}^{m \times s}$ with $s = \Omega(\log n/\epsilon^2)$, and then (b) factoring $\hat{\mathbf{U}}$, gives a $(1+\epsilon)$-approximation to factoring the original $\mathbf{U}$ (in Frobenius norm, with appropriately chosen embedding dimension). Random projection is a data-oblivious, linear computation, making it well suited for multi-party computation.

Applying this technique, Nudge's compute does *not* need to scale as $O(dmn)$. Rather, the three Nudge servers can jointly sample a sketch matrix $\mathbf{S} \in \mathbb{R}^{n \times s}$, locally apply it to their secret shares $[\![\mathbf{U}]\!]$ to obtain $[\![\hat{\mathbf{U}}]\!] := [\![\mathbf{U}]\!] \cdot \mathbf{S}$, and run their private matrix factorization on $[\![\hat{\mathbf{U}}]\!]$ from there. We describe this process in Appendix E. Crucially, though the servers know the sketch $\mathbf{S}$, applying it to the matrix $\mathbf{U}$ on the right side treats all users' data symmetrically (and thus does not affect Nudge's anonymity property). As a result, Nudge's compute can scale as $O(mn + dm \log n/\epsilon^2)$, or even $O(m\sqrt{n} + dm \log n/\epsilon^2)$ with more aggressive sparse sketching [117].

When the number of items n is sufficiently large, we expect dimensionality reduction to give substantial improvements: for example, on the Common Crawl data set with 364 M web pages [149], sketching with standard parameters [1] shrinks the dimension n = 364 M to down to $s \leq 17{,}000$. Running Nudge at this scale would be bottlenecked by the cost of performing two passes over the original matrix $\mathbf{U} \in \mathbb{R}^{m \times n}$ (one to sketch it down at the beginning, and one to recover the item embeddings at the end), not the time to perform private power iteration on the $20{,}000\times$ smaller matrix $\hat{\mathbf{U}} \in \mathbb{R}^{m \times s}$.

9 Extension: Enhancing Nudge's privacy protections

Recommendations with differential privacy. A sequence of works studies differentially private algorithms for principal component analysis [48, 67], low-rank matrix approximation [109], power iteration [23, 92, 93, 141], and matrix-factorization-based recommendations [76, 126, 131, 148, 162]. Differential privacy offers the guarantee that the learned item embeddings $\mathbf{B}$ cannot leak any one user's rating vector with high probability. This guarantee is complementary to the privacy property provided by Nudge (that no party learns anything about the user inputs *beyond* the item embeddings $\mathbf{B}$), and approaches to achieving it integrate neatly into Nudge.

In particular, Dwork et al. [67] show that, to achieve differential privacy, it suffices to add gaussian noise to each entry of the matrix $\mathbf{U}^\top\mathbf{U} \in \mathbb{R}^{n \times n}$. Following this approach, Nudge can provide differential privacy with two extra steps:

- After collecting ratings, the three Nudge servers normalize each row of the matrix $[\![\mathbf{U}]\!]$ (by dividing the ith row in $[\![\mathbf{U}]\!]$ by the number of ratings submitted by user i).
- Each pair of servers picks a matrix in $\mathbb{R}^{n \times n}$ with i.i.d. gaussian entries and splits it across all three servers with replicated secret sharing. (This requires no communication, using PRFs.) The servers sum these matrices to $[\![\mathbf{E}]\!] \in \mathbb{R}^{n \times n}$. At each step of power iteration, when the servers compute $\mathsf{Mul}([\![\mathbf{U}^\top]\!], \mathsf{Mul}([\![\mathbf{U}]\!], [\![\mathbf{v}]\!]))$, they add to this $\mathsf{Mul}([\![\mathbf{E}]\!], [\![\mathbf{v}]\!])$. This incurs no communication.

On the Netflix data set [7], we measure that power iteration extended in this way to provide (ϵ, δ)-differential privacy with $\epsilon = 1$ and $\delta = 2^{-40}$ achieves a nDCG@20 score of 0.24 (as

defined in Section 7.1). McSherry and Mironov similarly conclude that retrofitting recommender systems with differential privacy does not cause large degradations in quality [131].

Recommendations on aggregate data. To further prevent user-level information from being revealed via the item embeddings **B**, certain use cases of Nudge could compute recommendations over aggregate user data as opposed to individual user data. To give an example: Nudge can first aggregate users by zip code, and then provide private yet personalized recommendations of the form "users in area X often click on Y." This would allow a private search engine [99] to rank websites about local businesses, local governments, etc. higher if they are geographically relevant to a user. One benefit is that the dimension m would now correspond to the number of aggregated user "categories" (rather than the total number of users), letting Nudge factor a far smaller matrix. One limitation is that the servers must know which users to aggregate over, so this mapping cannot be based on highly sensitive data.

10 Discussion: Nudge applications

Nudge shows the potential viability of other applications:

Collaborative recommendations. With Nudge, multiple services could jointly compute recommendations without ever sharing user data. For example, Netflix, Hulu, and Disney+ could jointly train a recommendation model that leverages ratings across all three platforms. Or, LinkedIn and Facebook could compute friend recommendations across a joint version of their social networks. Or, Amazon and Instagram could compute recommendations across a user's joint purchase and watch history. In this setting, multiple non-colluding parties, who could each run a Nudge server, already exist.

Beyond recommendation. Nudge's core power-iteration algorithm computes a low-rank approximation of a secret-shared matrix. Low-rank matrix approximations form the computational backbone of data-mining tasks like principal component analysis [96], information-retrieval tasks like learning low-dimensional embeddings [62, 145] and autoencoders [24, 33], link-analysis tasks like PageRank [90, 144], graph-analysis tasks like spectral clustering [120], and data-compression tasks like matrix denoising [102, 107]. By expressing power iteration as a matrix-vector program and optimizing its low-level gates, Nudge shows that these applications could be executed on large, secret-shared matrices, data sets, or graphs.

11 Related Work

Multiple approaches preserve user privacy in recommender systems, with the goal of protecting the sensitive data needed to train recommendation models [138].

A first approach relies on trusted execution environments [13, 63] but is vulnerable to attacks on the underlying hardware [50, 84]. A second approach uses federated learning [130] to share only intermediate model parameters (not the original user data) with the recommender system, though an honest-but-curious server may still deduce user ratings [43,86,121,164]. Protocols combining federated learning with secure aggregation [26,129,152] address this leakage, but inherently involve all users in the many-round training process and reveal intermediate model versions to users and servers. A third approach uses cryptographic tools to ensure that no party can learn users' training data, as long as the two or more parties processing the data do not collude [40, 71, 111, 139, 142, 166]. As shown in Section 7.2, Nudge improves on the cost and scale of these cryptographic systems. Other work assumes pre-trained deep learning models exist, and shows how to privately serve recommendations given such public models [79].

Nudge, which at its core performs private matrix factorization, is related to systems for privacy-preserving principal component analysis [14, 29, 51, 73, 77]. The work of Cho et al. [51] works in the same setting as Nudge but offers different security guarantees: it uses leaky truncation protocols [42] and random projection in a way that breaks user anonymity [55]. Other works [14, 73, 77] factor a matrix that is sharded across a handful of parties (e.g., a few hospitals performing joint data analysis), allowing them to perform many computations locally. In contrast, Nudge factors a matrix that depends on data from potentially millions of users. Relative to the most performant private PCA system [77], Nudge runs on two orders of magnitude larger matrices.

Extensive research applies multi-party computation to private ML inference and training across two parties [41, 45, 74, 87, 88, 95, 104, 106, 133–135, 155, 171, 174], three parties [22,46,66,115,116,134,147,151,158,163,165,167–169], or more [47, 59, 115, 124]. Nudge diverges from these works in that, rather than running general-purpose gradient descent, it uses a simple algorithm from numerical linear algebra (Section 5.1). Nevertheless, Nudge shares many common techniques: using three parties to evaluate non-interactive matrix-vector products [28,49,116,134,165,166] and function secret sharing [36] for non-linear gates [12, 37, 87–89, 104, 156, 158, 163, 167, 173]. Nudge contributes new three-party protocols for truncation and normalization.

12 Conclusion

Nudge is a recommender system that provides strong privacy by splitting trust across three servers. We show how matrix factorization is well-suited to three-party computation, and contribute an efficient and scalable design and implementation. We leave open three questions: first, to provide malicious security; second, to run across more parties while tolerating the compromise of several of them; third, to serve private recommendations with only *two* non-colluding parties without harming Nudge's performance.

Acknowledgments. We thank our anonymous shepherd and the USENIX Security reviewers. We thank Sam Hopkins and Ainesh Bakshi for conversations about matrix decomposition, Elette Boyle and Mayank Rathee for conversations about three-party computation, David Wu for conversations about private PCA, and Elaine Shi, Xiao Wang, Kartik Nayak, and Adithya Vadapalli for conversations about related work. Nickolai Zeldovich, Mayank Rathee, Derek Leung, Ariel Szekely, and Ryan Lehmkuhl provided feedback on an early draft. This work was supported by gifts from Apple, Google, Meta, and NSF grants 2452708 and 1804222. Alexandra Henzinger was supported by the NSF GRFP under Grant No. 2141064; this work was done in part while visiting the Simons Institute for the Theory of Computing.

