Secure and Lightweight Feature Selection for Horizontal Federated Learning

Xiaoyuan Liu, *Student Member, IEEE*, Hongwei Li (Corresponding author), *Senior Member, IEEE*, Guowen Xu, *Member, IEEE*, Xilin Zhang, *Member, IEEE*, Tianwei Zhang, *Member, IEEE*, Jianying Zhou

***Abstract*—In this paper, we introduce SeiFS, a Secure and Lightweight Feature Selection system designed to ensure high-quality inputs for Machine Learning (ML) tasks. Unlike previous approaches involving multiple non-colluding servers, SeiFS operates in a natural ML scenario where multiple entities interact with a single server, without relying on additional strong assumptions. Our work presents intrinsic optimizations in feature selection that yield substantial performance improvements, including a customized data encoding method, a size-optimized comparison circuit, and a shared oblivious dimensionality reduction technique. The customized data encoding method, combined with an optimized secure data access protocol, reduces expensive comparison operations from $O(m)$ to $O(\log m)$, where m represents the number of samples. The size-optimized comparison circuit achieves up to a quadruple reduction in size compared to naïve implementations. Additionally, the shared oblivious dimensionality reduction technique incorporates a novel approximated top-k selection algorithm, resulting in a circuit size reduction of approximately $k\times$. Comprehensive experiments conducted across various network settings demonstrate that our protocols outperform existing solutions, delivering efficiency improvements of an order of magnitude. Specifically, the end-to-end execution of SeiFS on real-life datasets achieves at least 62.7$\times$ improvements in runtime compared to the naïve implementation and takes up to 112.9$\times$ fewer runtimes than the state-of-the-art in the LAN setting.**

***Index Terms*—Feature selection, machine learning, server-aided computation, secure evaluation**

I. Introduction

In recent years, the advancement of Machine Learning (ML) applications and services, such as ChatGPT, has been fueled by the availability of ample data. However, data generated across various domains are increasingly characterized by high dimensionality and often contain irrelevant or obstreperous information, which would compromise their quality and utility. A significant challenge encountered when applying ML algorithms to high-dimensional data is the "curse of dimensionality". This phenomenon occurs when data becomes sparser in high-dimensional space, negatively impacting the performance of algorithms designed for low-dimensional spaces. Additionally, a substantial number of irrelevant features can make training the model computationally intensive and difficult to implement in production. To address these challenges, feature selection (FS) [1], [2] is a common method that can effectively improve the data by identifying an optimal subset of features that characterize the entire feature space. This leads to reduced processing time for training and can even improve the accuracy of the model [3].

Such practices may seem appealing, however, gaining direct access to raw data collected from various data owners is often heavily restricted due to privacy concerns. Consider an example of a credit approval application, feature selection (FS) plays a vital role in training a decision tree model using data from multiple customers within a banking consortium [4]. This decision-relevant data usually includes a large volume of information that are sensitive to privacy, such as monthly income and customer transaction history. The risks of a confidentiality breach have led users to be more reluctant to share their personal data and, in some cases, to not use digital services at all. Regulations like the General Data Protection Regulation (GDPR) also contribute to the limitations on accessing personal data [5]. Additionally, the essence of FS implies that well-preprocessed data features are more likely to possess distinct classification characteristics. However, this also increases the likelihood of potential adversaries intentionally manipulating feature values to mislead the target model, thereby resulting in more severe security threats [6]. As such, ensuring the privacy of raw data and safeguarding well-selected data features from disclosure by adversaries during FS evaluation are of paramount importance.

A. Related Work

To meet this urgent need, several works have explored the feasibility of performing FS under ciphertext using general-purpose encryption techniques. Two notable contributions in this regard are the private FS algorithms proposed by Rao et al. [7] and Ono et al. [8], which rely on homomorphic encryption (HE). However, these methods suffer from significant computational complexity and limited applicability, making them suitable primarily for trivial binary features or small datasets. An alternative approach is secure multiparty computation (MPC), wherein the server performing HE tasks is replaced with a small set of untrusted servers. These servers do not have input to the computation or receive output, rather, their computational resources are available to the parties

Xiaoyuan Liu, Hongwei Li, and Xilin Zhang are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China. (e-mail: xiaoyuanliu@std.uestc.edu.cn; hongweili@uestc.edu.cn; xilinzhang@uestc.edu.cn)

Guowen Xu is with the City University of Hong Kong. (e-mail: guowenxu@cityu.edu.hk)

Tianwei Zhang is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. (e-mail: tianwei.zhang@ntu.edu.sg)

Jianying Zhou is with the Singapore University of Technology and Design, Singapore. (e-mail: jianying_zhou@sutd.edu.sg)

involved. For example, Li et al. [3] proposed a MPC based approach to construct filter based FS, in which discrete or continuous features are ranked based on their predictive ability, measured through metrics such as Gini impurity. By distributing the computational workload across multiple non-colluding servers, this scheme demonstrates exceptional efficiency and scalability. However, it is important to note that the assumption of non-collusion among parties, although prevalent in secure distributed systems, is vulnerable to covert and undetectable collusion.

