SparseMPC: Secure Sparse Operations using Multi-Party Computation

Marc Damie[1 [0000−0002−9484−4460]]

Radboud University Nijmegen, The Netherlands

Abstract. Multi-party computation (MPC) enables multiple parties to jointly process sensitive data without revealing their inputs. However, existing MPC protocols remain inefficient for high-dimensional sparse data. In plaintext, sparse linear algebra algorithms address this problem using two fundamental primitives, Scatter and Gather.
We propose SparseMPC, an outsourced MPC protocol that securely implements Scatter and Gather and uses them to perform sparse matrix multiplication. Our protocol supports an arbitrary number of data owners and provides a low memory footprint, constant round complexity, and low communication cost. Beyond sparse matrix multiplication, SparseMPC provides a foundation for efficiently realizing a broader class of sparse computations in outsourced MPC.

Keywords: Sparse data · Multi-Party Computations · Privacy-Preserving Machine Learning · Secret Sharing.

1 Introduction

Driven by recent data privacy regulations [1, 15], cryptographers have designed various protocols to process sensitive data securely. Among these approaches, multi-party computation (MPC) [62, 67, 74] has found applications in many domains, as it enables multiple parties to jointly perform computations without revealing their respective inputs. For example, in Privacy-Preserving Machine Learning (PPML), MPC protocols are used to train Machine Learning (ML) models without revealing the training data. Beyond ML, MPC has also been applied to fraud detection [65], graph analysis [2, 51], and heavy hitters [3]. While MPC has become very efficient for operations such as vector multiplication [28, 52, 58, 61, 63, 74], existing protocols remain inefficient for some types of high-dimensional data, such as sparse data.

Sparse data refers to datasets containing a large majority of zeros, typically more than 95%. Sparse data occurs naturally in many applications, including recommender systems, ML, and genomics. For example, consider a recommender system deployed on a large streaming platform with n users and m movies. The recommender system takes as input the watch history of all users, represented as an $n \times m$ matrix W, where $W_{ij} = 1$ if user i has seen movie j. As modern platforms have millions of users and thousands of movies, the matrix W is extremely large, but it is also sparse because each user has watched only a small fraction

of the catalog. If stored as a standard matrix, the watch history would require an impractical amount of memory, even on high-capacity servers. This example illustrates the main challenge of processing sparse data in MPC: representing the entire matrix causes the computation and memory cost to depend on the matrix dimensions rather than on the number of non-zero values.

To avoid this bottleneck, the linear algebra literature [31] has proposed so-called "sparse algorithms" to efficiently process sparse data. These algorithms reduce the computational cost by operating on non-zero values, making their cost proportional to the number of non-zero values rather than the matrix size. In plaintext, BLAS[1] [30] implements sparse algorithms using two fundamental primitives: Scatter and Gather. Scatter writes values to positions specified by indices, while Gather retrieves values from positions specified by indices. Efficiently supporting these two primitives is therefore a natural way to construct sparse linear algebra operations.

In MPC, several works have proposed protocols optimized for sparse data, especially for vector aggregation [4, 6, 7, 27] and matrix multiplication [20, 22, 26, 23, 67]. Most of these works do not reproduce the generic Scatter and Gather primitives used in plaintext. Instead, they design dedicated protocols for the targeted operation. For example, Damie et al. [26] designed a sparse matrix multiplication protocol based on oblivious sorting.

Schoppmann et al. [67] is the only work in MPC that has attempted to implement MPC variants of the Scatter and Gather primitives. Unfortunately, as also noted in [26], their protocols are only compatible with a limiting MPC setup involving two data owners. This assumption limits the applications of their protocols, making them incompatible with applications such as PPML, where many data owners can be involved.

A more general setting is outsourced MPC, which is used in several recent works [26, 62]. In this setting, data owners secret-share their data with a group of computation servers, which then perform the necessary computations on the secret-shared dataset. The data owners do not need to participate in the computation after sharing their inputs, making the setting suitable for applications involving many data owners. Supporting Scatter and Gather in this setting would therefore provide generic building blocks for sparse MPC operations while avoiding the two-data-owner limitation of prior work.

Our Contributions We propose "SparseMPC", an outsourced MPC protocol implementing two fundamental linear algebra primitives: Scatter and Gather. We then use these primitives to construct sparse matrix multiplication and compare it asymptotically to state-of-the-art MPC protocols [26]. Our protocol simultaneously provides a low memory footprint, constant round complexity, and low communication cost.

Other Related Work Recently, several works have proposed algorithms based on fully homomorphic encryption (FHE) to process encrypted sparse matrices

[1] BLAS (Basic Linear Algebra Subprograms) is a popular low-level library providing implementations of many fundamental linear algebra operations.

[21, 37, 78]. While closely related, FHE makes different assumptions from MPC, making these protocols not directly comparable to ours. However, their underlying intuitions can still inspire solutions in MPC, as they address similar privacy and efficiency constraints.

In parallel with these works, protocols for multiplying secret-shared sparse matrices have also been presented in the information theory literature [75, 9, 32]. These works focus on system architectures that are incompatible with classic MPC setups. Xhemrishi et al. [75] study the multiplication of a private sparse matrix with a *public* vector. Egger et al. [32] present the multiplication of two private sparse matrices, *both owned by a single party.*

2 Preliminaries

2.1 Secret sharing

Secret-sharing schemes enable a secret to be split into n shares such that each individual share reveals no information about the secret. A threshold secret-sharing scheme is characterized by a pair (t, n), where any set of at least t shares can reconstruct the secret, while fewer than t shares reveal no information about it. Secret sharing is a core building block of many MPC protocols [33]. The MPC literature provides protocols for performing basic arithmetic and comparison operations on secret-shared data, as well as more complex operations such as sorting [43] and shuffling [53]. We refer to Evans et al. [33] for an overview of fundamental MPC protocols.

Our work relies on Shamir's secret-sharing scheme [68]. In Shamir's scheme, the secret holder generates a random polynomial p of degree $t - 1$ such that $p(0) = s$, where s is the secret value. The shares are evaluations of this polynomial at distinct non-zero points. To reconstruct the secret, one interpolates the polynomial from any t shares and evaluates it at 0. We use the notation $[\![x]\!]_j$ to denote the j-th share of a secret value x.

2.2 Sparse data

The term "sparse data" refers to data containing a large proportion of zeros. Sparse data is common in ML applications, especially in recommender systems. For example, the MovieLens 1M dataset [45] is sparse, while the Netflix [8], Yahoo Movies, and Bookcrossing [80] datasets contain more than 99% zero entries. Sparsity also occurs in other domains, including drug discovery [42, 48], text analysis [64], social network analysis [19], and graph analysis [59, 54]. The proportion of zero entries can be extremely high, reaching 99.999% in some datasets.

As sparse datasets can be high-dimensional, classic matrix algorithms can be impractical on such data. First, they perform unnecessary operations on zero entries. Second, storing a high-dimensional sparse dataset as a dense matrix can require impractical amounts of memory. To address these issues, several works [5, 30, 31] have proposed algorithms that exploit sparsity by storing and processing

only non-zero elements. We refer to these algorithms as "sparse algorithms," in contrast to "dense algorithms," which operate on the complete matrix.

Sparse algorithms are widely used to process sparse data in ML. Although many ML models are represented using dense data structures, their training algorithms often operate on sparse inputs. For example, recommender systems typically use matrix factorization to transform large sparse training matrices into smaller dense matrices [46]. During training, gradient-based algorithms repeatedly multiply the sparse training matrix by dense vectors or matrices representing the model. Similarly, many other ML applications require operations between sparse input data and dense model parameters. Therefore, efficiently supporting sparse-dense multiplication is important for ML applications involving high-dimensional sparse data.

Sparse ML models also exist, including sparse neural networks [38, 55, 70] and Lasso regression [71, 72]. However, sparsity in the model does not necessarily imply sparse computation during training. For example, one approach to training sparse neural networks is to first train a dense neural network and then remove weights with small magnitudes. Thus, these training algorithms can still require sparse-dense multiplication when the input data is sparse.

2.3 Low-level primitives for sparse operations

Duff et al. [30] proposed low-level primitives for implementing plaintext sparse algorithms. They rely on two fundamental primitives: Scatter and Gather.

The Scatter operation takes as input a list of L values $\{v_1, \ldots, v_L\}$ and a list of L indices $\{i_1, \ldots, i_L\}$. It outputs an n-dimensional vector containing v_k at index i_k and zeros at the remaining indices. The Gather operation takes as input an n-dimensional vector and a list of L indices $\{i_1, \ldots, i_L\}$. It outputs a list containing the values stored at these indices. Thus, Scatter and Gather provide conversions between a sparse representation and a dense vector.