Ethical Considerations

The stakeholders in this work include the authors, as well as companies and organizations that may use Nudge to generate recommendations and users that may submit ratings to Nudge. We did not identify any significant potential harms to the authors during the research process. For users that may submit ratings to Nudge: Nudge provides stronger privacy properties than deployed cleartext recommendation algorithms, where the provider learns both the model and user ratings. One potential harm is that, in Nudge, some information about user ratings is revealed via the model, which is the output of Nudge's matrix-factorization protocol. We discuss how to mitigate this with differential privacy in Section 9. For companies or organizations deploying Nudge: Nudge's recommendation quality is not as good as that of a neural-network-based approach (Figure 5), the cost of training the model is higher than on cleartext data, and the protocol requires three non-colluding servers. However, companies and organizations do not have access to user ratings in the clear, which has the potential to reduce liability, which may be particularly important for some applications. Upon weighing the potential benefits and harms of our work using the principle of "beneficence" from the Menlo Report [65], we concluded that the benefits of completing the research outweighed the potential harms.

Open Science

Our full Nudge system code is open sourced at: `https://zenodo.org/records/17968761`. We evaluate Nudge on the released Netflix prize [7] and MovieLens [5] data sets. We also use the size of existing recommender data sets [6, 8, 68, 146, 172] to report Nudge's performance on matrices with the same dimensions.

References

[1] The johnson-lindenstrauss bound for embedding with random projections. `scikit-learn.org/stable/auto_examples/miscellaneous/plot_johnson_lindenstrauss_bound.html`.

[2] Prio server. github.com/divviup/prio-server.

[3] Spotify. `github.com/spotify/annoy`.

[4] Twitter's recommendation algorithm. `blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm`.

[5] Movielens 1m dataset. `grouplens.org/datasets/movielens/1m/`, 2003.

[6] Criteo dataset. `huggingface.co/datasets/criteo/criteo-attribution-dataset`, 2017.

[7] Netflix prize data. `www.kaggle.com/datasets/netflix-inc/netflix-prize-data`, 2019.

[8] Yelp dataset. `www.kaggle.com/datasets/yelp-dataset/yelp-dataset`, 2022.

[9] Vertexai search for commerce, manage models. https://docs.cloud.google.com/retail/docs/manage-models, 2025.

[10] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder–scalable, robust anonymous committed broadcast. In *ACM SIGSAC*, 2020.

[11] Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In *EUROCRYPT*, 2024.

[12] Amit Agarwal, Stanislav Peceny, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. Communication-efficient secure logistic regression. In *EuroS&P*, 2024.

[13] Esma Aïmeur, Gilles Brassard, José M Fernandez, and Flavien Serge Mani Onana. Alambic: a privacy-preserving recommender system for electronic commerce. *Int. J. Inf. Secur.*, 7(5), 2008.

[14] Mohammad Al-Rubaie, Pei-yuan Wu, J Morris Chang, and Sun-Yuan Kung. Privacy-preserving PCA on horizontally-partitioned data. In *IEEE DSC*, 2017.

[15] Bobby Allyn. Google to delete search data of millions who used 'incognito' mode, 2024. www.npr.org/2024/04/01/1242019127/google-incognito-mode-settlement-search-history.

[16] Vito Walter Anelli, Alejandro Bellogín, Tommaso Di Noia, Dietmar Jannach, and Claudio Pomo. Top-N recommendation algorithms: A quest for the state-of-the-art. In *ACM UMAP*, 2022.

[17] Apple and Google. Exposure notification privacy-preserving analytics (ENPA) white paper. 2021.

[18] Apple Inc. iCloud private relay overview. Technical report, Apple Inc., Cupertino, CA, December 2021.

[19] Toshinori Araki, Assi Barak, Jun Furukawa, Marcel Keller, Yehuda Lindell, Kazuma Ohara, and Hikaru Tsuchida. Generalizing the SPDZ compiler for other protocols. In *CCS*, 2018.

[20] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, 2016.

[21] Apple Developer Article. Getting up-to-date calling and blocking information for your app. developer.apple.com/documentation/identitylookup/getting-up-to-date-calling-and-blocking-information-for-your-app.

[22] Nuttapong Attrapadung, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Takahiro Matsuda, Ibuki Mishina, Hiraku Morita, and Jacob CN Schuldt. Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation. *arXiv:2106.02203*, 2021.

[23] Maria-Florina Balcan, Simon Shaolei Du, Yining Wang, and Adams Wei Yu. An improved gap-dependency analysis of the noisy power method. In *COLT*, 2016.

[24] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1), 1989.

[25] Shafi Bashar and Alex Gillmor. Scaling collaborative filtering with PySpark. engineeringblog.yelp.com/2018/05/scaling-collaborative-filtering-with-pyspark.html, May 2018.

[26] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Phillipp Schoppmann. Willow: Secure aggregation with one-shot clients. In *CRYPTO*, 2025.

[27] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*. 1988.

[28] Remco Bloemen, Bryan Gillespie, Daniel Kales, Philipp Sippl, and Roman Walch. Large-scale MPC: Scaling private iris code uniqueness checks to millions of users. *arXiv:2405.04463*, 2024.

[29] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Implementation and evaluation of an algorithm for cryptographically private principal component analysis on genomic data. *TCBB*, 15(5), 2018.

[30] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *CRYPTO*, 2019.

[31] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *IEEE S&P*, 2021.

[32] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Arithmetic sketching. In *CRYPTO*, 2023.

[33] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4), 1988.

[34] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *EUROCRYPT*, 2021.

[35] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*, 2015.

[36] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[37] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *TCC*, 2019.

[38] Justin Brickell and Vitaly Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *SIGKDD*, 2008.

[39] Joseph A Calandrino, Ann Kilzer, Arvind Narayanan, Edward W Felten, and Vitaly Shmatikov. "You might also like:" privacy risks of collaborative filtering. In *IEEE S&P*, 2011.

[40] John Canny. Collaborative filtering with privacy via factor analysis. In *SIGIR*, 2002.

[41] Sergiu Carpov, Kevin Deforth, Nicolas Gama, Mariya Georgieva, Dimitar Jetchev, Jonathan Katz, Iraklis Leontiadis, Mohsen Mohammadi, Abson Sae-Tang, and Marius Vuille. Manticore: Efficient framework for scalable secure multiparty computation protocols. *Cryptology ePrint Archive*, 2021.

[42] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *FC*, 2010.

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

[44] Kelvin Chan. Tiktok fined $600 million for china data transfers that broke eu privacy rules, 2025. apnews.com/article/tiktok-ireland-european-union-data-privacy-regulation-d386ec74becc716905d7f686d6a448e2.

[45] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: Programmable and efficient secure two-party computation for machine learning. In *EuroS&P*, 2019.

[46] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. Astra: High throughput 3PC over rings with application to secure prediction. In *CCSW*, 2019.

[47] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv:1912.02631*, 2019.

[48] Kamalika Chaudhuri, Anand D Sarwate, and Kaushik Sinha. A near-optimal algorithm for differentially-private principal components. *The Journal of Machine Learning Research*, 14(1), 2013.

[49] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In *ASIACRYPT*, 2020.

[50] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D Garcia. VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface. In *USENIX Sec*, 2021.

[51] Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 36(6), 2018.

[52] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[53] Federal Trade Commission. Google and youtube will pay record $170 million for alleged violations of children's privacy law, 2019. www.ftc.gov/news-events/news/press-releases/2019/09/google-youtube-will-pay-record-170-million-alleged-violations-childrens-privacy-law.

[54] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. Random walks in recommender systems: exact computation and simulations. In *WWW*, 2014.

[55] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, 2017.

[56] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, 2016.

[57] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, 2005.

[58] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, 2010.

[59] Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four: Honest-Majority Four-Party secure computation with malicious security. In *USENIX Sec*, 2021.

[60] Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In *EUROCRYPT*, 2017.

[61] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A private time-series database from function secret sharing. In *IEEE S&P*, 2022.

[62] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41(6), 1990.

[63] Akash Dhasade, Nevena Dresevic, Anne-Marie Kermarrec, and Rafael Pires. TEE-based decentralized recommender systems: The raw data sharing redemption. In *IPDPS*, 2022.

[64] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Sec*, 2004.

[65] David Dittrich and Erin Kenneally. Annoy. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research, 2012. `catalog.caida.org/paper/2012_menlo_report_actual_formatted`.

[66] Ye Dong, Chen Xiaojun, Weizhan Jing, Li Kaiyun, and Weiping Wang. Meteor: Improved secure 3-party neural network inference with reducing online communication costs. In *ACM Web Conference*, 2023.

[67] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze gauss: optimal bounds for privacy-preserving principal component analysis. In *STOC*, 2014.

[68] Travis Ebesu, Bin Shen, and Yi Fang. Collaborative memory network for recommendation systems. In *SIGIR*, 2018.

[69] Annie Edmundson. The rise (and lessons learned) of ML models to personalize content on home (part i). engineering.atspotify.com/2021/11/the-rise-and-lessons-learned-of-ml-models-to-personalize-content-on-home-part-i, 2021.

[70] Steven Englehardt. Next steps in privacy-preserving telemetry with prio. blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

[71] Zekeriya Erkin, Michael Beye, Thijs Veugen, and Reginald L Lagendijk. Efficiently computing private recommendations. In *ICASSP*, 2011.

[72] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In *CRYPTO*, 2020.

[73] Xiaoyu Fan, Guosai Wang, Kun Chen, Xu He, and Wei Xu. PPCA: Privacy-preserving principal component analysis using secure multiparty computation (MPC). *arXiv:2105.07612*, 2021.