Overall, prior works have achieved either high security by using HE with prohibitive computational overhead [7], [8] or high efficiency relying on MPC with a strong assumption of multiple non-colluding cloud servers [3]. However, none of these approaches simultaneously achieve both high security and high efficiency.

B. Contributions

To overcome the aforementioned security-efficiency dilemma, we introduce SeiFS, a secure and lightweight feature selection pipeline prior to the commencement of secure horizontal federated learning training [9], [10]. In this paper, we focus on the filter-based FS technique, as it allows us to maintain independence from the training process. This technique is characterized by classifier independence, independent evaluation of each feature, and efficiency and scalability advantages. It comprehensively assesses all features and identifies the optimal subset by analyzing their properties using the Gini impurity. Discussions on other FS methods will be explored in future studies.

Working in a single-server-aided secure computation framework and providing security against static semi-honest adversaries, SeiFS achieves remarkable performance while offering a higher level of security. This is accomplished through a careful co-design of the secure function evaluation framework, garbled circuits, distributed oblivious RAM, and the secure top-k algorithm. We make several optimizations on each of these components to boost online efficiency. Each of these optimizations is of independent interest and can be leveraged for other secure computation tasks. Our specific contributions can be summed up as follows:

- We design a secure and efficient feature scoring process that enables multiple data owners to score all features collaboratively and obliviously with the assistance of a single cloud server. This is achieved through the careful integration of a modified feature encoding method, efficient oblivious data access functionalities, and secure computation techniques. Compared to the state-of-the-art method by Li et al. [3], our protocol demonstrates a remarkable improvement in efficiency, surpassing it by at least an order of magnitude. In particular, our modified feature encoding method plays a crucial role in reducing the number of expensive secure comparison protocols required from $O(m)$ to $O(\log m)$, where m denotes the number of samples. This optimization significantly minimizes the computational burden. Furthermore, we introduce a novel size-optimized circuit representation for comparison operation, resulting in a substantial reduction of the circuit size per comparison by a factor of 4.
- We propose and analyze an approximation algorithm for performing top-k selection in the process of dimensional reduction. Within SeiFS, the secure top-k selection protocol is obtained by garbled circuit. Superlatively, we substitute the comparison component in the top-k circuit with our size-optimized circuit representation, yielding a further speedup in computational complexity. These improvements are likely to be of independent interest for a range of other secure computation tasks.
- SeiFS has been implemented and evaluated in two computation environments that represent real-world LAN and WAN connections. We also compare the performance of SeiFS with the protocols proposed in [11], [3]. The end-to-end evaluation results demonstrate the significant improvements achieved by our protocols. Specifically, in a LAN setting, our protocols reduce the running time by approximately $45\times$ compared to [11]. In a higher-latency WAN setting, the running time is reduced by around $50\times$. Furthermore, when testing our protocols on various real-world datasets, SeiFS consistently outperforms the state-of-the-art approach proposed in [3]. Particularly in network systems with a bandwidth exceeding 400MB/s, SeiFS achieves at least an $18.5\times$ improvement in online running time. The experiments conducted validate the practicality and scalability of SeiFS, particularly for evaluating large volumes of high-dimensional data.

TABLE I
NOTATIONS AND DESCRIPTIONS.

Notations	Descriptions
κ	Cryptographic security parameter
ℓ	Date length in fixed-point representation
n	Number of feature vectors
m	Number of feature samples
N	Number of federated learning participants
$\mathbf{f}_{ij}$	j-th element in the i-th feature vector
θ	Threshold to split the instances into two sets $I_{<\theta}$ and $I_{\geq\theta}$
$p_z(I_j^i)$	Probability that I_j^i is classified as the class L_z ($z \in [1, \phi]$)
$[\mathrm{v}]$	Additive secret sharing in 2PC
$\langle \mathrm{v} \rangle$	Secret-shared value v in form of $\langle\cdot\rangle$-sharing in 2PC

Paper Organization. We begin with a brief review of the problem statement and preliminaries in Section II. Section III introduces the model architecture and threat model used in this paper. We elaborate the details of our design in Section IV. We provide detailed implementation and experimental results in Section V. Section VI concludes the paper and discusses future work.

II. PROBLEM STATEMENT AND PRELIMINARIES

We start by elucidating our problem statement and introducing relevant cryptographic tools, but we refer the interested reader to the bibliography.