Duff et al. [30] use these primitives to construct sparse matrix operations, including sparse matrix multiplication. They therefore provide two fundamental building blocks for implementing plaintext sparse algorithms. Schoppmann et al. [67] adapted this approach to MPC by providing secure variants of Scatter and Gather and constructing sparse matrix multiplication from these primitives.

3 System Properties

3.1 Outsourced MPC Setup

Outsourced MPC is an MPC setting designed to process private data from multiple data owners. In this setting, data owners secret-share their inputs with a group of non-colluding computation servers, which then perform the secure computation on the secret-shared data. As ML applications can involve hundreds or thousands of data owners, several PPML protocols [26, 47, 62, 73, 74] adopt this setting. Figure 1 compares outsourced and non-outsourced MPC settings.

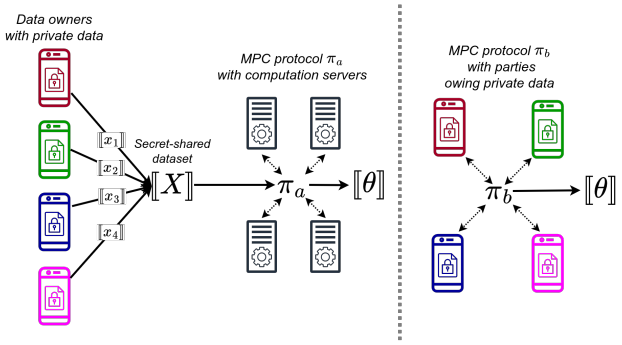


Fig. 1: Outsourced (on the left) and non-outsourced (on the right) MPC settings.

The outsourced setting has three main benefits. First, data owners only need to participate in the initial sharing of their inputs and can then disconnect. The computation therefore does not require data owners to remain available throughout the protocol, which is impractical when thousands of data owners are involved. Second, the communication cost does not grow quadratically with the number of data owners. In a non-outsourced MPC protocol, data owners may need to communicate with each other during the computation, resulting in communication costs that depend quadratically on the number of data owners. In outsourced MPC, data owners only communicate with the computation servers, so their number affects the input size linearly. Third, an outsourced MPC protocol can also be used in a non-outsourced setting by letting the data owners act as computation servers. The converse does not hold in general, as a non-outsourced protocol cannot be directly used in an outsourced setting.

Most existing works on secure sparse multiplication [20, 22, 67] consider two-party protocols and are therefore limited to non-outsourced settings. Damie et al. [26] recently proposed the first secure sparse matrix multiplication protocol compatible with an outsourced setting. Our work also supports the outsourced setting and uses the protocol of Damie et al. as a baseline for sparse matrix multiplication.

We describe outsourced protocols using two phases. First, the *preparation phase* requires the data owners to share their private information with the computation servers. This phase can consist of simple input secret-sharing, but can also involve more complex forms of secret information sharing. For example, Section 4.3 introduces a form of "secret-shared" permutation. Second, the *processing phase* requires the computation servers to securely process the shared inputs and produce the desired output. As we focus on outsourced MPC, all our protocols and functionalities are described using these two phases.

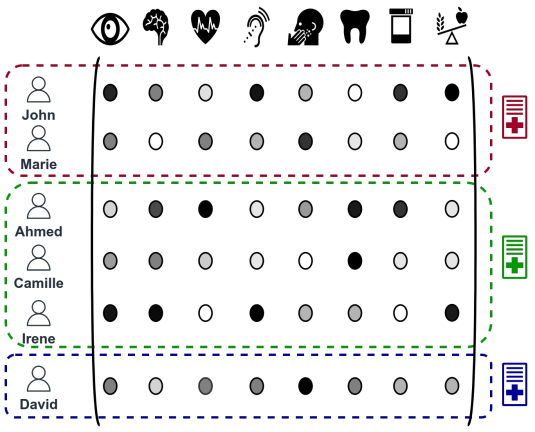


Fig. 2: Horizontal data partitioning.

3.2 Threat Model

We consider a semi-honest adversary that follows the protocol correctly but attempts to infer private information from its view of the execution. As in Damie et al. [26], we assume that data owners provide well-formed secret-shared inputs.

We assume an honest majority among the computation servers. This assumption has been used in several PPML protocols to efficiently perform matrix operations. We rely on the same assumption to construct efficient sparse matrix multiplication protocols. Damie et al. [26] also consider both honest-majority and dishonest-majority settings, but their honest-majority protocols provide more efficient implementations.

3.3 Horizontally Partitioned Data

MPC-based PPML protocols can involve multiple data owners, particularly in an outsourced setting. Such protocols therefore perform a form of decentralized ML. Let X be a matrix containing the training data of all data owners. In ML, each row typically represents a data point or individual, such as a patient, while each column represents a property, such as patient height. The ML literature [77] commonly distinguishes between horizontal and vertical data partitioning. In horizontal partitioning, each data owner holds multiple non-overlapping rows of X. In vertical partitioning, each data owner holds multiple columns of X.

Like many PPML protocols [11, 50, 60, 69], we focus on horizontal data partitioning. This setting captures many practical applications in which each data owner has data from multiple individuals. For example, hospitals can jointly train a model using their respective patient data without revealing the data to each other. Each hospital therefore holds multiple rows corresponding to its patients, which corresponds to horizontal partitioning. Damie et al. [26] also consider this data partitioning in their secure sparse matrix multiplication protocol.

Notations We use X to denote the $n \times p$ training data matrix. Each data owner C_l owns an $n_l \times p$ matrix $X^{(l)}$. Let $\text{nnz}(Y)$ denote the number of non-zero values in a matrix Y. Let $\text{nnzc}(Y)$ denote the number of columns in Y containing at least one non-zero value.

3.4 Leakage Profile

Information leakage is commonly used to improve the efficiency of MPC protocols [33]. Revealing selected information can avoid expensive secure computations. For sparse operations, some leakage is necessary to exploit sparsity. As noted by Damie et al. [26], the efficiency gains of sparse algorithms depend on the sparsity of the input. Therefore, an MPC protocol must reveal some information about the sparsity if it is to obtain similar efficiency gains. Some works also refer to the leakage as "Prior/Public Knowledge."

Schoppmann et al. [67] reveal the number of non-zero columns in the matrix of each data owner, making $\text{nnzc}(X^{(i)})$ public for each data owner C_i. Damie et al. [26], in contrast, reveal the number of non-zero values in each row, making $\text{nnz}(X_j)$ public for each row j. Our work uses the same leakage profile as Schoppmann et al. [67], namely $\text{nnzc}(X^{(i)})$. Damie [23] connects these leakage profiles to known leakage profiles in searchable encryption in order to demonstrate their necessity and acceptability.

Under our horizontal partitioning assumption, we argue that column sparsity reveals less information about individual data subjects than per-row sparsity. In ML datasets, each row typically corresponds to an individual, while each column corresponds to a property. Revealing the number of non-zero values in a row therefore provides information about a specific individual. In contrast, revealing the number of non-zero columns provides information about the dataset as a whole. This distinction does not apply when each data owner holds only one row, in which case the two leakage profiles are equivalent. Our focus on horizontal partitioning means that each data owner typically holds multiple rows, making the distinction relevant.

If the column sparsity leakage is too sensitive for a particular application, data owners can add dummy non-zero columns to their inputs. A data owner using this padding technique reveals only an *upper bound* on the number of non-zero columns. Damie et al. [26] proposes even more advanced padding technique to mitigate the leakage.

4 SparseMPC

This section presents our protocols for securely processing sparse data in an outsourced MPC setting. Our architecture follows the approach used in plaintext sparse linear algebra: we directly implement Scatter and Gather and use them as building blocks for higher-level sparse operations. Appendix A provides proof sketches for the protocols presented in this section.

Parties: M computation servers $\{P_1, \dots P_M\}$, one data owner.
Inputs:
- Each computation server P_k owns a share of $[\![v]\!]_k$ a dense L-dimensional vector, with $v \in \mathcal{V}^L$ (all the elements in $\mathcal{V}$ are of size S).
- The data owner knows a list of L unique indices $\{\alpha_1, \dots, \alpha_L\}$, with the indices between 1 and n.

Outputs:
- **Prepration phase** Scatter<prep>: the data owner outputs a form of shared indices $\{\langle\alpha_1\rangle, \dots \langle\alpha_L\rangle\}$; one share per computation server.
- **Processing phase** Scatter<proc>: the computation servers use the shared indices $\{\langle\alpha_1\rangle, \dots, \langle\alpha_L\rangle\}$ to output a secret-shared n-dimensional vector $[\![w]\!]$ (i.e., $w \in \mathcal{V}^n$) defined as follows: for all $j \in \{1, \dots n\}$, $w_j = v_i$ if there exists $\alpha_i \in \{\alpha_1, \dots, \alpha_L\}$ such that $\alpha_i = j$, otherwise, $w_j = 0$.