[74] Jun Feng, Yefan Wu, Hong Sun, Shunli Zhang, and Debin Liu. Panther: Practical secure two-party neural network inference. *Trans. Info. For. Sec.*, 2025.

[75] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *TOIS*, 39(2), 2021.

[76] Arik Friedman, Shlomo Berkovsky, and Mohamed Ali Kaafar. A differential privacy framework for matrix factorization recommender systems. *UMUAI*, 26, 2016.

[77] David Froelicher, Hyunghoon Cho, Manaswitha Edupalli, Joao Sa Sousa, Jean-Philippe Bossuat, Apostolos Pyrgelis, Juan R Troncoso-Pastoriza, Bonnie Berger, and Jean-Pierre Hubaux. Scalable and privacy-preserving federated principal component analysis. In *IEEE S&P*, 2023.

[78] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. In *SIGKDD*, 2008.

[79] Karthik Garimella, Austin Ebel, Gabrielle De Micheli, and Brandon Reagen. HE-LRM: Encrypted deep learning recommendation models using fully homomorphic encryption. *arXiv:2506.18150*, 2025.

[80] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. In *PETS*, 2017.

[81] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[82] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

[83] Michael Greenacre, Patrick JF Groenen, Trevor Hastie, Alfonso Iodice d'Enza, Angelos Markos, and Elena Tuzhilina. Principal component analysis. *Nature Reviews Methods Primers*, 2(1), 2022.

[84] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of Rowhammer defenses. In *IEEE S&P*, 2018.

[85] Tianyao Gu, Afonso Tinoco, Sri Harish Govinda Rajan, and Elaine Shi. PicoGRAM: Practical garbled RAM from Decisional Diffie-Hellman. In *CRYPTO*, 2025.

[86] Xiao Guo, Xiang Li, Xiangyu Chang, Shusen Wang, and Zhihua Zhang. Fedpower: privacy-preserving distributed eigenspace estimation. *Machine Learning*, 113(11), 2024.

[87] Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. Shark: Actively secure inference using function secret sharing. In *IEEE S&P*, 2025.

[88] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure GPT inference with function secret sharing. *Cryptology ePrint Archive*, 2023.

[89] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. Llama: A low latency math library for secure inference. *Cryptology ePrint Archive*, 2022.

[90] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *WWW*, 2013.

[91] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2), 2011.

[92] Moritz Hardt and Eric Price. The noisy power method: A meta algorithm with applications. *Advances in neural information processing systems*, 27, 2014.

[93] Moritz Hardt and Aaron Roth. Beyond worst-case analysis in private singular vector computation. In *STOC*, 2013.

[94] Christopher Harth-Kitzerow, Ajith Suresh, and Georg Carle. Truncation untangled: Scaling fixed-point arithmetic for privacy-preserving machine learning to large models and datasets. *Cryptology ePrint Archive*, 2024.

[95] Christopher Harth-Kitzerow, Yongqin Wang, Rachit Rajat, Georg Carle, and Murali Annavaram. PIGEON: A high throughput framework for private inference of neural networks using secure multiparty computation. In *PETS*, 2025.

[96] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[97] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, 2017.

[98] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *AdKDD*, 2014.

[99] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with Tiptoe. In *SOSP*, 2023.

[100] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[101] Nicolas Hug. Surprise: A python library for recommender systems. *J. Open Source Softw.*, 5(52), 2020.

[102] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. Low-rank matrix completion using alternating minimization. In *STOC*, 2013.

[103] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM TOIS*, 20(4), 2002.

[104] Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: FSS-based secure training and inference with GPUs. In *IEEE S&P*, 2024.

[105] William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(1), 1984.

[106] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Sec*, 2018.

[107] Samruddhi Kahu and Reena Rahate. Image compression using singular value decomposition. *IJOART*, 2(8), 2013.

[108] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *JACM*, 61(1), 2014.

[109] Michael Kapralov and Kunal Talwar. On differentially private low rank approximation. In *SODA*, 2013.

[110] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *ACM SIGSAC*, 2020.

[111] Jinsu Kim, Dongyoung Koo, Yuna Kim, Hyunsoo Yoon, Junbum Shin, and Sungwook Kim. Efficient privacy-preserving matrix factorization for recommendation via fully homomorphic encryption. *TOPS*, 21(4), 2018.

[112] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[113] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[114] Stan Korzilius and Berry Schoenmakers. Divisions and square roots with tight error analysis from newton–raphson iteration in secure fixed-point arithmetic. *Cryptography*, 7(3), 2023.

[115] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. SWIFT: Super-fast and robust privacy-preserving machine learning. In *USENIX Sec*, 2021.

[116] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *IEEE S&P*, 2020.

[117] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *SIGKDD*, 2006.

[118] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. Efficient 3PC for binary circuits with application to Maliciously-Secure DNN inference. In *USENIX Sec*, 2023.

[119] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *WWW*, 2018.

[120] Frank Lin and William W Cohen. Power iteration clustering. 2010.

[121] Guanyu Lin, Feng Liang, Weike Pan, and Zhong Ming. Fedrec: Federated recommendation with explicit feedback. *IEEE Intelligent Systems*, 36(5):21–30, 2020.

[122] Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography.*, 2017.

[123] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2), 1995.

[124] Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multiparty computation protocols for machine learning in the honest-majority setting. In *USENIX Sec*, 2024.

[125] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: garbled circuits with 1 bit per gate. In *EUROCRYPT*, 2025.

[126] Ziqi Liu, Yu-Xiang Wang, and Alexander Smola. Fast differentially private matrix factorization. In *RecSys*, 2015.

[127] Wen-jie Lu, Yixuan Fang, Zhicong Huang, Cheng Hong, Chaochao Chen, Hunter Qu, Yajin Zhou, and Kui Ren. Faster secure multiparty computation of adaptive gradient descent. In *Workshop on Privacy-Preserving Machine Learning in Practice*, 2020.

[128] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning. In *IEEE S&P*.

[129] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *IEEE S&P*, 2023.

[130] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 2017.

[131] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *ACM SIGKDD*, 2009.

[132] RV Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM-Journal of Applied Mathematics and Mechanics*, 9(1), 1929.

[133] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference system for neural networks. In *USENIX Sec*, 2020.

[134] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *CCS*, 2018.

[135] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

[136] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against Intel SGX. In *IEEE S&P*, 2020.

[137] Christopher Musco. The power method and spectral methods for graph partitioning. `www.cs.princeton.edu/courses/archive/fall18/cos521/Lectures/lec14.pdf`, 2018.

[138] Arvind Narayanan and Vitaly Shmatikov. Robust deanonymization of large sparse datasets. In *IEEE S&P*, 2008.

[139] Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. GraphSC: Parallel secure computation made easy. In *IEEE S&P*, 2015.

[140] Lily Hay Newman. A new google+ blunder exposed data from 52.5 million users, 2018. www.wired.com/story/google-plus-bug-52-million-users-data-exposed/.

[141] Julien Nicolas, César Sabater, Mohamed Maouche, Sonia Ben Mokhtar, and Mark Coates. Differentially private and decentralized randomized power method. *arXiv:2411.01931*, 2024.

[142] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. Privacy-preserving matrix factorization. In *CCS*, 2013.

[143] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P*, 2013.

[144] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[145] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *PODS*, 1998.

[146] Apurva Pathak, Kshitiz Gupta, and Julian McAuley. Generating and personalizing bundle recommendations on steam. In *SIGIR*, 2017.

[147] Arpita Patra and Ajith Suresh. BLAZE: Blazing fast privacy-preserving machine learning. In *NDSS*, 2020.

[148] Huseyin Polat and Wenliang Du. SVD-based collaborative filtering with privacy. In *SAC*, 2005.

[149] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. arxiv.org/abs/1910.10683.

[150] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SiRNN: A math library for secure RNN inference. In *IEEE S&P*, 2021.

[151] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *ACM SIGSAC*, 2020.

[152] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *IEEE S&P*, 2023.

[153] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *RecSys*, 2020.

[154] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. Revisiting the performance of iALS on item recommendation benchmarks. In *RecSys*, 2022.

[155] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS*, 2018.

[156] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *arXiv:2006.04593*, 2020.

[157] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

[158] Manuel B. Santos, Dimitris Mouris, Mehmet Ugurbil, Stanislaw Jarecki, José Reis, Shubho Sengupta, and Miguel de Vega. Curl: Private LLMs through wavelet-encoded look-up tables. Cryptology ePrint Archive, Paper 2024/1127, 2024.

[159] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[160] Noam Scheiber and Mike Isaac. Facebook halts ad targeting cited in bias complaints, 2019. www.nytimes.com/2019/03/19/technology/facebook-discrimination-ads.html.

[161] Hans-Rudolf Schwarz. *Methode der finiten Elemente*, volume 47. Springer-Verlag, 2013.

[162] Yilin Shen and Hongxia Jin. Privacy-preserving personalized recommendation: An instance-based approach via differential privacy. In *ICDM*, 2014.

[163] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast $(2+1)$-PC for $\mathbb{Z}_{2^n}$ via (2, 2)-DPFs. In *CSS*, 2023.

[164] Zehua Sun, Yonghui Xu, Yong Liu, Wei He, Lanju Kong, Fangzhao Wu, Yali Jiang, and Lizhen Cui. A survey on federated recommendation systems. *IEEE Trans. Neural Netw.*, 36(1), 2024.