A. Gini Impurity-based Feature Selection

One of the crucial decisions in constructing a ML structure is the selection of the attributes that are used for further

model training. Among various FS techniques [12], [13], [14], [15], [16], [17], wrapper-based FS [12] integrates a supervised learning algorithm into the training process and takes into account feature dependencies. Embedded techniques [13], [14] search for an optimal feature subset while constructing the classifier, offering comparable advantages to wrapper techniques but with higher computational complexity. Conversely, filter-based techniques [15], [16], [17] concentrate on evaluating the importance of features based on data characteristics. The classifier-independent property makes it more versatile and flexible, allowing us to provide a secure and efficient pipeline for feature selection that can be integrated with existing federated learning solutions [9], [10].

Filter-based methods, widely applied in machine learning models like decision trees [18] and neural networks [3], entail quantitatively evaluating each feature to identify those with high information gain relative to class distribution. To quantify the impurity of each feature, the Gini impurity measures are often employed [19]. This method calculates the probability that a randomly selected sample would be mislabeled if a label is assigned randomly according to the label distribution. A Gini impurity score of 0 represents the minimum impurity achievable.

Given a data set D consisting of n feature vectors $(\mathbf{f}_1, \cdots, \mathbf{f}_n)$ and one label vector with ϕ different classes, we assume that the i-th feature $\mathbf{f}_i$ is a discrete feature and that all its feature instances can be counted at h different values. Let I_j^i for $j \in [1, h]$ be a set of instances with the value of the j-th feature in $\mathbf{f}_i$. Then, the partition $I_1^i \cup I_2^i \cup \cdots \cup I_h^i$ is formed. Let $p_z(I_j^i)$ implies the probability that the j-th instance set of the i-th feature (i.e., I_j^i) is classified as the z-th class L_z ($z \in [1, \phi]$). Then the Gini impurity of each instance set is calculated as:

$$GI(I_j^i) = \sum_{z=1}^{\phi} p_z(I_j^i) \times (1 - p_z(I_j^i)) \tag{1}$$

where $\sum_{z=1}^{\phi} p_z(I_j^i) = 1$, which is equivalent to $GI(I_j^i) = 1 - \sum_{z=1}^{\phi} p_z^2(I_j^i)$. To determine the quality of $\mathbf{f}_i$ for all $i \in [1, n]$, the weighted Gini impurities are taken as its Gini Score as follows:

$$GS(\mathbf{f}_i) = \frac{1}{m} \sum_{j=1}^{h} |I_j^i| \times GI(I_j^i) \tag{2}$$

where m denotes the number of instances.

However, conventional Gini only works in scenarios where we have categorical features. It is ineffective when dealing with continuous ones. To overcome this issue, we utilize the Mean-Split GINI proposed by Li et al. [3], and modify it to be compatible with MPC. Let $\mathbf{f}_{ij}$ be the j-th element in the i-th feature vector $\mathbf{f}_i$. The core idea behind Mean-Split GINI is to split the instances in the i-th feature into two sets based on their feature value and a threshold θ. The sets are defined as $I_{<\theta}^i = \{\mathbf{f}_{ij} | \mathbf{f}_{ij} < \theta, j \in [1, m]\}$ and $I_{\geq\theta}^i = \{\mathbf{f}_{ij} | \mathbf{f}_{ij} \geq \theta, j \in [1, m]\}$. Now, the Gini impurities of sets $I_{<\theta}^i$ and $I_{\geq\theta}^i$ can be computed as follows:

$$GI(I_{<\theta}^i) = 1 - \sum_{z=1}^{\phi} p_z^2(I_{<\theta}^i); GI(I_{\geq\theta}^i) = 1 - \sum_{z=1}^{\phi} p_z^2(I_{\geq\theta}^i) \tag{3}$$

where $p_z(I_{<\theta}^i) = \frac{|I_{<\theta}^i \cap L_z|}{|I_{<\theta}^i|}$ and $p_z(I_{\geq\theta}^i) = \frac{|I_{\geq\theta}^i \cap L_z|}{|I_{\geq\theta}^i|}$ denote the probabilities that the elements in the set $I_{<\theta}^i$ and $I_{\geq\theta}^i$ are classified as L_z, respectively. The Gini Score of the i-th continuous feature is then calculated as:

$$GS(\mathbf{f}_i) = \frac{1}{m} \cdot (|I_{<\theta}^i| \cdot GI(I_{<\theta}^i) + |I_{\geq\theta}^i| \cdot GI(I_{\geq\theta}^i)) \tag{4}$$

This process makes the continuous features consistent with the original approach (Eq. (1) and (2) by dividing their values into two discrete sets. Finally, based on the Gini Score of each feature, we select the top-k features with the lowest scores.