Fig. 3: Functionality of Outsourced Scatter

4.1 Outsourced scatter and gather

We first define the outsourced Scatter and Gather functionalities. These functionalities capture the operations that our protocols implement and explicitly separate the preparation and processing phases of outsourced MPC. Figure 3 presents the outsourced Scatter functionality, while Figure 4 presents the outsourced Gather functionality.

An important property of these functionalities is that the output of the preparation phase can be reused across multiple processing phases. This property is particularly useful in iterative ML algorithms, where the same data is processed repeatedly. For example, gradient descent repeatedly multiplies the training data matrix by a model parameter vector.

Our functionalities describe a single data owner for simplicity. In our setting, each data owner has its own list of indices and can independently perform the preparation phase. The functionality therefore extends directly to N data owners by running the preparation and processing phases independently for each data owner. We use the single-owner formulation to keep the functionality concise without loss of generality.

The protocols of Schoppmann et al. [67] do not implement these outsourced functionalities. Their protocols require the data owner to remain involved throughout the computation because one of the computation parties must know the indices. This requirement is incompatible with our outsourced setting, where data owners can disconnect after the preparation phase.

The standard definitions of Scatter and Gather usually operate on vectors of scalar values [67]. We generalize these definitions to vectors whose elements belong to an arbitrary domain $\mathcal{V}$, where each element has size S. This notation covers vectors of scalars, vectors of vectors, and vectors of matrices. The latter

Parties: M computation servers $\{P_1, \dots P_M\}$, one data owner.
Inputs:
- Each computation server P_k owns a share of $[\![w]\!]_k$ a sparsely populated n-dimensional vector, with $w \in \mathcal{V}^n$ (all the elements in $\mathcal{V}$ are of size S).
- The data owner knows a list of L unique indices $\{\alpha_1, \dots, \alpha_L\}$, with the indices between 1 and n.

Outputs:
- **Preparation phase** Gather<prep>: the data owner outputs a form of shared indices $\{\langle\alpha_1\rangle, \dots \langle\alpha_L\rangle\}$; one share per computation server.
- **Processing phase** Gather<proc>: the computation servers use the shared indices $\{\langle\alpha_1\rangle, \dots, \langle\alpha_L\rangle\}$ to output a secret-shared L-dimensional vector $[\![v]\!]$ (i.e., $v \in \mathcal{V}^L$) defined as follows: for all $j \in \{1, \dots L\}$, $v_j = w_{\alpha_j}$.

Fig. 4: Functionality of Outsourced Gather

two cases are useful for matrices and tensors; widely used in ML. In practice, implementations of sparse operations on matrices and tensors can extend the corresponding vector operations, so our generic definition also covers these cases.

Finally, our functionalities do not restrict the origin of the input secret-shared vectors. The vectors may have been directly secret-shared by a data owner or may have been produced by a previous secure computation. This abstraction allows the same Scatter and Gather protocols to be used as building blocks in different applications.

4.2 Secret-index-based implementations

Secret indexing takes as input a secret-shared index $[\![\alpha]\!]$, with $\alpha \in \{1, \dots, n\}$, and outputs a secret-shared unit vector $[\![u_\alpha]\!] \in \{0, 1\}^n$ such that u_α is equal to 1 at index α and 0 elsewhere. De Hoogh [28] proposed a secret-indexing protocol based on Shamir's secret sharing that requires $O(\log n)$ secure comparisons. Although the protocol requires $O(n)$ communication, it reduces the number of secure comparisons, which are typically more expensive than multiplications.

We use secret indexing to implement Gather as follows. For each secret-shared index $[\![\alpha_i]\!]$, the servers generate the corresponding secret-shared unit vector $[\![u_{\alpha_i}]\!]$. They then multiply each unit vector with the secret-shared input vector and obtain the value at the requested index. Figure 5 describes the resulting protocol.

Similarly, Scatter can be implemented by generating one secret-shared unit vector for each index and multiplying it by the corresponding value. The servers then sum the resulting vectors. Figure 6 describes this protocol.

The secret-index-based outsourced Gather has communication complexity $O(nL + LS)$. The first term comes from generating the L unit vectors, while the second term comes from multiplying the unit vectors with the input vector,

Parties: M computation servers $\{P_1, \dots P_M\}$, N data owners $\{C_1, \dots, C_N\}$.
Inputs:
- Each computation server P_k owns a share of $[\![w]\!]_k$ a sparsely populated n-dimensional vector.
- The data owner knows a list of L unique indices $\{\alpha_1, \dots, \alpha_L\}$, with the indices between 1 and n.

Protocol:

Preparation: The data owner shares the indices using Shamir's secret-sharing.

Processing:

1. For each secret-shared index $[\![\alpha]\!]$, the servers generate a secret-shared unit vector $[\![u_\alpha]\!]$ (i.e., u_α is equal to 1 at index α, 0 otherwise) using the secret index protocol by De Hoogh [28].
2. Output $[\![v]\!]$ such that $[\![v_i]\!] = [\![u_{\alpha_i}]\!] \cdot [\![w^T]\!]$.

Fig. 5: Secret-index-based outsourced Gather protocol

whose elements have size S. Similarly, the secret-index-based outsourced Scatter has communication complexity $O(nL + Sn)$. The first term comes from generating the L unit vectors, while the second term comes from multiplying the unit vectors with the element of size S.

4.3 Permutation-based implementations

The secret-index-based protocols have a cost that grows linearly with the number of indices. This cost can become significant for high-dimensional sparse matrices with many non-zero columns. We therefore design alternative Scatter and Gather protocols whose cost is independent of the number of indices.

Our protocols rely on a "secret-shared" permutation, based on the secret shuffling protocol of [53]. The data owner obtains a random permutation π and distributes correlated seeds to the computation servers. The servers can then apply π to a secret-shared vector without learning the permutation itself. This kind of shared permutation has also been used recently by [76]. Figure 7 describes the protocol.

For a permutation π and a vector $y \in \mathcal{V}^n$, we write $\pi(y)$ for the vector obtained by moving the value at position j to position $\pi(j)$. Thus,

$$\big(\pi(y)\big)_{\pi(j)} = y_j,$$

or equivalently,

$$\big(\pi(y)\big)_j = y_{\pi^{-1}(j)}.$$

The data owner generates $K = \binom{M}{t+1}$ random seeds $\{r_1, \dots, r_K\}$ and sends each seed to a distinct subset of $t + 1$ computation servers. For each seed r_k,

Parties: M computation servers $\{P_1, \dots P_M\}$, N data owners $\{C_1, \dots, C_N\}$.
Inputs:
- Each computation server P_k owns a share of $[\![v]\!]_k$ a dense L-dimensional vector.
- The data owner knows a list of L unique indices $\{\alpha_1, \dots, \alpha_L\}$, with the indices between 1 and n.

Protocol:

Preparation: The data owner shares the indices using Shamir's secret-sharing.

Processing:

1. For each secret-shared index $[\![\alpha]\!]$, the servers generate a secret-shared unit vector $[\![u_\alpha]\!]$ (i.e., u_α is equal to 1 at index α, 0 otherwise) using the secret index protocol by De Hoogh [28].
2. Output $[\![w]\!] = \sum_{i=1}^{L} [\![v_i]\!][\![u_{\alpha_i}]\!]$.

Fig. 6: Secret-index-based outsourced Scatter protocol

the corresponding servers derive a permutation π_k, apply it to their local shares, and reshare the permuted vector. The final permutation is

$$\pi = \pi_K \circ \cdots \circ \pi_2 \circ \pi_1.$$

Since the data owner knows all seeds, it can reconstruct π. Each computation server receives a "share" $\langle \pi \rangle_k$ consisting of the seeds sent to subsets containing that server.

Because the adversary corrupts at most $t < M/2$ computation servers, at least one subset S_k contains no corrupted server. The corresponding permutation therefore remains unknown to the adversary. This would not hold without the honest-majority assumption.

The same construction also supports a secret-shared inverse permutation. The servers apply the inverse permutations in reverse order, using

$$\pi^{-1} = \pi_1^{-1} \circ \pi_2^{-1} \circ \cdots \circ \pi_K^{-1}.$$

Figure 8 describes our permutation-based outsourced Gather protocol. The data owner first obtains a secret-shared permutation and publishes the permuted indices. Because the permutation is unknown to the computation servers, the published indices do not reveal the original indices. The servers can nevertheless use the published indices to extract the corresponding elements from the permuted input vector without performing secure indexing.