[165] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CrypTGPU: Fast privacy-preserving machine learning on the gpu. In *IEEE S&P*, 2021.

[166] Adithya Vadapalli, Fattaneh Bayatbabolghani, and Ryan Henry. You may also like... privacy: Recommendation systems meet PIR. *PETS*, 2021.

[167] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *PETS*, 2022.

[168] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *PETS*, 2019.

[169] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. In *PETS*, 2021.

[170] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *SIGKDD*, 2015.

[171] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A GPU platform for secure computation. In *USENIX Sec*, 2022.

[172] Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, et al. Mind: A large-scale dataset for news recommendation. In *Annual meeting of the association for computational linguistics*, 2020.

[173] Peng Yang, Zoe Lin Jiang, Shiqi Gao, Hongxiao Wang, Jun Zhou, Yangyiye Jin, Siu-Ming Yiu, and Junbin Fang. FssNN: communication-efficient secure neural network training via function secret sharing. *Cryptology ePrint Archive*, 2023.

[174] Peng Yang, Zoe Lin Jiang, Shiqi Gao, Hongxiao Wang, Jun Zhou, Yangyiye Jin, Siu-Ming Yiu, and Junbin Fang. Communication-efficient secure neural network via key-reduced distributed comparison function. In *ProvSec*, 2025.

[175] Martin Zbudila, Erik Pohle, Aysajan Abidin, and Bart Preneel. Master: maliciously secure truncation for replicated secret sharing without pre-processing. In *CANS*, 2024.

[176] Shijie Zhang, Wei Yuan, and Hongzhi Yin. Comprehensive privacy analysis on federated recommender system against attribute inference attacks. *IEEE Trans. on Knowledge and Data Engineering*, 36(3), 2023.

[177] Lijing Zhou, Qingrui Song, Su Zhang, Ziyu Wang, Xianggui Wang, and Yong Li. Bicoptor 2.0: Addressing challenges in probabilistic truncation for enhanced privacy-preserving machine learning. *arXiv:2309.04909*, 2023.

A Additional material on system design

Corrigan-Gibbs and Boneh [55] give a formal definition of the notion of *anonymity* in a data-collection protocol: intuitively, a function $f(\mathbf{x}_1, \ldots, \mathbf{x}_m)$, computed on m users' inputs $\mathbf{x}_1, \ldots, \mathbf{x}_m$, is "anonymous" if, for any pair of honest users i and j, the servers running the data collection cannot distinguish the world where user i submits vector $\mathbf{x}_i$ and user j submits vector $\mathbf{x}_j$, from another world where user i submits vector $\mathbf{x}_j$ and user j submits vector $\mathbf{x}_i$.

Approximate matrix factorization computed via the power-iteration algorithm (Section 5) satisfies anonymity: effectively, the algorithm

1. builds the matrix $\mathbf{U}$ whose i^{th} row is the vector of ratings $\mathbf{x}_i \in \mathbb{R}^n$ submitted by user i,
2. builds the covariance matrix $\mathbf{U}^\top \cdot \mathbf{U}$, and
3. applies this covariance matrix to a known vector many times and outputs the result.

Crucially, the covariance matrix $\mathbf{U}^\top \cdot \mathbf{U}$ is anonymous: its entry at position (a, b) is $\sum_{i \in [m]} x_{i,a} \cdot x_{i,b}$. So, if two honest users who contributed the *same* number of ratings shuffle their inputs, (1) the view of the servers during data collection remains the same, and (2) the covariance matrix $\mathbf{U}^\top \cdot \mathbf{U}$ remains identical.

B Additional material on 3PC for matrix-vector programs

B.1 Background and notation

We describe key primitives in the 3PC-with-PRF model that work over the ring $\mathcal{R} = \mathbb{Z}_{2^b}$ for some integer bit length $b \geq 1$. That is, $\mathcal{R}$ represents the integers $\{0, \ldots, 2^b - 1\}$ with addition and multiplication modulo 2^b. All arithmetic is over $\mathcal{R}$.

Secret sharing. A secret-sharing scheme lets a user split a value $x \in \mathcal{R}$ into shares, where each share individually hides x but together the shares make it possible to reconstruct x.

3-out-of-3 additive secret sharing. An additive sharing of x is a tuple of values (x_0, x_1, x_2), where $x_0 + x_1 + x_2 = x$ and party i holds x_i. We denote an additive sharing of x as $[x]$.

2-out-of-3 replicated secret sharing. A replicated sharing of x is a tuple of pairs $((x_1, x_2), (x_0, x_2), (x_0, x_1))$, where $x_0 + x_1 + x_2 = x$ and party i holds the ith pair. We denote a replicated sharing of x as $[\![x]\!]$.

In both cases, to sample a fresh secret sharing of x, it suffices to sample $x_0, x_1 \xleftarrow{R} \mathcal{R}$ and set $x_2 := x - x_0 - x_1$.

Computing on secret-shared data. Additive and replicated sharings are linear: given shares of two values $x, y \in \mathcal{R}$, the parties can locally compute shares of the sum $(x + y) \in \mathcal{R}$ under the same secret-sharing scheme. Replicated secret sharing also allows for one non-interactive multiplication, whose result is shared additively [20, 57]: that is, from $[\![x]\!]$ and $[\![y]\!]$, the parties can locally compute $[xy]$. Extending these notions to matrix operations (by sharing and computing on each value element-wise) gives a way to evaluate matrix-vector products on secret-shared data without any interaction:

Fact B.1 (Non-interactive multiplication [134, Section 5.2])**.** For dimensions $d, d' \in \mathbb{N}$, any matrix $\mathbf{M} \in \mathcal{R}^{d \times d'}$, and any vector $\mathbf{v} \in \mathcal{R}^{d'}$, three parties can compute

$$[\mathbf{Mv}] := \mathsf{MatrixMul}([\![\mathbf{M}]\!], [\![\mathbf{v}]\!]).$$

Evaluating MatrixMul requires no interaction; each party performs $6dd'$ multiplications and additions in $\mathcal{R}$.

To convert from an additive sharing of a value to a replicated sharing of the same value, the parties must interact:

Fact B.2 (Interactive re-sharing [20, Section 2.3])**.** For any dimension $d \in \mathbb{N}$ and vector $\mathbf{v} \in \mathcal{R}^d$, three parties can compute

$$[\![\mathbf{v}]\!] := \mathsf{Reshare}([\mathbf{v}]).$$

Evaluating Reshare requires:

- each party to perform $2d$ PRF calls and $2d$ adds in $\mathcal{R}$,
- party i to send a vector in $\mathcal{R}^d$ to party $(i+1) \bmod 3$, for $i \in \{0, 1, 2\}$.

Under the hood, interactive resharing works as follows. For $i, j \in \{0,1,2\}$, letting $\mathbf{v}_i$ be the additive share of vector $\mathbf{v}$ held by party i and letting $\mathbf{r}_{ij}$ be a random vector held by parties i and j (derived by applying a PRF to their pairwise shared randomness, as per Section 4):

1. Party 0 sends $\mathbf{v}_0 + \mathbf{r}_{01} + \mathbf{r}_{02}$ to party 2.
   Party 1 sends $\mathbf{v}_1 - \mathbf{r}_{01} - \mathbf{r}_{12}$ to party 0.
   Party 2 sends $\mathbf{v}_2 + \mathbf{r}_{12} - \mathbf{r}_{02}$ to party 1.

2. Now, the parties can build replicated shares of $\mathbf{v}$, where party 0 holds $(\mathbf{y}_1, \mathbf{y}_2)$, party 1 holds $(\mathbf{y}_0, \mathbf{y}_2)$, party 2 holds $(\mathbf{y}_0, \mathbf{y}_1)$, and $\mathbf{y}_0 + \mathbf{y}_1 + \mathbf{y}_2 = \mathbf{v}$, for:

   - $\mathbf{y}_0 = \mathbf{v}_2 + \mathbf{r}_{12} - \mathbf{r}_{02}$,
   - $\mathbf{y}_1 = \mathbf{v}_0 + \mathbf{r}_{01} + \mathbf{r}_{02}$, and
   - $\mathbf{y}_2 = \mathbf{v}_1 - \mathbf{r}_{01} - \mathbf{r}_{12}$.

By chaining Facts B.1 and B.2, three parties can evaluate any function on secret-shared data [20, 27]. However, a long line of work has shown that custom protocols can evaluate certain non-linear functions with far better concrete efficiency [34, 36, 37, 60, 104, 158, 163, 167]. We will invoke these special-purpose protocols as subroutines. To abstract away their details, we use the notion of a "3PC protocol":

Definition B.3 (3PC protocol)**.** In the 3PC-with-PRF model, on any integer dimension $d \geq 1$, we define a *3PC protocol* for a deterministic function $g : \mathcal{R}^d \to \mathcal{R}^d$ to be a constant-round, interactive protocol between three parties, which we write as the following algorithm:

$$\mathsf{3PCEval}(1^\lambda, g, [\![\mathbf{x}]\!]) \to [\![\mathbf{y}]\!],$$

which takes as input a security parameter λ, the function g, and replicated shares of any input vector $\mathbf{x} \in \mathcal{R}^d$ and produces replicated shares of the output vector $\mathbf{y} \in \mathcal{R}^d$. We require a 3PC protocol to meet two properties:

1. *Correctness*: For any dimension $d \geq 1$ and point $\mathbf{x} \in \mathcal{R}^d$,

   $$\Pr\Big([\![g(\mathbf{x})]\!] = \mathsf{3PCEval}(1^\lambda, g, [\![\mathbf{x}]\!]) \Big) = 1,$$

   where $[\![g(\mathbf{x})]\!]$ denotes any replicated sharing of $g(\mathbf{x}) \in \mathcal{R}^d$ and the probability is taken over the randomness of the 3PCEval multi-party computation.