B. Secure Function Evaluation

Secure Function Evaluation (SFE), a more general notion of MPC, focuses on feasibility results. Essentially, an SFE protocol should ensure that parties learn nothing during the execution beyond the output of the protocol and what is inherently leaked from it. In the context of privacy-preserving FS, SFE proves valuable when mutually distrustful parties need to collaborate under the guidance of a central server (e.g., a service provider) without revealing their inputs to one another. In this scenario, we follow the single-server-aided SFE framework proposed by Kamara et al. [20], [21], where parties $\{1, 2, \cdots, N\}$, each with a private input x_i for $i \in [1, N]$, have only access to a single server S that has no input to the computation but a vast yet bounded amount of computational resources. The goal is to jointly evaluate an m-ary function f on their data [20], [21], [22]. The idea is as follows: rather than requiring all parties to do the same work, let $C \in \{1, 2, \cdots, N\}$ be the party with significantly more resources than others and taking on the work linear in the size of the circuit representation of the function. The remaining players distribute their private data between S and C using secret sharing primitives and run general-purpose MPC protocols for securely evaluating the desired function f on all players' private data.

As such, single-server-aided SFE, along with the underlying secure two-party computation protocols, plays a crucial role in the development of secure and privacy-preserving technologies [23]. It offers a promising approach to designing efficient and practical protocols for secure feature selection. In this work, we specifically investigate the application of this model to develop customized protocols that outperform previous approaches in terms of both security and efficiency for feature selection tasks [3].

C. Secure Two-party Computation

We introduce two types of secret sharing semantics and several cryptographic primitives in secure two-party computation, including Oblivious Transfer [24] and Garbled Circuit [25]. All operations in this paper are performed over the boolean world $\mathbb{Z}_2$ and arithmetic world $\mathbb{Z}_{2^\ell}$. As is well known, the boolean world is efficient for boolean circuits composed of XOR and AND gates, which are commonly used to build non-linear operations. It is rather efficient to evaluate the linear computations over the arithmetic world such as addition and multiplication [26]. To improve efficiency, we mix both types

of field operations and will describe the conversion between them as needed. All the operations over the boolean world are specifically considered to be instances of the arithmetic world when $\ell = 1$. This can be achieved by replacing addition and subtraction with XORs ($\oplus$) and multiplication with ANDs ($\otimes$).

1) Secret Sharing Semantics: In this paper, we use two types of secret sharing semantics, i.e., additive secret sharing and one of its variants.

$[\cdot]$-sharing. This is a typical 2-out-of-2 additive secret sharing, which splits a ℓ-bit secret value v in two ring elements in $\mathbb{Z}_{2^\ell}$ as $[\mathrm{v}]_s$ and $[\mathrm{v}]_c$ such that $[\mathrm{v}]_s + [\mathrm{v}]_c \equiv \mathrm{v} \mod \mathbb{Z}_{2^\ell}$, where $[\mathrm{v}]_i$ for $i \in \{S, C\}$ is sampled independently and uniformly at random from $\mathbb{Z}_{2^\ell}$. Given two $[\cdot]$-shared values x and y, two parties (denoted as S and C) efficiently evaluate addition/subtraction ($[z]_i = [x]_i + [y]_i$ for $i \in \{S, C\}$) and scalar multiplication by a public value a ($[a \cdot x]_i = a \cdot [x]_i$ for $i \in \{S, C\}$) at local with no interaction. For rather complicated multiplication over two $[\cdot]$-shared values x and y, it holds that $z = ([x]_s + [x]_c) \cdot ([y]_s + [y]_c) = [x]_s[y]_s + [x]_s[y]_c + [x]_c[y]_s + [x]_c[y]_c$. For this, party $i \in \{S, C\}$ locally computes $[x]_i[y]_i$, and then two parties use Beaver's triple multiplication technique [27] to generate the additive shares of $[x]_s[y]_c$ and $[x]_c[y]_s$. We also provide the detailed implementation of Beaver's triple multiplication based on the Oblivious Transfer technique and Homomorphic Encryption in the Appendix A.