In particular, let $\beta_i = \pi(\alpha_i)$ and $\widetilde{w} = \pi(w)$. By the definition of the permutation,

$$\widetilde{w}_{\beta_i} = \widetilde{w}_{\pi(\alpha_i)} = w_{\alpha_i}.$$

Thus, the servers can recover the requested values without learning the original indices.

Parties: M computation servers $\{P_1, \ldots, P_M\}$, one data owner.
Inputs:
- The data owner has no input.
- Each computation server P_k owns a share of a dense vector $[\![y]\!]_k$.

Outputs:
- The data owner obtains a permutation π.
- Each computation server P_k owns a share $[\![w]\!]_k$, where $w = \pi(y)$. P_k only knows a "share" $\langle \pi \rangle_k$.

Protocol:

Preparation: For each subset S_k of $t+1$ computation servers, the data owner generates a random seed r_k and sends this seed to the servers in S_k.

Processing:
- The servers initialize the secret-shared output $[\![w]\!]^{(0)}$ to $[\![y]\!]$.
- For k in $\{1, \ldots, \binom{M}{t+1}\}$, the computation servers in S_k (1) derive the permutation π_k from the seed r_k, (2) permute their local shares of $[\![w]\!]^{(k-1)}$ according to π_k, and (3) reshare the permuted vector with all parties to obtain $[\![w]\!]^{(k)} = [\![\pi_k(w^{(k-1)})]\!]$.
- Output $[\![w]\!] = [\![w]\!]^{(K)}$, with $K = \binom{M}{t+1}$.

Fig. 7: "Secret-shared" permutation protocol

Figure 9 describes the corresponding Scatter protocol. The data owner again publishes the permuted indices. The servers construct the permuted sparse vector using these indices and then apply the inverse secret-shared permutation to obtain the final output.

Both protocols have communication complexity $O(nS)$, independent of the number of indices. They therefore become attractive when the number of indices is large. This asymptotic cost hides an important dependency on the number of computation servers. The secret-shared permutation requires $K = \binom{M}{t+1}$ successive resharing steps, and therefore scales exponentially with the number of computation servers. In contrast, the secret-index-based protocols scale polynomially with M. This limits the applicability of the permutation-based protocols to settings with a small number of computation servers.

In practice, many outsourced MPC systems use two [62, 47] or three computation servers [73, 74]. For these settings, the permutation-based construction remains practical.

4.4 Sparse-dense matrix multiplication

We can use the Scatter and Gather protocols to construct a sparse-dense matrix multiplication protocol, shown in Figure 10. The protocol first uses Gather to extract the rows of the dense matrix corresponding to the non-zero columns of

Parties: M computation servers $\{P_1, \ldots, P_M\}$, N data owners $\{C_1, \ldots, C_N\}$.
Inputs:
- Each computation server P_k owns a share of $[\![w]\!]_k$, a sparsely populated n-dimensional vector.
- The data owner knows a list of L unique indices $\{\alpha_1, \ldots, \alpha_L\}$, with the indices between 1 and n.

Protocol:

Preparation:
- Execute SecSharedPerm<prep>() $\rightarrow$ The data owner receives π and each P_k receives a share $\langle \pi \rangle$.
- For each $i \in \{1, \ldots, L\}$, let $\beta_i = \pi(\alpha_i)$. The data owner sends $\{\beta_1, \ldots, \beta_L\}$ to all servers.

Processing:
- Execute SecSharedPerm<proc>($[\![w]\!], \langle \pi \rangle$) $\rightarrow$ each P_k receives a share $[\![\widetilde{w}]\!]_k$, where $\widetilde{w} = \pi(w)$.
- Output $[\![v]\!]$ such that $v_i = \widetilde{w}_{\beta_i}$ for $i \in \{1, \ldots, L\}$.

Fig. 8: Permutation-based outsourced Gather protocol

the sparse matrix. It then multiplies the resulting dense matrix with the non-zero columns of the sparse matrix using an optimized dense matrix multiplication protocol [28].

We describe the protocol for multiplying a horizontally partitioned sparse matrix by a dense matrix. This operation covers several ML models, including logistic regression and principal component analysis. More complex models, such as neural networks, may involve dense tensors rather than matrices. Our approach extends naturally to these settings by using Gather to extract the relevant sub-tensor and then applying the corresponding optimized dense operation.

Matrix-vector multiplication costs Table 1 compares the asymptotic costs of existing and proposed sparse-dense multiplication protocols. The optimized dense protocol [28, 74] has low communication complexity, $O(n)$, but requires $O(np)$ memory and therefore does not exploit sparsity. The sorting-based protocol [26] reduces the memory footprint by exploiting sparsity, but its communication complexity is $O((\text{nnz}(X) + p) \log(\text{nnz}(X) + p))$.

The permutation-based protocol has communication complexity $O(pN + n)$, compared with $O(n)$ for the optimized dense protocol. Its communication overhead therefore grows linearly with the number of data owners. For applications involving a moderate number of data owners, this overhead can remain small.

The permutation-based protocol also has a low memory footprint of $O(\text{spsize}_* + p + n)$. This is slightly larger than the $O(\text{nnz}(X) + p + n)$ footprint of the sorting-based protocol because our protocol exploits non-zero columns rather than indi-

Parties: M computation servers $\{P_1, \ldots, P_M\}$, N data owners $\{C_1, \ldots, C_N\}$.
Inputs:
- Each computation server P_k owns a share of $[\![v]\!]_k$, a dense L-dimensional vector.
- The data owner knows a list of L unique indices $\{\alpha_1, \ldots, \alpha_L\}$, with the indices between 1 and n.

Protocol:

Preparation:
- Execute SecSharedPerm<prep>() $\rightarrow$ The data owner receives π and each P_k receives a share $\langle \pi \rangle$.
- For each $i \in \{1, \ldots, L\}$, let $\beta_i = \pi(\alpha_i)$. The data owner sends $\{\beta_1, \ldots, \beta_L\}$ to all servers.

Processing:
- The servers build $[\![\widetilde{w}]\!]$ such that $[\![\widetilde{w}]\!]_{\beta_i} = [\![v_i]\!]$ for $i \in \{1, \ldots, L\}$, and $[\![\widetilde{w}]\!]_j = 0$ otherwise.
- Execute SecSharedInvertPerm<proc>($[\![\widetilde{w}]\!], \langle \pi \rangle$) $\rightarrow$ each P_k receives a share $[\![w]\!]_k$.
- Output $[\![w]\!]$.

Fig. 9: Permutation-based outsourced Scatter protocol

vidual non-zero elements. As discussed in Section 3.4, this weaker exploitation of sparsity also results in a less detailed leakage profile.

The secret-index-based protocol provides a different trade-off. It is preferable when the number of data owners is large, while the permutation-based protocol is preferable when the number of non-zero columns is large. Thus, neither protocol dominates the other for all parameter regimes.

Matrix-matrix multiplication costs For simplicity, Table 1 focuses on matrix-vector multiplication. However, it is also useful to consider the multiplication of an $n \times p$ sparse matrix with a $p \times q$ dense matrix. The costs of the state-of-the-art protocols simply scale by a factor of q compared with the matrix-vector case. Our permutation-based protocol has the same linear dependency on q. The secret-index-based protocol has slightly better scaling: its matrix-vector communication cost is $O(p \log(p)\, \mathrm{nnzc}_* + \mathrm{spsize}_*)$, while its matrix-matrix communication cost is $O((p \log(p) + pq)\, \mathrm{nnzc}_* + q \cdot \mathrm{spsize}_*)$. This difference comes from the fact that the secret-indexing protocol is executed once for all columns of Y. If Y has many columns, this creates an amortized cost. Hence, the secret-index-based protocol can be cheaper than the permutation-based protocol for sufficiently large models, where the dense matrix typically represents the ML model.

Row-sparse matrices The sparse-dense multiplication protocol in Figure 10 only relies on outsourced Gather. Nevertheless, outsourced Scatter is also essential for implementing many other sparse operations. For example, training a logistic

Parties: M computation servers $\{P_1, \dots P_M\}$, N data owners $\{C_1, \dots, C_N\}$.
Inputs:
- Each computation server P_k owns a share of an $p \times q$ dense matrix Y: $[\![Y]\!]_k$. Let Y be stored as a vector of matrix rows.
- Each data owner C_l owns an $n_l \times p$ sparse matrix $X^{(l)}$.

Output: an $n \times q$ matrix $Z = X \cdot Y$
Protocol:

Preparation: Each data owner C_l performs the following operations:
1. Let $\mathcal{A}^{(l)}$ be the indices of non-zero columns in $X^{(l)}$ and $X_r^{(l)}$ be the sub-matrix containing only the non-zero columns of $X^{(l)}$.
2. Execute Gather<prep>$(\mathcal{A}^{(l)}) \rightarrow$ each P_k receives a list of index shares $\left\langle \mathcal{A}^{(l)} \right\rangle_k$.
3. C_l shares $X_r^{(l)}$ using Shamir's Secret Sharing $\rightarrow$ each P_k receives a share $[\![X_r^{(l)}]\!]$.