2. *Security*: For each party $i \in \{0,1,2\}$, there exists an efficient, randomized algorithm $\mathsf{Sim}_{g,i}$ that takes as input the security parameter λ and party i's replicated share of the input $[\![\mathbf{x}]\!]_i$ and then outputs party i's view in the protocol. That is, the following two distributions are computationally indistinguishable:

   - $\{\mathsf{Sim}_{g,i}(1^\lambda, [\![\mathbf{x}]\!]_i)\}$ and
   - $\{\mathsf{View}_{g,i}(\lambda, [\![\mathbf{x}]\!])\}$, which denotes party i's view in an execution of the $\mathsf{3PCEval}(1^\lambda, g, [\![\mathbf{x}]\!])$ protocol.

Protocol: 3PC for matrix-vector programs. Parameterized by security parameter $\lambda \in \mathbb{N}$ and matrix-vector program $\mathcal{P}$ with parameters $b, \ell \in \mathbb{N}$ where $\mathcal{R} = \mathbb{Z}_{2^b}$, dimensions $d_0, d_1, \ldots, d_\ell \in \mathbb{N}$, and functions $f_1, \ldots, f_\ell$, where $f_j : \mathcal{R}^{d_j} \to \mathcal{R}^{d_j}$, that each have 3PC protocols.

Inputs. Each of the three parties holds replicated secret shares of the input: $[\![\mathbf{M}_i]\!]$ for $j \in \{1, \ldots, \ell\}$ and $[\![\mathbf{v}^{(0)}]\!]$.

Protocol. For $j = 1, \ldots, \ell$:

1. Each party locally computes $[\mathbf{u}^{(j)}] := \mathsf{MatrixMul}([\![\mathbf{M}_j]\!], [\![\mathbf{v}^{(j-1)}]\!])$.
2. The parties communicate to obtain $[\![\mathbf{u}^{(j)}]\!] := \mathsf{Reshare}([\mathbf{u}^{(j)}])$.
3. The parties interact to evaluate the 3PC protocol for function f_j at evaluation point $\mathbf{u}^{(j)}$: $[\![\mathbf{v}^{(j)}]\!] := \mathsf{3PCEval}(1^\lambda, f_j, [\![\mathbf{u}^{(j)}]\!])$.

Outputs. Each party holds its share of $[\![\mathbf{v}^{(\ell)}]\!]$.

Figure 9: A 3PC protocol for matrix-vector programs.

B.2 3PC protocol for matrix-vector programs

We give the 3PC protocol that proves Theorem 4.2 in Figure 9. Throughout, let $\lambda \in \mathbb{N}$ be a computational security parameter and let $\mathcal{R} = \mathbb{Z}_{2^b}$ be a ring with bit length $b \geq 1$. For a value v that is secret shared with replicated sharing, we write party i's share as $[\![v]\!]_i$. For a value v that is secret shared with additive sharing, we write party i's share interchangeably as $[v]_i$ or v_i.

Proof of Theorem 4.2. The correctness and efficiency of Theorem 4.2 follow by inspection. Next, we show that the protocol is secure. To show that three parties can securely evaluate a given function, we rely on the simulation paradigm [122]: we define an "ideal functionality" where the parties learn nothing beyond the output of the function. We then show that each party cannot distinguish a "real-world" execution of our protocol from an "ideal-world" execution, in which a trusted party runs the ideal functionality and passes its output to a simulator, who generates each party's view.

To this end, we define the following ideal functionality for a matrix-vector program:

Ideal functionality for matrix-vector programs. On a matrix-vector program $\mathcal{P}$ with parameters $b, \ell, d_0, \ldots, d_\ell \in \mathbb{N}$ where $\mathcal{R} = \mathbb{Z}_{2^b}$, the ideal functionality takes as input a set of replicated shares from each party. Each party's replicated share is ℓ matrices $[\![\mathbf{M}_1]\!] \in \mathcal{R}^{d_1 \times d_0}, \ldots, [\![\mathbf{M}_\ell]\!] \in \mathcal{R}^{d_\ell \times d_{\ell-1}}$, and a vector $[\![\mathbf{v}]\!] \in \mathcal{R}^{d_0}$. Then, the functionality:

1. reconstructs the shares,

2. evaluates the program $\mathbf{v}' := \mathcal{P}(\mathbf{M}_1, \ldots, \mathbf{M}_\ell, \mathbf{v})$ and
3. outputs replicated secret shares of the result $\mathbf{v}' \in \mathcal{R}^{d_\ell}$ to each of the three parties.

Let $\mathcal{P}$ be a matrix-vector program with parameters $b, \ell, d_0, \ldots, d_\ell \in \mathbb{N}$ where $\mathcal{R} = \mathbb{Z}_{2^b}$ and functions $f_1, \ldots, f_\ell$. Consider replicated shares of any inputs $\mathbf{M}_1, \ldots, \mathbf{M}_\ell$, and $\mathbf{v}$ to the matrix-vector program. We consider a probabilistic, polynomial-time adversary $\mathcal{A}$ that statically corrupts one of the three parties and obtains its view during the protocol execution. The corrupted party is honest-but-curious, and still follows the protocol.

We work in the hybrid model [122], using the secure protocol Reshare described in prior works to convert additive shares to replicated shares [20, 61, 165] and the secure protocol $\mathsf{3PCEval}(1^\lambda, f_j, \cdot)$ for each non-linear function f_j for $j \in \{1, \ldots, \ell\}$. We construct a simulator for party $i \in \{0, 1, 2\}$ that, given access to party i's inputs (i.e., party i's replicated shares $[\![\mathbf{M}_1]\!]_i, \ldots, [\![\mathbf{M}_\ell]\!]_i$, and $[\![\mathbf{v}]\!]_i$) and to the ideal functionality's outputs (i.e., the replicated share of the result $[\![\mathbf{v}']\!]_i$), simulates party i's view in the protocol.

Simulator: Given the matrix-vector program $\mathcal{P}$, the identity of the party i, party i's inputs $[\![\mathbf{M}_1]\!]_i, \ldots, [\![\mathbf{M}_\ell]\!]_i$, the simulator

1. locally computes party i's share $,[\mathbf{u}^{(j)}]_i$ using MatrixMul
2. computes party i's share $[\![\mathbf{u}^{(j)}]\!]_i$ using the simulator for Reshare, and
3. computes party i's share $[\![\mathbf{v}^{(j)}]\!]_i$ using the simulator for the 3PC gate $\mathsf{3PCEval}(1^\lambda, f_j, [\![\mathbf{u}^{(j)}]\!])$.

We now show that the corrupted party i cannot distinguish between a "real-world" execution of the protocol in Figure 9 and an "ideal-world" execution of the protocol, where a trusted party runs the ideal functionality and then the simulator constructs party i's view. We do so via a sequence of hybrids:

- Hybrid 0 is the real-world execution of the protocol. This world is identical to the previous one.
- In Hybrid 1, we replace each invocation of the Reshare protocol with a call to its simulator (as done in step 2 above). When Reshare is instantiated with a secure PRF, the adversary's advantage in distinguishing these two worlds is negligible in λ [20].
- In Hybrid 2, we replace each invocation of the 3PCEval protocol with a call to the protocol's simulator (as done in step 3 above). When 3PCEval is instantiated with secure 3PC protocols, the adversary's advantage in distinguishing these two worlds is negligible in λ.
- Hybrid 3 is the ideal-world execution of the protocol. This world is identical to the previous one.

This completes the argument.

C Additional material on truncation and normalization

We use standard fixed-point techniques [80, 110, 134, 143], mapping elements in $\mathcal{R} = \mathbb{Z}_{2^b}$ to signed integers [34, Section 6.3]. Given a number of fractional bits of precision $f < b$, we represent the real value v as the ring element $\lfloor v \cdot 2^f \rfloor \in \mathcal{R}$. We choose the bit lengths b and f so that the computation never wraps around—that is, all represented values fall in the range $[-2^{b-f-1}, 2^{b-f-1})$.

We first give a definition of function secret sharing (FSS).

Syntax of FSS with additive output shares. Consider any two rings $\mathcal{R}, \mathcal{R}'$. A FSS scheme [35, 36, 81] for a family of functions $\mathcal{F}$ over $\mathcal{R} \to \mathcal{R}'$, parameterized by $\alpha \in \mathcal{R}'$ and $\beta \in \mathcal{R}$, is a tuple of algorithms:

- $\mathsf{FSS.Gen}(1^\lambda, \alpha, \beta) \to \mathsf{key}_1, \mathsf{key}_2$: a randomized algorithm that takes in the security parameter λ and the secret function parameters $\alpha \in \mathcal{R}'$ and $\beta \in \mathcal{R}$ and produces two keys.
- $\mathsf{FSS.Eval}(i, \mathsf{key}_i, x) \to y$: a deterministic algorithm that takes in the party index $i \in \{1, 2\}$, a key, and an evaluation point $x \in \mathcal{R}$ and produces an output point $y \in \mathcal{R}'$.