$\langle\cdot\rangle$-sharing. We call the value $\mathrm{v} \in \mathbb{Z}_{2^\ell}$ to be $\langle \mathrm{v} \rangle$-shared when the party P_i for $i \in \{S, C\}$ holds the values $(\Delta_\mathrm{v}, [\delta_\mathrm{v}]_i) \in \mathbb{Z}_{2^\ell}^2$, and there exist $\delta_\mathrm{v} \equiv_p [\delta_\mathrm{v}]_s + [\delta_\mathrm{v}]_c$ and $\Delta_\mathrm{v} \equiv_p \mathrm{v} + \delta_\mathrm{v}$. To add/subtract two $\langle\cdot\rangle$-shared values x and y or multiply a public value a, party $i \in \{S, C\}$ locally executes $\langle x + y \rangle = \langle x \rangle + \langle y \rangle$ or $\langle a \cdot x \rangle = a \cdot \langle x \rangle$. Besides, to obtain the $\langle\cdot\rangle$-shared product z of two $\langle\cdot\rangle$-shared values x and y, party $i \in \{S, C\}$ locally computes $[\Delta_z]_i$ and sends it to another party:

$$[\Delta_z]_i = i \cdot \Delta_x \Delta_y - [\delta_x]_i \Delta_y - [\delta_y]_i \Delta_x + [[\delta_x][\delta_y]]_i + [\delta_z]_i \quad (5)$$

where $[[\delta_x][\delta_y]]_i$ can be obtained by executing the instance of Beaver's protocol [27] in the data-independent phase. We denote the case that either S or C performs $\Delta_x \Delta_y$ as $i \cdot \Delta_x \Delta_y$.

2) Oblivious Transfer: Oblivious Transfer (OT) [24] is a ubiquitous cryptographic primitive that can be used to develop a variety of cryptographic protocols. In a generalized t-out-of-n OT protocol presented by Ishai et al. [28], the sender inputs n messages, each with ℓ-bit length, while remaining unaware of which, if any, of the messages has been transmitted. On the other hand, the receiver can obtain t of the messages by inputting t choice values, but has no knowledge of the remaining $(n-t)$ messages. Typically, Beaver's triple multiplication over two ℓ-bit private values can be implemented through ℓ instances of the 1-out-of-2 OT protocol. Before evaluating the Garbled Circuit, the evaluator (as the receiver) and the generator (as the sender) perform a 1-out-of-2 OT protocol to obtain the garbled value of the input from the evaluator. The OT protocol proposed by Yang et al. [29] is used as a foundation for designing more efficient non-linear evaluations in secure feature selection.

3) Garbled Circuit: The garbled circuit (GC) technique, introduced by Yao [30], is a popular approach for secure two-party computation. It enables two parties to evaluate arbitrary boolean functions cooperatively without revealing their private inputs. The basic idea is that the circuit constructor (i.e., garbler) generates an encrypted version of a circuit $\widetilde{G}$ that evaluates the function f using the algorithm CreateGC. The garbler assigns garbled values to each wire w_i of the circuit - $\widetilde{w}_i^0$ and $\widetilde{w}_i^1$ - and keeps the values of j secret. For each gate G_i inside the circuit, the garbler creates a garbled table $\widetilde{T}_i$ using symmetric encryption. Each entry in the table is created as follows [31]:

$$\mathrm{Enc}^\kappa_{\widetilde{w}_i^{b_i}, \widetilde{w}_j^{b_j}}(\widetilde{w}_o^{G(b_i,b_j)}) = H(\widetilde{w}_i^{b_i} \| \widetilde{w}_j^{b_j} \| \kappa) \oplus \widetilde{w}_o^{G(b_i,b_j)} \quad (6)$$

With it, the garbled value of G_i's output is only revealed given the garbled values of G_i's inputs. The garbler then sends the garbled inputs $\widetilde{w}_i$ and the garbled circuit $\widetilde{G}$, consisting of these garbled tables $\widetilde{T}$, to the evaluator. The evaluator uses the Oblivious Transfer (OT) technique (described in Section II-C2) to obtain their garbled inputs and can then evaluate the garbled circuit using algorithm EvalGC. On the garbled values corresponding to inputs of both parties, the evaluator evaluates the garbled tables gate by gate, without learning any intermediate values. If desired, the evaluator can return the result to the generator.

Private evaluation with GC requires the function to be decomposed as a circuit of binary gates that process the input bit-wise. Accordingly, evaluating any functions with ℓ-bit input requires the communication complexity of at least $O(\kappa\ell)$, where κ is the cryptographic security parameter. There have been several optimizations to GC, such as free-XOR [32] and garbled-row reduction [33], [34], which can be applied to our implementation.

D. Double-masking Aggregation

Secure aggregation protocol is a specialized multi-party computation protocol that allows a set of clients to compute the summation (a.k.a. aggregation) of their inputs. Let $\mathcal{U} = \{1, \cdots, N\}$ be a set of clients, each holding a private input x_i (e.g., integer, group element, vector). A protocol Π is a secure aggregation protocol if it securely implements the following ideal functionality: $f_{\mathrm{Agg}}(x_1, \cdots, x_N) = (X, \cdots, X)$ for $X = \sum_{i \in \{1, \cdots, N\}} x_i$.