Processing:
1. For each C_l, the servers execute Gather<proc>$(\left\langle \mathcal{A}^{(l)} \right\rangle, [\![Y]\!]) \rightarrow$ each P_k receives a share $[\![Y_r^{(l)}]\!]$.
2. For each C_l, the servers compute the $n_l \times q$ matrix $[\![Z^{(l)}]\!] = [\![X_r^{(l)}]\!] \cdot [\![Y_r^{(l)}]\!]$ using the optimized matrix multiplication by De Hoogh [28].
3. Output $[\![Z]\!]$, the vertical concatenation of $\{[\![Z]\!]_1, \dots, [\![Z]\!]_N\}$.

Fig. 10: Secure protocol for sparse-dense matrix multiplications

regression model requires two matrix-vector multiplications [67]: the multiplication of X with a dense parameter vector and the multiplication of X^T with a loss vector. Our sparse-dense multiplication protocol directly covers the first multiplication. The second multiplication requires a different algorithm because the transposed matrix is vertically partitioned and is row-sparse.

We can construct this multiplication similarly to Figure 10. We first multiply the non-zero rows of X^T with the dense vector and then scatter the resulting values. The resulting performance gains are comparable to those of the protocol described in Figure 10.

Towards other sparse multiplications As motivated in Section 2.2, our paper focuses on sparse-dense multiplication because it is an important operation in ML algorithms. More generally, Scatter and Gather are standard building blocks in sparse linear algebra libraries [30]. Scatter and Gather therefore provide a general interface for extending our approach to other sparse operations.

Protocols	Dense [28, 61, 74]	Sorting-based sparse mult. [26]	Perm.-based (**Ours**)	Ind.-based (**Ours**)
Comm.	$O(n)$	$O((\text{nnz}(X)+p)\log(\text{nnz}(X)+p))$	$O(pN+n)$	$O(p\log(p)\,\text{nnzc}_*+n)$
Comp.	$O(np)$	$O((\text{nnz}(X)+p)\log(\text{nnz}(X)+p))$	$O(pN+\text{spsize}_*)$	$O(p\log(p)\,\text{nnzc}_*+\text{spsize}_*)$
Rounds	$O(1)$	$O(\log(\text{nnz}(X)+p))$	$O(1)$	$O(\log(p))$
Memory	$O(np+p+n)$	$O(\text{nnz}(X)+p+n)$	$O(\text{spsize}_*+p+n)$	$O(\text{spsize}_*+p+n)$

Table 1: Comparison of the (outsourced) secure multiplication protocols for the multiplication of an $n \times p$ sparse matrix X (horizontally partitioned between N data owners) with a p-dimensional dense vector y, with $\text{nnzc}_* = \sum_{l=1}^{N} \text{nnzc}(X^{(l)})$ and $\text{spsize}_* = \sum_{l=1}^{N} n_l\,\text{nnzc}(X^{(l)})$

5 Extensions

5.1 DPF-based scatter/gather

Section 4.2 presented secure Scatter and Gather protocols based on the secret-indexing protocol of De Hoogh [28]. These protocols use secret indexing to transform a secret-shared index $[\![i]\!]$ into a secret-shared n-dimensional unit vector $[\![v]\!]$ such that $v_i = 1$ and $v_j = 0$ for all $j \neq i$. This transformation requires $O(n)$ communication.

Distributed Point Functions (DPFs) [12–14, 17, 24, 25, 27, 39, 40, 81] provide a potential alternative. Instead of generating a secret-shared unit vector explicitly, a DPF can represent such a vector compactly and allow its entries to be evaluated locally by the computation parties. A DPF represents a point function $f(x) = \beta$ if $x = \alpha$, 0 otherwise.

To share such a function, the function owner uses a p-party DPF scheme to generate p keys and distributes one key to each shareholder. Each shareholder can then locally evaluate its share of the function at any point x by computing $f(x; K_j)$. In most existing schemes, including [13, 14, 27, 39], the function shares are additive, such that $f(x) = \sum_{j=1}^{p} f(x; K_j)$.

A DPF-based implementation of Scatter and Gather would therefore avoid sharing the index itself with the computation servers. Instead, the data owner would generate DPF keys and distribute them to the servers. The servers could then locally evaluate their DPF keys to obtain shares of the corresponding unit vector, avoiding the $O(n)$ communication required by explicit secret indexing.

DPF schemes nevertheless require further improvements to become a practical alternative for our applications, particularly PPML. Boyle et al. [13, 14] described DPF schemes with logarithmic key size, but these constructions are two-party schemes and therefore do not directly support our honest-majority setting. More recently, Zyskind et al. [81] achieved similar asymptotic complexity for three parties. However, existing p-party DPF schemes are substantially less efficient [17, 24, 25, 27, 40]. Their key sizes range from $O(\sqrt{N})$ to $O(N^{1/4})$, where N is the function domain size, with different trade-offs. For example, some constructions have an exponential dependency on the number of parties

[17], while others target more restricted applications [24]. We therefore leave a detailed evaluation of DPF-based Scatter and Gather for future work.

5.2 DORAM-based scatter/gather

Distributed Oblivious RAM (DORAM) [16, 29, 35, 57, 79] provides a RAM-like data structure for performing oblivious read and write operations within MPC. As already highlighted by Damie et al. [26], DORAM can be used to implement sparse-dense operations and, in particular, provides natural implementations of Scatter and Gather.

In a DORAM-based implementation, the data owner would provide secret-shared indices as in our secret-index-based protocols. For Gather, the servers would store the dense vector in the DORAM and perform oblivious reads at the secret-shared indices. For Scatter, the servers would initialize the DORAM with a zero vector and perform oblivious writes at the secret-shared indices.

DORAM-based protocols are therefore a promising alternative. However, the efficiency of existing DORAM schemes remains limited [10], although ongoing work continues to improve their performance [35, 36, 49]. Moreover, many DORAM constructions target two-party settings [29, 57, 79] and therefore do not directly support our honest-majority setting. Recent works have introduced three-party DORAM protocols [16, 35, 36, 49], but efficient DORAM constructions for anarbitrary number of parties remain an open direction. We leave a detailed comparison of DORAM-based sparse operations for future work.

Finally, Hamlin and Varia [44] presented DORAM schemes optimized either for reads or for writes. This distinction could be particularly beneficial for sparse operations: DORAM-based Gather requires only read operations, whereas DORAM-based Scatter requires only write operations.

5.3 Alternative threat models

Our paper focuses on semi-honest security with an honest-majority assumption. The outsourced construction can, however, be instantiated under other threat models. Our protocols rely primarily on three building blocks: Scatter, Gather, and optimized dense matrix multiplication. We therefore discuss how these building blocks can be implemented under different security assumptions.

Dishonest majority Secure matrix multiplication can be performed under a dishonest-majority assumption [26, 62], but it does not benefit from the same performance advantages provided by honest-majority MPC. For example, multiplying two n-dimensional vectors requires $O(n)$ communication under dishonest majority, whereas it can require only $O(1)$ communication under honest majority. Thus, although dishonest-majority MPC is possible, the honest-majority assumption provides significant performance advantages for our applications.

The compatibility of each Scatter and Gather construction with dishonest majority must also be considered separately. Our permutation-based protocols cannot be directly applied because the secret-shared permutation relies on the

honest-majority assumption. One possible alternative is for the data owner to generate and share a switching network [56]. Similarly, our secret-index-based protocols cannot be directly applied because the secret-indexing protocol relies on Shamir's secret sharing [28, 66].

DPF-based protocols provide a more direct route to dishonest-majority settings. Two-party DPF schemes can be used directly [14], while multiparty constructions such as [13] provide a potential generalization to more parties. DORAM-based protocols can likewise be instantiated in two-party dishonest-majority settings using existing two-party DORAM schemes [29, 57, 79].

Malicious security Optimized matrix multiplication protocols supporting malicious security under honest majority also exist [63, 74]. Thus, the dense matrix multiplication component of our construction can be instantiated with maliciously secure protocols.

As in the dishonest-majority setting, the remaining building blocks must be considered separately. Our permutation-based protocol is derived from the oblivious shuffling protocol of Laur et al. [53]. Laur et al. also describe a maliciously secure variant of their shuffling protocol, which could therefore be used to construct a maliciously secure permutation-based Scatter and Gather protocol.

For the secret-index-based protocols, De Hoogh [28] does not provide a maliciously secure secret-indexing protocol. Consequently, our construction cannot be directly extended to malicious security without replacing this building block.