We reiterate the two properties required of a FSS scheme [36, 37]: correctness and security.

Correctness. For any $\alpha \in \mathcal{R}'$ and $\beta \in \mathcal{R}$, let $f_{\alpha,\beta} : \mathcal{R} \to \mathcal{R}'$ be the function from the function family $\mathcal{F}$ parameterized by α, β. For any evaluation point $x \in \mathcal{R}$, the following probability (taken over the randomness of FSS.Gen) is equal to 1:

$$\Pr\left[y_1 + y_2 = f_{\alpha,\beta}(x) \;\middle|\; \begin{array}{r} \mathsf{key}_1, \mathsf{key}_2 \leftarrow \mathsf{FSS.Gen}(1^\lambda, \alpha, \beta) \\ \forall i \in \{1,2\}, y_i \leftarrow \mathsf{FSS.Eval}(i, \mathsf{key}_i, x) \end{array} \right]$$

Security. For each party $i \in \{1, 2\}$, there exists an efficient, randomized simulator Sim_i that can generate a FSS key that is indistinguishable from a real one, without knowing the secret function parameters $\alpha \in \mathcal{R}'$, $\beta \in \mathcal{R}$. That is, for all $i \in \{1, 2\}$, $\alpha \in \mathcal{R}'$ and $\beta \in \mathcal{R}$, the following two distributions are computationally indistinguishable:

- $\{\mathsf{key}_i : (\mathsf{key}_1, \mathsf{key}_2) \leftarrow \mathsf{FSS.Gen}(1^\lambda, \alpha, \beta)\}$ and
- $\{\mathsf{Sim}_i(1^\lambda, \mathcal{R}, \mathcal{R}')\}$.

3PC for truncation using FSS. We give our three-party truncation protocol in Figure 10. Our truncation gate takes as input and outputs ring elements in $\mathcal{R}$; to truncate vectors, it suffices to apply the protocol element-wise, as a SIMD operation. At the highest level, our protocol combines three techniques from prior work to minimize the amount of communication: it implements truncation (a) using a secret-shared comparison function to handle carries in the lowest-order bits [34, 36], (b) using padding in the highest-order bits to prevent overflow [72], and (c) using bit-length extension (in our case, by

3-party truncation (i.e., arithmetic right shift). Parameterized by security parameter $\lambda \in \mathbb{N}$, ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with bit length $b \geq 3$, and an integer number of bits to truncate $t \in [0, b-1]$. Given replicated shares $[\![x]\!]$ of an input $x \in \mathcal{R}$ in the range $[-2^{b-2}, 2^{b-2})$ when interpreted as a signed integer, this protocol produces replicated shares of the output $\mathsf{Trunc}(x) = (x \gg t) \in \mathcal{R}$.

Notation. Define the ring $\mathcal{R}' = \mathbb{Z}_{2^t}$ and the pad $\Delta = 2^{b-1} \in \mathcal{R}$. Let $(\mathsf{DCF.Gen}, \mathsf{DCF.Eval})$ denote a FSS scheme for the class of comparison functions $f : \mathcal{R}' \to \mathbb{F}_2$ with secret parameter $\beta \in \mathcal{R}'$, defined as $f_\beta(x) = \mathbb{1}_{x \geq \beta}$. For any value $w \in \mathcal{R}$, we write $\mathsf{PadTrunc}(w) = (w + \Delta) \gg t \in \mathcal{R}$ and we write $\mathsf{Rem}(w) = w \bmod 2^t \in \mathcal{R}'$.

Inputs. Each party holds replicated secret shares of the input: $[\![x]\!]$. We write party 0's share as (r_1, r_2), party 1's share as (r_2, z), and party 2's share as (z, r_1).

Protocol.

1. Party 0: samples $m, \hat{m} \xleftarrow{\text{R}} \mathcal{R}$, sets $r = -r_1 - r_2 \in \mathcal{R}$, generates $\mathsf{key}_1, \mathsf{key}_2 \leftarrow \mathsf{DCF.Gen}(1^\lambda, \mathsf{Rem}(r))$, sets $c_0 = 0$ and $v_0 = 0$, each in $\mathcal{R}$, and sends $(\mathsf{key}_1, m, \hat{m})$ to party 1 and sends $(\mathsf{key}_2, m - \mathsf{Trunc}(r), \hat{m} - \mathsf{PadTrunc}(r))$ to party 2.

2. Given message $(\mathsf{key}_1, m, \hat{m})$, party 1: evaluates $c_1 = \mathsf{DCF.Eval}(\mathsf{key}_1, \mathsf{Rem}(z))$, lifting the output from $\{0,1\}$ to $\mathcal{R}$, and if $z \in [\Delta/2, 2^b - \Delta/2)$, sets $v_1 = \mathsf{Trunc}(z) - m - 1 \in \mathcal{R}$. Else, sets $v_1 = \mathsf{PadTrunc}(z) - \hat{m} - 1 \in \mathcal{R}$.

3. Given message $(\mathsf{key}_2, n, \hat{n})$, party 2: evaluates $c_2 = -\mathsf{DCF.Eval}(\mathsf{key}_2, \mathsf{Rem}(z))$, lifting the output from $\{0,1\}$ to $\mathcal{R}$, and if $z \in [\Delta/2, 2^b - \Delta/2)$, sets $v_2 = n \in \mathcal{R}$. Else, sets $v_2 = \hat{n} \in \mathcal{R}$.

4. The three parties run $[\![c]\!] = \mathsf{Reshare}([c])$.

5. Each party locally computes $[v] = \mathsf{Add}([v], \mathsf{Mul}([\![c]\!], [\![c]\!]))$.

6. The three parties run $[\![y]\!] = \mathsf{Reshare}([y])$.

Outputs. Each party holds replicated shares $[\![y]\!]$.

Figure 10: A 3PC for truncation [34, 72, 104].

performing a multiplication) to allow the secret-shared comparison function to output as few bits as possible, at the cost of introducing one additional round of interaction [104]. For step (b) to work, we require that there is one bit of slack [72] between the value x to be truncated and the ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with $b \geq 3$—that is, viewed as a signed integer, x falls in the range $[-2^{b-2}, 2^{b-2})$.

In concrete costs, our protocol uses a single secret-shared comparison function and three rounds of interaction, rather than two comparison functions and three rounds (as used by Orca [104]) or four comparison functions and one round (as used by Boyle et al. [34]).

Proof sketch. Consider any security parameter λ, ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with bit length $b \geq 3$, and integer number of bits to truncate $t \in [0, b-1]$. We show correctness and security separately.

Correctness. Consider any input value $x \in \mathcal{R}$ in the range $[-\Delta/2, \Delta/2)$ when interpreted as a signed integer, where $\Delta = 2^{b-1}$. Since the parties hold replicated shares of the input $[\![x]\!]$, it must be that the value z held by parties 1 and 2 is equal to $x + r$. Then, by cases:

- If $z \in [\Delta/2, 2^b - \Delta/2)$, it must be that $x + r$ does not "wrap around" in $\mathcal{R}$. So, by [104, Lemma 1], it holds that

  $$\mathsf{Trunc}(x) = \mathsf{Trunc}(z) - \mathsf{Trunc}(r) - \mathbb{1}_{\mathsf{Rem}(z) < \mathsf{Rem}(r)},$$

  where "$\mathbb{1}_{\mathsf{Rem}(z) < \mathsf{Rem}(r)}$" denotes the indicator variable that is "1" if $\mathsf{Rem}(z) < \mathsf{Rem}(r)$ (using the natural integer representation of values in $\mathcal{R}'$) and "0" otherwise. By computing $v_0 = 0$, $v_1 = \mathsf{Trunc}(z) - m - 1$, and $v_2 = m - \mathsf{Trunc}(r)$, the parties hold additive shares of $v = \mathsf{Trunc}(z) - \mathsf{Trunc}(r) - 1$. It suffices to add to this additive shares of $\mathbb{1}_{\mathsf{Rem}(z) \geq \mathsf{Rem}(r)}$.

- If $z \notin [\Delta/2, 2^b - \Delta/2)$, it must be that $z + \Delta \in [\Delta/2, 2^b - \Delta/2)$. Then, letting $\hat{r} = r + \Delta \in \mathcal{R}$, it must be that $x + \hat{r}$ does not "wrap around" in $\mathcal{R}$. So, by [104, Lemma 1], it holds that

  $$\mathsf{Trunc}(x) = \mathsf{PadTrunc}(z) - \mathsf{PadTrunc}(r) - \mathbb{1}_{\mathsf{Rem}(z) < \mathsf{Rem}(r)}.$$

  By computing $v_0 = 0$, $v_1 = \mathsf{PadTrunc}(z) - \hat{m} - 1$, and $v_2 = m - \mathsf{PadTrunc}(r)$, the parties hold additive shares of $v = \mathsf{PadTrunc}(z) - \mathsf{PadTrunc}(r) - 1$. It suffices to add to this additive shares of $\mathbb{1}_{\mathsf{Rem}(z) \geq \mathsf{Rem}(r)}$.