To implement such a functionality, Bonawitz et al. [9] build up a secure and communication-efficient aggregation protocol based on double-masking technique. Specifically, the double-masking of private value x_i includes two parts: the self-mask generated by the owner i and the pairwise-mask generated between i and each other client j ($j \in \{1, \cdots, N\} \backslash i$). The double-masked value x_i is denoted by χ_i:

$$\chi_i = x_i + \underbrace{\mathtt{PRG}(b_i)}_{\text{self-mask}} + \underbrace{\sum_{i<j} \mathtt{PRG}(\mathrm{msk}_{i,j}) - \sum_{i>j} \mathtt{PRG}(\mathrm{msk}_{i,j})}_{\text{pairwise-mask}} \quad (7)$$

where b_i is a secret seed sampled by the client i and $\mathrm{msk}_{i,j} = \mathrm{msk}_{j,i}$ is a pairwise agreed value between clients i and j for each $j \in \{1, \cdots, N\} \backslash i$. In real-world applications, the pairwise can be generated by Diffie-Hellman key agreement [35], [36]. Besides, $\mathtt{PRG}(\cdot)$ represents the pseudorandom generator instantiated with [37].

E. Distributed Oblivious RAM

A similar primitive to the oblivious dimensionality reduction required by the Gini impurity-based FS is referred to as Oblivious RAM (ORAM) [38], [39], [40]. ORAM allows the client to outsource its encrypted storage to an untrusted server while hiding the data access patterns made to the server at an index idx computed from some secret input. Each ORAM scheme supports two basic functions, i.e., Read(idx) and Write(idx, v) for semantic reads and writes to locations specified by a secret-shared index idx.

Our implementation relies on FLORAM [40], a distributed ORAM scheme based on Function Secret Sharing (FSS) [41]. In FLORAM, the private dataset D is $\langle\cdot\rangle^B$-shared between two non-colluded servers S_0 and S_1 for reading, and $[\cdot]^B$-shared between S_0 and S_1 for writing. To perform Read(idx), the implementation utilizes FSS to generate a unit vector $\mathbf{e}$ with all zero elements except $\mathbf{e}[idx] = 1$. Each party holds one piece of additive shares of vector $\mathbf{e}$. Then, party S_i for $i \in \{0, 1\}$ locally executes $[\Delta_{D[idx]}]_i^B = \bigoplus_{j\in[0,n)}[\mathbf{e}[j]]_i^B \cdot \Delta_{D[j]}$, where $D[j]$ represents the j-th item of dataset D. It is easy to verify that $\Delta^B_{D[idx]} = [\Delta_{D[idx]}]_0^B \oplus [\Delta_{D[idx]}]_1^B$. To write to the location idx, parties execute an instance of FSS protocol to construct the additive shares of a vector $\mathbf{e}$ consisting of all zero elements except $\mathbf{e}[idx]^B = \mathrm{v}^B$. Then, parties individually add these shares into the $[\cdot]^B$-shared dataset they hold, i.e., $[D^{'}[j]]^B = [\mathbf{e}[j]]^B \oplus [D[j]]^B$ for $j \in [0, n)$. It follows that $[A^{'}[j]]^B = [D[j]]^B$ at all positions other than the targeted index $idx \in [0, n)$. At index $idx \in [0, n)$, there is $[D^{'}[idx]]^B = [v]^B \oplus [D[idx]]^B$, producing a new value at the target index.

III. System Model

We start this section by presenting our system architecture in Section III-A. We then formally set up the threat model and discuss the privacy requirement in Section III-B.

A. Architecture

We consider a typical horizontal federated learning setting as our system architecture, as described in [42]. This setting involves a central server and a set of N users ($\{1, 2, \cdots, N\}$). Each user owns a privacy-sensitive dataset $\mathcal{D}_i$ consisting of n feature vectors $(\mathbf{f}_1, \cdots, \mathbf{f}_n)$ and corresponding classification label L_z over a given data space. The goal of the users is to collaboratively refine the feature of the private samples they hold, as outlined in Section II-A. The server is responsible for orchestrating the procedure of feature selection. Naturally, communication between users is facilitated through the server, as there are no direct connections between users due to geographical restrictions.

B. Threat Model and Privacy Requirement

Existing secure computation frameworks often rely on a strong assumption that there exist multiple non-colluding cloud servers [43], [44]. Different from them, our system follows the conventional federated learning setting [45], i.e., all secure computations are implemented with the assistance of only one cloud server. Following prior works in the single-server-aided setting [46], [47], we assume a semi-honest adversary $\mathcal{A}$ who can corrupt any subset of clients or the server. This captures the property that the clients and server are always not colluding, i.e., if one party is compromised by the adversary $\mathcal{A}$, the second party behaves honestly. Compromised parties will execute operations as specified by the protocols, but may attempt to deduce additional information. The rationale behind such a threat assumption is that the service providers are typically well-established companies such as Google and Amazon. They have few incentives to jeopardize their reputation, and compliance with the agreement is the basis for long-term cooperation among all participants. Another practical intuition comes from the fear of financial audit and the stringent data protection regulations such as GDPR. How to extend our system to the malicious adversary model [48], [49], [50] is future work.