For DPF-based protocols, De Castro and Polychroniadou [18] describe a verifiable two-party DPF scheme. This construction does not directly support our honest-majority setting, but a suitable multiparty verifiable DPF could provide a route to maliciously secure DPF-based Scatter and Gather.

For DORAM-based protocols, several works [34, 49] describe three-party DORAM schemes with malicious security. These constructions could therefore be used as building blocks for maliciously secure Scatter and Gather.

Finally, even when maliciously secure building blocks are available, additional care is required to compose them into a secure outsourced protocol. In particular, the interfaces between the preparation and processing phases must preserve both correctness and the required security properties. We leave a complete maliciously secure instantiation to future work.

Conclusion

We presented SparseMPC, an approach for securely processing sparse data in an outsourced MPC setting. Our approach uses outsourced Scatter and Gather as general building blocks for sparse linear algebra. We designed two constructions for these operations based on secret indexing and secret-shared permutations. The secret-index-based construction has a communication cost that grows with the number of requested indices, while the permutation-based construction has communication complexity independent of this number. The permutation-based

construction therefore becomes attractive for highly sparse data with many non-zero elements, whereas the secret-index-based construction can be preferable when the number of data owners or the number of requested indices is large. We used these building blocks to construct a sparse-dense matrix multiplication protocol and analyzed its communication, computation, and memory costs.

Several directions remain open to further improve SparseMPC. In particular, more efficient multi-party DPF and DORAM constructions could provide alternative implementations of outsourced Scatter and Gather. It would also be interesting to extend our protocols to malicious security and dishonest-majority settings without sacrificing their efficiency advantages.

Acknowledgments. A Large Language Model was carefully used to polish the writing style and grammar of this paper.

References

1. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) (Apr 2016), http://data.europa.eu/eli/reg/2016/679/oj/eng, legislative Body: EP, CONSIL
2. Araki, T., Furukawa, J., Ohara, K., Pinkas, B., Rosemarin, H., Tsuchida, H.: Secure Graph Analysis at Scale. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 610–629. ACM, Virtual Event Republic of Korea (Nov 2021). https://doi.org/10.1145/3460120.3484560, https://dl.acm.org/doi/10.1145/3460120.3484560
3. Asharov, G., Hamada, K., Ikarashi, D., Kikuchi, R., Nof, A., Pinkas, B., Takahashi, K., Tomida, J.: Efficient Secure Three-Party Sorting with Applications to Data Analysis and Heavy Hitters. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 125–138. CCS '22, Association for Computing Machinery, New York, NY, USA (Nov 2022). https://doi.org/10.1145/3548606.3560691, https://doi.org/10.1145/3548606.3560691
4. Asi, H., Feldman, V., Keller, H., Rothblum, G.N., Talwar, K.: PREAMBLE: Private and Efficient Aggregation of Block Sparse Vectors and Applications (2025), https://eprint.iacr.org/2025/490, publication info: Preprint.
5. Bank, R., Douglas, C.: Sparse Matrix Multiplication Package (SMMP). Advances in Computational Mathematics **1** (May 2001). https://doi.org/10.1007/BF02070824
6. Beguier, C., Andreux, M., Tramel, E.W.: Efficient Sparse Secure Aggregation for Federated Learning (Oct 2021), http://arxiv.org/abs/2007.14861, arXiv:2007.14861 [cs, stat]
7. Bell, J., Gascón, A., Ghazi, B., Kumar, R., Manurangsi, P., Raykova, M., Schoppmann, P.: Distributed, Private, Sparse Histograms in the Two-Server Model. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 307–321. CCS '22, Association for Computing Machinery, New York, NY, USA (Nov 2022). https://doi.org/10.1145/3548606.3559383, https://dl.acm.org/doi/10.1145/3548606.3559383

8. Bennett, J., Lanning, S., others: The netflix prize. In: Proceedings of KDD cup and workshop. vol. 2007, p. 35. Citeseer (2007)
9. Bitar, R., Egger, M., Wachter-Zeh, A., Xhemrishi, M.: Sparsity and Privacy in Secret Sharing: A Fundamental Trade-Off (Aug 2023), http://arxiv.org/abs/2308.06413, arXiv:2308.06413 [cs, math]
10. Blanton, M., Yuan, C.: Binary Search in Secure Computation. In: Proceedings 2022 Network and Distributed System Security Symposium. Internet Society, San Diego, CA, USA (2022). https://doi.org/10.14722/ndss.2022.23106, https://www.ndss-symposium.org/wp-content/uploads/2022-106-paper.pdf
11. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical Secure Aggregation for Privacy-Preserving Machine Learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191. ACM, Dallas Texas USA (Oct 2017). https://doi.org/10.1145/3133956.3133982, https://dl.acm.org/doi/10.1145/3133956.3133982
12. Boyle, E., Gilboa, N., Hamilis, M., Ishai, Y., Tu, Y.: Improved Constructions for Distributed Multi-Point Functions. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 2414–2432. IEEE Computer Society (2025). https://doi.org/10.1109/SP61157.2025.00044
13. Boyle, E., Gilboa, N., Ishai, Y.: Function Secret Sharing. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 337–367. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
14. Boyle, E., Gilboa, N., Ishai, Y.: Function Secret Sharing: Improvements and Extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303. ACM, Vienna Austria (Oct 2016). https://doi.org/10.1145/2976749.2978429, https://dl.acm.org/doi/10.1145/2976749.2978429
15. Bukaty, P.: The California Consumer Privacy Act (CCPA): An implementation guide. IT Governance Publishing (2019). https://doi.org/10.2307/j.ctvjghvnn, https://www.jstor.org/stable/j.ctvjghvnn
16. Bunn, P., Katz, J., Kushilevitz, E., Ostrovsky, R.: Efficient 3-Party Distributed ORAM. In: Security and Cryptography for Networks, vol. 12238, pp. 215–232. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-57990-6_11, https://link.springer.com/10.1007/978-3-030-57990-6_11, series Title: Lecture Notes in Computer Science
17. Bunn, P., Kushilevitz, E., Ostrovsky, R.: CNF-FSS and Its Applications. In: Public-Key Cryptography – PKC 2022, vol. 13177, pp. 283–314. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-97121-2_11, https://link.springer.com/10.1007/978-3-030-97121-2_11, series Title: Lecture Notes in Computer Science
18. de Castro, L., Polychroniadou, A.: Lightweight, Maliciously Secure Verifiable Function Secret Sharing. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022. pp. 150–179. Lecture Notes in Computer Science, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-06944-4_6
19. Cha, M., Mislove, A., Gummadi, K.P.: A measurement-driven analysis of information propagation in the flickr social network. In: Proceedings of the 18th international conference on World wide web. pp. 721–730. ACM, Madrid Spain (Apr 2009). https://doi.org/10.1145/1526709.1526806, https://dl.acm.org/doi/10.1145/1526709.1526806