By evaluating the secret-shared comparison function, the parties 1 and 2 hold bits c_1 and c_2 that sum to $\mathbb{1}_{\mathsf{Rem}(z) \geq \mathsf{Rem}(r)}$. By lifting these values to $\mathcal{R}$ and taking $c_0 = 0$, the three parties now hold 3-way additive shares of $\pm\mathbb{1}_{\mathsf{Rem}(z) \geq \mathsf{Rem}(r)} \in \mathcal{R}$. By evaluating one multiplication to square this value, the parties now hold 3-way additive shares of $\mathbb{1}_{\mathsf{Rem}(z) \geq \mathsf{Rem}(r)} \in \mathcal{R}$. So, they correctly end up with additive secret shares of $[\mathsf{Trunc}(x)]$. Running Reshare completes the protocol.

Security. We build a simulator for each party:

- Simulator for party 0: Given the security parameter λ and party 0's input share $[\![x]\!]_0$,
  1. proceed as in the truncation-gate protocol to send messages to parties 1 and 2 and compute c_0 and v_0,
  2. compute $[\![c]\!]_0$ using the simulator for Reshare, and
  3. proceed as in the truncation-gate protocol to compute the additive share y_0, and
  4. compute $[\![y]\!]_0$ using the simulator for Reshare.

  A sequence of hybrids shows that a corrupt party 0 cannot distinguish between a real-world execution of the truncation protocol and an ideal-world execution using the ideal functionality and the above simulator to generate its view:
  - Hybrid 0: the real-world execution.
  - Hybrid 1: replace the Reshare protocol by its simulator. When Reshare is instantiated with a secure PRF, this is computationally indistinguishable from the prior world.
  - Hybrid 2: the ideal-world execution. This is identical to the prior world.
- Simulator for parties 1 and 2: for $i \in \{1,2\}$, given the security parameter λ and party i's input share $[\![x]\!]_i$,
  1. simulate the message $(\mathsf{key}, m, \hat{m})$ received from party 0 by running the simulator for the FSS comparison function to sample key and sampling $m, \hat{m} \xleftarrow{R} \mathcal{R}$,
  2. proceed as in the truncation protocol to compute c_i and v_i,
  3. compute $[\![c]\!]_i$ using the simulator for Reshare,
  4. proceed as in the truncation protocol to compute the additive share y_i, and
  5. compute $[\![y]\!]_i$ using the simulator for Reshare.

  A sequence of hybrids shows that a corrupt party i cannot distinguish between a real-world execution of the truncation protocol and an ideal-world execution using the ideal functionality and simulator to generate its view:
  - Hybrid 0: the real-world execution.
  - Hybrid 1: replace the share-exchange protocol Reshare by its simulator. When Reshare is instantiated with a secure PRF, this is computationally indistinguishable from the prior world.
  - Hybrid 2: replace the message from party 0 to party i by a call to the FSS simulator and two random samples from $\mathcal{R}$ (as in step 1 above). When instantiated with a secure FSS scheme, this is computationally indistinguishable from the prior world.
  - Hybrid 3: the ideal-world execution. This is identical to the prior world.

This completes the argument.

Approximate-normalization protocol using FSS. We give our 3PC protocol for approximate normalization in Figure 11. Our protocol blends two techniques from prior work. Given replicated secret shares of an input vector $\mathbf{x} \in \mathcal{R}^d$, it works by (a) computing $[\![\|\mathbf{x}\|^2]\!]$, (b) using Boyle et al.'s FSS for the "most significant non-zero bit" function [34] to compute an approximation of $[\![1/\|\mathbf{x}\|]\!]$, and (c) refining this approximation using the Newton–Raphson method to compute inverse square-roots, as proposed by Catrina and Saxena [42, 110, 169, 171].

In concrete costs, our protocol uses one invocation of the FSS scheme for the "most significant non-zero bit" function (which, under the hood, requires $b+1$ FSS keys for comparison functions), along with a constant number of invocations of the Newton–Raphson method, each of which evaluates a degree-3 polynomial [114].

Proof sketch. Consider any security parameter λ, ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with bit length $b \geq 1$, vector dimension $d \in \mathbb{N}$, number of fractional bits of precision f in the inputs/outputs and f' in the intermediate values, and number of iterations of Newton's method $c \in \mathbb{N}$. We show correctness and security separately.

Correctness. Let $\mathbf{x} \in \mathcal{R}^d$ be any input vector. By construction, we see that the value z held by parties 1 and 2 is equal to $\|\mathbf{x}\|^2 + r$. Then, parties 1 and 2 construct $v_1, v_2 \in \mathcal{R}$ such that

$$\begin{aligned} v_1 + v_2 &= \sum_{j=0}^{b-1} \mathbb{1}_{j \leq \log(z-r) < j+1} \cdot 2^{f'-(j-f)/2} \\ &= \sum_{j=0}^{b-1} \mathbb{1}_{j \leq \log(\|\mathbf{x}\|^2) < j+1} \cdot 2^{f'-(j-f)/2} \\ &= 2^{f'-(\lfloor \log(\|\mathbf{x}\|^2) \rfloor - f)/2} \\ &= 2^{f'} \cdot 2^{-(\lfloor \log(\|\mathbf{x}\|^2) \rfloor - f)/2}. \end{aligned}$$

This is exactly a fixed-point representation of $2^{-(\lfloor \log(\|\mathbf{x}\|^2) \rfloor)/2}$ with f' bits of decimal precision (accounting for the fact that $\|\mathbf{x}\|^2$ is represented using f bits of fractional precision).

After this, the protocol computes c iterations of the Newton-Raphson method for computing inverse-square-roots, using the update method from [114, Equation 10] and truncating at each step to maintain the correct data format. Finally, the protocol multiplies the secret-shared vector $\mathbf{x}$ by the secret-shared approximation of $1/\|\mathbf{x}\|$. So, the parties correctly end up with replicated secret shares of $[\![\mathsf{ApproxNormalize}(\mathbf{x})]\!]$.

Security. We build a simulator for each party:

- Simulator for party 0: Given the security parameter λ and party 0's input share $[\![\mathbf{x}]\!]_0$,
  1. Use the simulators for Reshare and Trunc to compute $[\![m]\!]_0$.

3-party approximate normalization. Parameterized by security parameter $\lambda \in \mathbb{N}$, vector dimension $d \in \mathbb{N}$, ring $\mathcal{R} = \mathbb{Z}_{2^b}$ with bit length $b \geq 1$, number of fractional bits of precision f for the input/output and f' for the intermediate values where $0 \leq f \leq f' \leq b$, and number of iterations of Newton's method $c \in \mathbb{N}$. Given replicated shares of a vector $\mathbf{x} \in \mathcal{R}^d$, this protocol produces replicated shares of the vector $\mathsf{ApproxNormalize}(\mathbf{x}) \in \mathcal{R}^d$, computed as (1) $v := 2^{-\lfloor \log(\|\mathbf{x}\|^2) \rfloor/2}$, (2) for c iterations: update $v := v \cdot \left(3 - \|\mathbf{x}\|^2 \cdot v^2\right)/2$, [114], and (3) output $\mathbf{y} := v \cdot \mathbf{x}$.

Notation. Let $(\mathsf{MSB.Gen}, \mathsf{MSB.Eval})$ be a FSS scheme for the most-significant-bit function $f : \mathcal{R} \to \mathcal{R}^b$ with secret parameter $\beta \in \mathcal{R}'$, defined as: $f_\beta(x) = \left[\mathbb{1}_{0 \leq \log(x-\beta) < 1} \;\|\; \cdots \;\|\; \mathbb{1}_{b-1 \leq \log(x-\beta) < b}\right]$. On $v \in \mathcal{R}$, we write $\mathsf{TruncReshare}([v], t)$ to denote the interactive process $\mathsf{Reshare}(\mathsf{Trunc}(\mathsf{Reshare}([v]), t))$.

Inputs. Each party holds replicated shares of input $[\![\mathbf{x}]\!]$.

Protocol. *// Step 1: compute* $v = 2^{-\lfloor \log(\|\mathbf{x}\|^2) \rfloor/2}$

1. Each party computes $[m] = \mathsf{MatrixMul}([\![\mathbf{x}^\top]\!], [\![\mathbf{x}]\!])$ and $[\![m]\!] = \mathsf{TruncReshare}([m], f)$. *// $m = \|\mathbf{x}\|^2$*
2. Party 0: parses $[\![m]\!]_0$ as (r_1, r_2), sets $r = -r_1 - r_2$ and $v_0 = 0$, each in $\mathcal{R}$, generates $\mathsf{key}_1, \mathsf{key}_2 \leftarrow \mathsf{MSB.Gen}(1^\lambda, r)$, and sends key_1 to party 1 and key_2 to party 2.
3. Given message key_i for $i \in \{1,2\}$, party i parses $[\![m]\!]_i$ as $(_, z)$ if $i = 1$, or $(z, _)$ if $i = 2$. Then, party i sets $v_i = \sum_{j=0}^{b-1} \mathsf{MSB.Eval}(i, \mathsf{key}_i, z)_j \cdot 2^{f' - (j-f)/2}$.
4. The parties run $[\![v]\!] = \mathsf{Reshare}([v])$.
5. For c iterations: *// Step 2: Refine* $v \approx 1/\|\mathbf{x}\|$
   (a) Compute $[u] = \mathsf{Mul}([\![v]\!], [\![m]\!])$.
   (b) Run $[\![u]\!] = \mathsf{TruncReshare}([u], f)$.
   (c) Compute $[w] = \mathsf{Mul}([\![v]\!], [\![u]\!])$.
   (d) Run $[\![w]\!] = \mathsf{TruncReshare}([w], f')$.
   (e) Compute $[v] = \mathsf{Mul}([\![v]\!], 3 - [\![w]\!])$.
   (f) Run $[\![v]\!] = \mathsf{TruncReshare}([v], f'+1)$.