Formally, we aim to achieve security against a static and semi-honest probabilistic polynomial time adversary following the simulation paradigm [21]. Intuitively, the security guarantees that executing a protocol Π in the real model is equivalent to Π in an ideal model with a trusted third party. Let OUT_i denote the output or view of client i during the execution of Π. We define the execution of Π in real model among parties in the presence of adversaries $\mathcal{A}$ as:

$$\{\mathrm{REAL}^{(i)}(\mathbf{x}; \mathbf{r})\} = \{OUT_j : j \text{ is honest}\} \cup OUT_i$$

where $\mathbf{r}$ represents the randomness. In the ideal execution, OUT_i represents the output returned to honest party i or any value given by corrupted party i. The ideal-model execution of Π between parties in the presence of independent simulators $\mathcal{S}$ is defined as follows:

$$\{\mathrm{IDEAL}^{(i)}(\mathbf{x}; \mathbf{r})\} = \{OUT_j : j \text{ is honest}\} \cup OUT_i$$

Definition III.1 (Security)**.** *For a function $\mathcal{F}$, n-party protocol Π securely computes $\mathcal{F}$ if there exist a set $\{\boldsymbol{Sim}_i\}_{i\in[m+1]}$ of polynomial-size transformations such that for all polynomial-size adversaries $\mathcal{A} = (\mathcal{A}_1, \cdots, \mathcal{A}_{m+1})$, for all input $\boldsymbol{x}$ and random value $\boldsymbol{r}$, and for all $i \in [m + 1]$,*

$$\{\mathit{REAL}^{(i)}(\boldsymbol{x}; \boldsymbol{r})\} =_c \{\mathit{IDEAL}^{(i)}(\boldsymbol{x}; \boldsymbol{r})\}$$

where $=_c$ denotes computational indistinguishability.

We now present a Lemma from [47], [51] that we will utilize to prove security. Since many of our protocols involve multiple sub-protocols, we describe them with the hybrid model. This lemma asserts that to prove the security of a protocol Π in the presence of a set of adversaries $\mathcal{A}$, it suffices to show that the protocol is secure in the semi-honest model.

Lemma III.1. *($\mathcal{F}$-hybrid model) A protocol that invoking a functionality $\mathcal{F}$ is said to be in $\mathcal{F}$-hybrid model if each multi-party sub-protocol Π_i securely computes corresponding functionalities in the presence of a set semi-honest adversaries $\mathcal{A}$. Then Π is also secure in the presence of the adversary $\mathcal{A}$.*

IV. SeiFS Design

In this section, we present a series of secure functions as the main building blocks for secure feature selection.

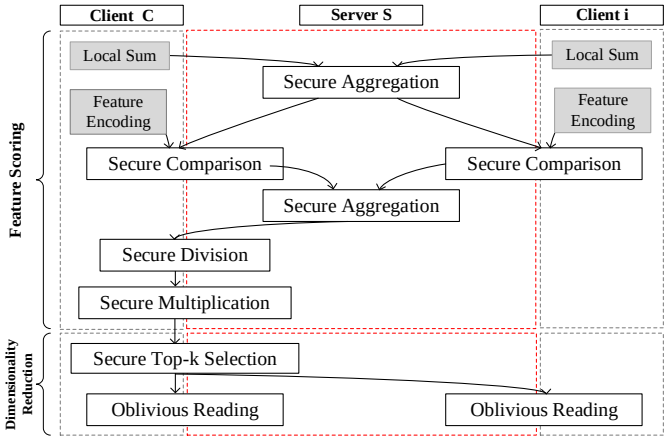


Fig. 1. Technical overview. The server and client C ($C \in \{1, 2, \cdots, N\}$) perform work that is polynomial in the size of the circuit, while the $(N-1)$ remaining clients do sublinear work.

Unlike the previous scheme [3] that outsources all private data to multiple non-colluding cloud servers, our approach leverages a single-server-aid framework [21] and optimizes time-consuming components. We begin by giving a high-level technical overview of our single-server-aid feature selection framework in Section IV-A. Following that, we dive into the details of the core components that we have optimized to improve the efficiency of the selection process. Finally, we present how these components can be integrated into a secure feature selection protocol in Section IV-F.