20. Chen, C., Zhou, J., Wang, L., Wu, X., Fang, W., Tan, J., Wang, L., Liu, A.X., Wang, H., Hong, C.: When Homomorphic Encryption Marries Secret Sharing: Secure Large-Scale Sparse Logistic Regression and Applications in Risk Control. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 2652–2662. KDD '21, Association for Computing Machinery, New York, NY, USA (Aug 2021). https://doi.org/10.1145/3447548.3467210, https://doi.org/10.1145/3447548.3467210
21. Chowdhury, M.N., Bauer, A., Zhou, M.: Efficient Privacy-Preserving Recommendation on Sparse Data using Fully Homomorphic Encryption (Sep 2025). https://doi.org/10.48550/arXiv.2509.03024, http://arxiv.org/abs/2509.03024, arXiv:2509.03024 [cs]
22. Cui, J., Chen, C., Lyu, L., Yang, C., Li, W.: Exploiting Data Sparsity in Secure Cross-Platform Social Recommendation. Advances in Neural Information Processing Systems **34** (2021)
23. Damie, M.: Privacy-Preserving Computations on Sparse Data. Ph.D. thesis, Université de Lille (2025)
24. Damie, M., Hahn, F., Peter, A., Ramon, J.: Ddh-based schemes for multi-party function secret sharing. In: Secure IT Systems. pp. 3–22. Springer Nature Switzerland, Cham (2026)
25. Damie, M., Hahn, F., Peter, A., Ramon, J.: Eliminating exponential key growth in prg-based distributed point functions. In: Computer Security. ESORICS 2025 International Workshops. pp. 116–124. Springer Nature Switzerland, Cham (2026)
26. Damie, M., Hahn, F., Peter, A., Ramon, J.: Secure sparse matrix multiplications and their applications to privacy-preserving machine learning. In: Proceedings of the Sixteenth ACM Conference on Data and Application Security and Privacy. p. 71–84. CODASPY '26, Association for Computing Machinery, New York, NY, USA (2026). https://doi.org/10.1145/3800506.3803493, https://doi.org/10.1145/3800506.3803493
27. Damie, M., Mazzone, F., Hahn, F., Peter, A., Ramon, J.: Noisy function secret sharing and its applications to differentially private computations. Cryptology ePrint Archive, Paper 2025/1937 (2025), https://eprint.iacr.org/2025/1937
28. De Hoogh, S.: Design of large scale applications of secure multiparty computation : secure linear programming. PhD Thesis, Technische Universiteit Eindhoven, Eindhoven (2012). https://doi.org/10.6100/IR735328, iSBN: 9789038632032
29. Doerner, J., Shelat, A.: Scaling ORAM for Secure Computation. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 523–535. CCS '17, Association for Computing Machinery, New York, NY, USA (Oct 2017). https://doi.org/10.1145/3133956.3133967, https://doi.org/10.1145/3133956.3133967
30. Duff, I.S., Heroux, M.A., Pozo, R.: An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum. ACM Transactions on Mathematical Software **28**(2), 239–267 (Jun 2002). https://doi.org/10.1145/567806.567810, https://doi.org/10.1145/567806.567810
31. Duff, I.: A survey of sparse matrix research. Proceedings of the IEEE **65**(4), 500–535 (Apr 1977). https://doi.org/10.1109/PROC.1977.10514, conference Name: Proceedings of the IEEE
32. Egger, M., Xhemrishi, M., Wachter-Zeh, A., Bitar, R.: Sparse and Private Distributed Matrix Multiplication with Straggler Tolerance (Jun 2023), http://arxiv.org/abs/2306.15134, arXiv:2306.15134 [cs, math]

33. Evans, D., Kolesnikov, V., Rosulek, M.: A Pragmatic Introduction to Secure Multi-Party Computation. Foundations and Trends® in Privacy and Security **2**(2-3), 70–246 (Dec 2018). https://doi.org/10.1561/3300000019, https://www.nowpublishers.com/article/Details/SEC-019, publisher: Now Publishers, Inc.
34. Falk, B., Noble, D., Ostrovsky, R., Shtepel, M., Zhang, J.: DORAM revisited: Maliciously secure RAM-MPC with logarithmic overhead (2023), https://eprint.iacr.org/2023/578, report Number: 578
35. Falk, B., Ostrovsky, R., Shtepel, M., Zhang, J.: GigaDORAM: breaking the billion address barrier. In: Proceedings of the 32nd USENIX Conference on Security Symposium. pp. 3871–3888. SEC '23, USENIX Association, USA (Aug 2023)
36. Falk, B.H., Noble, D., Ostrovsky, R.: MetaDORAM: Info-Theoretic Distributed ORAM with Less Communication (2024), https://eprint.iacr.org/2024/011, publication info: Preprint.
37. Ferguson, A., Gibson, P., D'Agata, L., McLeod, P., Yaman, F., Das, A., Colbert, I., Cano, J.: Exploiting Unstructured Sparsity in Fully Homomorphic Encrypted DNNs (Mar 2025). https://doi.org/10.48550/arXiv.2503.09184, http://arxiv.org/abs/2503.09184, arXiv:2503.09184 [cs]
38. Gale, T., Elsen, E., Hooker, S.: The State of Sparsity in Deep Neural Networks (Feb 2019), http://arxiv.org/abs/1902.09574, arXiv:1902.09574 [cs, stat]
39. Gilboa, N., Ishai, Y.: Distributed Point Functions and Their Applications. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. pp. 640–658. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
40. Goel, A., Wang, M., Wang, Z.: Multiparty Distributed Point Functions. In: Tauman Kalai, Y., Kamara, S.F. (eds.) Advances in Cryptology – CRYPTO 2025. pp. 140–173. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-032-01884-7_5
41. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press (Sep 2009), google-Books-ID: LccLAQAAQBAJ
42. Guyon, I., Gunn, S., Ben-Hur, A., Dror, G.: Dorothea (2008). https://doi.org/10.24432/C5NK6X, https://archive.ics.uci.edu/dataset/169, published: UCI Machine Learning Repository
43. Hamada, K., Ikarashi, D., Chida, K., Takahashi, K.: Oblivious Radix Sort: An Efficient Sorting Algorithm for Practical Secure Multi-party Computation (2014), https://eprint.iacr.org/2014/121, report Number: 121
44. Hamlin, A., Varia, M.: Two-Server Distributed ORAM with Sublinear Computation and Constant Rounds. In: Public-Key Cryptography – PKC 2021, vol. 12711, pp. 499–527. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_18, https://link.springer.com/10.1007/978-3-030-75248-4_18, series Title: Lecture Notes in Computer Science
45. Harper, F.M., Konstan, J.A.: The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems **5**(4), 1–19 (Jan 2016). https://doi.org/10.1145/2827872, https://dl.acm.org/doi/10.1145/2827872
46. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural Collaborative Filtering. In: Proceedings of the 26th International Conference on World Wide Web. pp. 173–182. International World Wide Web Conferences Steering Committee, Perth Australia (Apr 2017). https://doi.org/10.1145/3038912.3052569, https://dl.acm.org/doi/10.1145/3038912.3052569

47. Hegde, A., Möllering, H., Schneider, T., Yalame, H.: SoK: Efficient Privacy-preserving Clustering. Proceedings on Privacy Enhancing Technologies **2021**(4), 225–248 (Oct 2021). https://doi.org/10.2478/popets-2021-0068, https://petsymposium.org/popets/2021/popets-2021-0068.php
48. Huang, R., Xia, M., Nguyen, D.T., Zhao, T., Sakamuru, S., Zhao, J., Shahane, S.A., Rossoshek, A., Simeonov, A.: Tox21Challenge to Build Predictive Models of Nuclear Receptor and Stress Response Pathways as Mediated by Exposure to Environmental Chemicals and Drugs. Frontiers in Environmental Science **3** (2016), https://www.frontiersin.org/articles/10.3389/fenvs.2015.00085
49. Ichikawa, A., Komargodski, I., Hamada, K., Kikuchi, R., Ikarashi, D.: 3-Party Secure Computation for RAMs: Optimal and Concretely Efficient (2023), https://eprint.iacr.org/2023/516, report Number: 516
50. Konečný, J., McMahan, H.B., Ramage, D., Richtárik, P.: Federated Optimization: Distributed Machine Learning for On-Device Intelligence (Oct 2016), https://arxiv.org/abs/1610.02527v1
51. Koti, N., Kukkala, V.B., Patra, A., Raj Gopal, B.: Graphiti: Secure Graph Computation Made More Scalable. In: Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. pp. 4017–4031. CCS '24, Association for Computing Machinery, New York, NY, USA (Dec 2024). https://doi.org/10.1145/3658644.3670393, https://dl.acm.org/doi/10.1145/3658644.3670393
52. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. pp. 2651–2668 (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/koti
53. Laur, S., Willemson, J., Zhang, B.: Round-Efficient Oblivious Database Manipulation. In: Lai, X., Zhou, J., Li, H. (eds.) Information Security. pp. 262–277. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24861-0_18
54. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection (2014)
55. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse Convolutional Neural Networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 806–814 (2015)
56. Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., Miller, A.: HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 887–903. CCS '19, Association for Computing Machinery, New York, NY, USA (Nov 2019). https://doi.org/10.1145/3319535.3354238, https://doi.org/10.1145/3319535.3354238
57. Lu, S., Ostrovsky, R.: Distributed Oblivious RAM for Secure Two-Party Computation. In: Sahai, A. (ed.) Theory of Cryptography. pp. 377–396. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_22
58. Lu, T., Zhang, B., Li, L., Ren, K.: Aegis: A Lightning Fast Privacy-preserving Machine Learning Platform against Malicious Adversaries (2023), https://eprint.iacr.org/2023/1890, publication info: Preprint.
59. Mattson, T., Bader, D., Berry, J., Buluc, A., Dongarra, J., Faloutsos, C., Feo, J., Gilbert, J., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C., Lumsdaine, A., Padua, D., Poole, S., Reinhardt, S., Stone-

braker, M., Wallach, S., Yoo, A.: Standards for graph algorithm primitives. In: 2013 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–2 (Sep 2013). https://doi.org/10.1109/HPEC.2013.6670338, https://ieeexplore.ieee.org/abstract/document/6670338

60. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR, Sydney, Australia (Apr 2017), https://proceedings.mlr.press/v54/mcmahan17a.html
61. Mohassel, P., Rindal, P.: ABY^3: A Mixed Protocol Framework for Machine Learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 35–52. CCS '18, Association for Computing Machinery, New York, NY, USA (Oct 2018). https://doi.org/10.1145/3243734.3243760, https://doi.org/10.1145/3243734.3243760
62. Mohassel, P., Zhang, Y.: SecureML: A System for Scalable Privacy-Preserving Machine Learning. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 19–38 (May 2017). https://doi.org/10.1109/SP.2017.12, iSSN: 2375-1207
63. Patra, A., Suresh, A.: BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In: Proceedings 2020 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2020). https://doi.org/10.14722/ndss.2020.24202, https://www.ndss-symposium.org/wp-content/uploads/2020/02/24202.pdf
64. Rennie, J., Lang, K.: The 20 Newsgroups data set (2008), http://qwone.com/ jason/20Newsgroups/
65. Sangers, A., van Heesch, M., Attema, T., Veugen, T., Wiggerman, M., Veldsink, J., Bloemen, O., Worm, D.: Secure Multiparty PageRank Algorithm for Collaborative Fraud Detection. In: Goldberg, I., Moore, T. (eds.) Financial Cryptography and Data Security. pp. 605–623. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_35
66. Schoenmakers, B.: MPyC—Python package for secure multiparty computation. In: Workshop on the Theory and Practice of MPC. https://github. com/lschoe/mpyc (2018)
67. Schoppmann, P., Gascón, A., Raykova, M., Pinkas, B.: Make Some ROOM for the Zeros: Data Sparsity in Secure Distributed Machine Learning. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1335–1350. CCS '19, Association for Computing Machinery, New York, NY, USA (Nov 2019). https://doi.org/10.1145/3319535.3339816, https://doi.org/10.1145/3319535.3339816
68. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
69. Shokri, R., Shmatikov, V.: Privacy-Preserving Deep Learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1310–1321. ACM, Denver Colorado USA (Oct 2015). https://doi.org/10.1145/2810103.2813687, https://dl.acm.org/doi/10.1145/2810103.2813687
70. Srinivas, S., Subramanya, A., Venkatesh Babu, R.: Training Sparse Neural Networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 138–145 (2017)

71. Tibshirani, R.: Regression Shrinkage and Selection Via the Lasso. Journal of the Royal Statistical Society: Series B (Methodological) **58**(1), 267–288 (Jan 1996). https://doi.org/10.1111/j.2517-6161.1996.tb02080.x, https://doi.org/10.1111/j.2517-6161.1996.tb02080.x
72. Tibshirani, R., Wasserman, L.: Sparsity and the lasso. Statistical machine learning pp. 1–15 (2015), publisher: Citeseer
73. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-Party Secure Computation for Neural Network Training. Proc. Priv. Enhancing Technol. **2019**(3), 26–49 (2019)
74. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. arXiv:2004.02229 [cs] (Sep 2020), http://arxiv.org/abs/2004.02229, arXiv: 2004.02229
75. Xhemrishi, M., Bitar, R., Wachter-Zeh, A.: Distributed Matrix-Vector Multiplication with Sparsity and Privacy Guarantees. In: 2022 IEEE International Symposium on Information Theory (ISIT). pp. 1028–1033 (Jun 2022). https://doi.org/10.1109/ISIT50566.2022.9834805, iSSN: 2157-8117
76. Xu, S., Zheng, Y., Hua, Z.: Harnessing Sparsification in Federated Learning: A Secure, Efficient, and Differentially Private Realization (2025). https://doi.org/10.48550/ARXIV.2511.07123, https://arxiv.org/abs/2511.07123, version Number: 1
77. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated Machine Learning: Concept and Applications. ACM Transactions on Intelligent Systems and Technology **10**(2), 12:1–12:19 (Jan 2019). https://doi.org/10.1145/3298981, https://doi.org/10.1145/3298981
78. Yu, J., Chen, K., Fan, X., Chen, Y., Zhu, X., Chen, W.: Lodia: Towards Optimal Sparse Matrix-Vector Multiplication for Batched Fully Homomorphic Encryption (2025), https://eprint.iacr.org/2025/1425, publication info: Published elsewhere. Minor revision. ACM CCS 2025
79. Zahur, S., Wang, X., Raykova, M., Gascón, A., Doerner, J., Evans, D., Katz, J.: Revisiting Square-Root ORAM: Efficient Random Access in Multi-party Computation. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 218–234 (May 2016). https://doi.org/10.1109/SP.2016.21, iSSN: 2375-1207
80. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: Proceedings of the 14th international conference on World Wide Web. pp. 22–32. WWW '05, Association for Computing Machinery, New York, NY, USA (May 2005). https://doi.org/10.1145/1060745.1060754, https://dl.acm.org/doi/10.1145/1060745.1060754
81. Zyskind, G., Yanai, A., Pentland, A.S.: High-Throughput Three-Party DPFs with Applications to ORAM and Digital Currencies. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (2024)

A Security proof sketch

This appendix sketches the security argument for our protocols. We consider the semi-honest model and use the real-ideal paradigm [33]. We first consider the secret-index-based Gather protocol (Fig. 5) and then explain how the argument extends to the other protocols.

Our goal is to show that the secret-index-based Gather protocol securely realizes the ideal functionality described in Fig. 4. As in [26], the proof relies on the composition theorem for the semi-honest model (Theorem 7.3.3 of [41]). The subprotocols used by our construction have already been shown to be correct and secure in the semi-honest model. Consequently, by the composition theorem, it suffices to argue that their composition implements the desired functionality.

Correctness of secret-index-based Gather For each requested index α_i, the secret-index protocol produces a secret-shared unit vector $[\![u_{\alpha_i}]\!]$ satisfying

$$(u_{\alpha_i})_j = \begin{cases} 1 & \text{if } j = \alpha_i, \\ 0 & \text{otherwise.} \end{cases}$$

The protocol then computes the inner product

$$[\![v_i]\!] = [\![u_{\alpha_i}]\!] \cdot [\![w]\!].$$

Therefore,

$$v_i = \sum_{j=1}^{n} (u_{\alpha_i})_j w_j = w_{\alpha_i}.$$

Thus, for every $i \in \{1, \ldots, L\}$, the protocol outputs $v_i = w_{\alpha_i}$, exactly as specified by the ideal Gather functionality.

Security of secret-index-based Gather For each secret-shared index $[\![\alpha_i]\!]$, the first step of the protocol invokes the secret-index protocol of De Hoogh [28]. This protocol securely generates a secret-shared unit vector without revealing α_i to the computation servers. The second step computes the inner product between this unit vector and the secret-shared input vector. This operation consists only of secure arithmetic operations, whose security follows from the underlying secret-sharing-based MPC protocol. Hence, each subprotocol can be replaced by its corresponding ideal functionality without revealing any additional information. By the composition theorem, the resulting protocol securely realizes the outsourced Gather functionality in the semi-honest model.

Secret-index-based Scatter The security argument for the secret-index-based Scatter protocol (Fig. 6) is analogous. For each index α_i, the protocol generates a secret-shared unit vector $[\![u_{\alpha_i}]\!]$ and computes

$$[\![w]\!] = \sum_{i=1}^{L} [\![v_i]\!][\![u_{\alpha_i}]\!].$$

Because the indices $\alpha_1, \ldots, \alpha_L$ are unique, the resulting vector satisfies

$$w_j = \begin{cases} v_i & \text{if } j = \alpha_i, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the protocol is correct. Its security follows from the security of the secret-index protocol and the secure arithmetic operations, together with the composition theorem.

Permutation-based protocols The permutation-based protocols additionally rely on the secret-shared permutation protocol described in Fig. 7. This protocol is based on the secret-shared shuffling technique of Laur et al. [53]. For each resharing step, a subset of computation servers applies a permutation derived from a seed known only to that subset and then reshares the resulting vector. Because at most $t < M/2$ servers are corrupted, at least one of the subsets contains no corrupted server. The corresponding permutation is therefore unknown to the adversary. The security of the resulting secret-shared permutation follows from the security argument of the underlying shuffling construction [53]. A related construction was also used by [76].

Given the security of the secret-shared permutation, the security of the permutation-based Gather and Scatter protocols follows by composition. The data owner additionally provides the permuted index list during the preparation phase. Since the permutation is hidden from the computation servers, these indices do not reveal the original indices.

Sparse-dense matrix multiplication Finally, our sparse-dense matrix multiplication protocol combines the outsourced Gather protocol with secure matrix multiplication. The security of the Gather subprotocol follows from the arguments above. The matrix multiplication is implemented using a secure semi-honest MPC protocol [28]. The composition theorem therefore implies that the complete sparse-dense matrix multiplication protocol securely realizes the corresponding functionality in the semi-honest model.