   // Step 3: compute and output $\mathbf{y} = v \cdot \mathbf{x} \approx 1/\|\mathbf{x}\| \cdot \mathbf{x}$
6. Each party computes $[\mathbf{y}] = \mathsf{MatrixMul}([\![v]\!], [\![\mathbf{x}]\!])$.
7. The parties run $[\![\mathbf{y}]\!] = \mathsf{TruncReshare}([\mathbf{y}], f')$.

Output. Each party holds shares $[\![\mathbf{y}]\!]$.

Figure 11: A 3PC for normalization [34, 169].

2. Proceed as in the normalization protocol to initialize $v_0 = 0$ and send messages to parties 1 and 2.
3. Compute the share $[\![v]\!]_0$ using the simulator for Reshare.
4. For c iterations: locally compute each Mul and Add and use the simulators for Reshare and Trunc.
5. Use the simulators for Reshare and Trunc to compute the replicated share $[\![\mathbf{y}]\!]_0$.

A sequence of hybrids shows that a corrupt party 0 cannot distinguish between a real-world execution of the normalization protocol and an ideal-world execution using the ideal functionality and the above simulator to generate its view:

- Hybrid 0: the real-world execution.
- Hybrid 1: replace each Reshare protocol by its simulator. When Reshare is instantiated with a secure PRF, this is computationally indistinguishable from the prior world.
- Hybrid 2: replace each truncation protocol for Trunc by its simulator. When Trunc is instantiated with a secure 3PC protocol (given in Figure 10), this is computationally indistinguishable from the prior world.
- Hybrid 3: the ideal-world execution. This is identical to the prior world.

- Simulator for parties 1 and 2: For $i \in \{1,2\}$, given the security parameter λ and party i's input share $[\![\mathbf{x}]\!]_i$,
  1. Use the simulators for Reshare and Trunc to compute $[\![m]\!]_i$.
  2. Use the simulator for the FSS scheme for the most-significant-bit function to obtain key_i, and from it compute v_i as in the protocol.
  3. Compute the share $[\![v]\!]_i$ using the simulator for Reshare.
  4. For c iterations: locally compute each Mul and Add and use the simulators for Reshare and Trunc.
  5. Use the simulator for Reshare and Trunc to compute the replicated share $[\![\mathbf{y}]\!]_i$.

  A series of hybrids shows that a corrupt party i cannot distinguish between a real-world execution of the normalization protocol and an ideal-world execution using the ideal functionality and simulator to generate its view:
  - Hybrid 0: the real-world execution.
  - Hybrid 1: replace each Reshare protocol by its simulator. When Reshare is instantiated with a secure PRF, this is computationally indistinguishable from the prior world.

– Hybrid 2: replace each truncation protocol for Trunc by its simulator. When Trunc is instantiated with a secure 3PC protocol (given in Figure 10), this is computationally indistinguishable from the prior world.

– Hybrid 3: replace the FSS scheme for the most-significant-bit function with its simulator. When the FSS scheme is instantiated with a secure PRF, this is computationally indistinguishable from the prior world.

– Hybrid 4: the ideal-world execution. This is identical to the prior world.

This completes the argument. The efficiency guarantees of Theorem 5.1 follow by inspection.

D Additional material on implementation

Collecting ratings from users. Taking α to encode its rating and β to encode the associated item, each user generates three compact FSS keys for a point function $f_{\alpha,\beta}(x) = \alpha \cdot \mathbb{1}_{x=\beta}$ that each individually hide α and β, and from which it is possible to derive secret shares of $[\![\alpha]\!]$ at evaluation point β and secret shares of $[\![0]\!]$ at all other evaluation points. A Nudge client submits one key to each Nudge server to log its rating. Given a key from client i, the servers then evaluate this key at each of the points in $\{1, 2, \ldots, n\}$ to reconstruct a secret-shared vector in $\mathcal{R}^n$. The servers add this vector to the ith row in their local share of the matrix $[\![\mathbf{U}]\!]$.

Checking that ratings are 0/1. In all datasets that we evaluate on, it is common practice to binarize the ratings to "1" or "0" (to indicate like or dislike) [75, 119]. As a result, Nudge needs to check that client ratings are in $\{0, 1\}$. To perform this check, the servers can compute the random degree-two combination

$$[b] = \sum_{i=1}^{m} \sum_{j=1}^{n} r_{i,j} \cdot [\![u_{i,j}]\!] \cdot ([\![u_{i,j}]\!] - 1),$$

where $[\![u_{i,j}]\!]$ denotes the entry at position (i, j) in matrix $[\![\mathbf{U}]\!]$ and each coefficient $r_{i,j} \xleftarrow{R} \mathcal{R}$. The servers then communicate to check whether $b = 0$; per prior work [32], if $b = 0$, then all clients must have submitted valid ratings with overwhelming probability $1 - 1/|\mathcal{R}| = 1 - 2^{-128}$.

Limiting the number of ratings. If desired, Nudge can enforce that each client may only submit a bounded number B of ratings. To this end, limiting the number of FSS keys submitted by each client is not enough: a malicious client could contribute more ratings by submitting malformed keys [31]. The servers need to check that, if client i uploaded $k_i \leq B$ keys, then there are exactly k_i "1"s in the client's rating vector. The servers enforce this with another random linear combination [32]: sampling $r_i \xleftarrow{R} \mathcal{R}$ for each user $i \in \{1, \ldots, m\}$,

$$[\![b']\!] = \sum_{i=1}^{m} r_i \cdot \left(\sum_{j=1}^{n} [\![u_{i,j}]\!] - k_i \right).$$

The servers can again exchange one field element to check whether $b' = 0$. If so, all clients must have submitted $\leq B$ ratings with overwhelming probability $1 - 1/|\mathcal{R}| = 1 - 2^{-128}$.

Extension: Checking arbitrary predicates on user data. More broadly, the servers may want to enforce that each user's submission satisfies an arbitrary arithmetic circuit C. For example, the servers could confirm that the users' submissions are one-hot vectors with a non-zero element in a specific range. To do so, the servers could use arithmetic sketching [32] to check that the vector is one-hot. Then, the users could provide a zero-knowledge proof on secret-shared data [30] proving that the sum of all elements in its submitted vector satisfies circuit C. This requires user-server communication linear in the number of multiplication gates in C. Checking the sketch and the proof requires the servers to compute a degree-two function on secret-shared data, which they can do non-interactively in our three-party setting. As a result, the Nudge servers can again take random linear combinations to check all proofs in a batch and communicate only one ring element.

To check more complex predicates on *each* element of the user's submissions, such as ensuring that every element satisfies a circuit, the user and servers could engage in a more powerful interactive zero-knowledge proof on secret-shared data [30, Corollary 6.7].

E Additional material on dimensionality reduction

When the number of items n is sufficiently large, Nudge may proceed as follows to reduce the dimensionality of its matrix before power iteration (again mapping all computations to fixed-point arithmetic): on error parameter ϵ,

1. Nudge samples a sketch $\mathbf{S} \in \mathbb{R}^{n \times s}$ of dimension $s = \Theta(\log n/\epsilon^2)$ according to a desired sketch distribution [108, 117]. To ensure that all user submissions are chosen *independently of the randomness of the sketch*, Nudge samples $\mathbf{S}$ after all user ratings are submitted.

2. Nudge computes the smaller matrix $[\![\hat{\mathbf{U}}]\!] = [\![\mathbf{U}]\!] \cdot \mathbf{S} \in \mathbb{R}^{m \times s}$. Since the sketch is linear, this step requires local computation and no communication between the parties.

3. Nudge runs its matrix-factorization protocol on $[\![\hat{\mathbf{U}}]\!] \in \mathbb{R}^{m \times s}$ to obtain $[\![\hat{\mathbf{A}}]\!] \in \mathbb{R}^{m \times d}$ and $\hat{\mathbf{B}} \in \mathbb{R}^{d \times s}$.

4. Nudge outputs $[\![\mathbf{A}]\!] := [\![\hat{\mathbf{A}}]\!] \in \mathbb{R}^{m \times d}$ and $[\mathbf{B}] = [\![\hat{\mathbf{A}}^\top]\!] \cdot [\![\mathbf{U}]\!] \in \mathbb{R}^{m \times d}$. This step is degree-two, requiring local computation and no communication between the parties.

F Additional discussion

Privately tuning hyperparameters. Developers may need to tune Nudge hyperparameters for a chosen application context. For applications where it is possible to measure Nudge's recommendation quality (i.e., there are existing data sets of representative user ratings), developers could use existing, published data to choose hyperparameters for Nudge.

When this is not the case, Nudge needs a mechanism for privately selecting hyperparameters. One way to do this would be to test different hyperparameters (e.g., gradually increasing the embedding dimension d until achieving suitable accuracy), and each time have the clients withhold some ratings, run the private matrix-factorization protocol, and then have the clients compute the scores for the withheld items in order to measure the recall/nDCG in a distributed manner. The Nudge servers could use private aggregation [55] to learn the aggregate recall/nDCG across all clients. One drawback of this approach is that a malicious client could disrupt hyperparameter training. An interesting direction for future work would be to privately tune hyperparameters while defending against malicious clients (e.g., via cryptographic proofs).