A. Technical Overview

Following the prior GC-based single-server-aid SFE framework [21], we develop customized constructions with several optimizations for corresponding secure computations over the secret-shared feature values. The core idea is to have the server and one client (assuming $C \in \{1, 2, \cdots, N\}$) perform work that is polynomial in the size of the circuit, while the remaining $(N-1)$ clients do sublinear work. Such a construction would yield a standard secure two-party protocol (2PC) with low communication and computation costs for one party. Further, instead of using the GC to evaluate any general-purpose function as done in [21], we exploit the mixed-protocols [44], [11] to evaluate operations with an efficient representation as an Arithmetic circuit (i.e., additions and multiplications) using secret-shared based protocols, and operations with an efficient representation as a Boolean circuit (e.g., comparisons and divisions) using GC. Several previous works have demonstrated that such a mixed-protocol approach can result in better performance than using GC alone [44], [11], [52].

Recall that the involved parties are N clients $\{1, 2, \cdots, N\}$ and a cloud server S in this paper. Each client i holds a private dataset D_i that includes n feature vectors $\{\mathbf{f}_1, \cdots, \mathbf{f}_n\}$ with length m and a label vector with ϕ different classes. We consider the case where the instances of the same feature are distributed uniformly and randomly among N clients. The clients aim to refine the features in a federated evaluation way without revealing the private dataset D_i and selected features to any clients or the server. We focus on the Gini impurity-based FS method, which is composed of two typical computational blocks: evaluating the Gini scores of features and reducing feature dimensionality to k features with the lowest score. As depicted in Fig. 1, we decompose each block into smaller, composable cryptographic gadgets while -Dmaintaining the security guarantees given in Section III-B. Each unit is tailored to be compatible with our single-server-aid SFE construction, enabling cost-effective and low-bandwidth execution of secure feature selection.

At a high level, the server and clients jointly evaluate the Gini scores for all features as described in Section II-A in a privacy-preserving way. This evaluation incorporates secure processes for aggregation, comparison, division, and multiplication. Along with the above single-server-aid SFE framework, our secure aggregation protocol Π_{Agg} for functionality $\mathcal{F}_{\text{Agg}}(x_1, \cdots, x_N) = \sum_{i=1}^{N} x_i$ starts with agreeing on a common random seed using distributed key generation protocol [53] among the clients who will receive the aggregation result. The participants then leverage the double-masking aggregation protocol (as described in Section II-D) to have the server obtain the sum $\sum_{i=1}^{N}(x_i - \texttt{PRG}(\text{Seed}))$. In the end, the clients have a secret sharing of sum $[\sum_{i=1}^{N} x_i]_i = \texttt{PRG}(\text{Seed})$ while the server holds $[\sum_{i=1}^{N} x_i]_s = \sum_{i=1}^{N} x_i - N \cdot \texttt{PRG}(\text{Seed})$. For ease of exposition, we provide the implementation in Fig. 7 in Appendix B. Besides, the server S and the client C perform the division using a GC as [44] and implement the secure multiplication Π_{Mul} with Beaver's triple multiplication technique [54]. For more challenging comparison, our efforts revolve around two orthogonal directions that are combined to enable better performance in terms of efficiency: 1) minimizing the number of comparison operations through the use of binary tree encoding and oblivious reading and writing protocols, as done in Section IV-B and IV-C; and 2) optimizing the circuit size and latency for lightweight comparison, as shown in Section IV-D.

The second block can be split into the secure top-k selection and the oblivious reading. The former selects the k lowest scores from the n Gini scores and returns their indexes. The latter is used to refine the private dataset consisting of n features such that the features with the selected indexes are retained without revealing to the client the selected items. However, the challenge we encounter is more complex since all inputs and outputs in this setting are secret-shared. To address this challenge, we present our optimization with the circuit size of $O(n + k^2)$ for top-k selection from n values in Section IV-E. Besides, we introduce an optimized reading protocol based on the latest distributed oblivious RAM technology for dimensionality reduction in Section IV-C.

B. Encoding and Evaluating Feature Vectors

As discussed in Section II-A, the Gini impurity based continuous feature selection measures the likelihood of incorrect classification of samples [4] by partitioning m feature samples into two sets $I_{<\theta}$ and $I_{\geq\theta}$ and counting all possible classification probabilities in each set, i.e., $p_z(I_{<\theta}) = \frac{|I_{<\theta} \cap L_z|}{|I_{<\theta}|}$

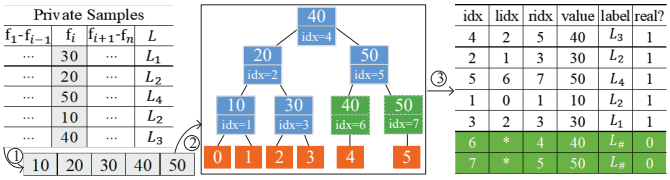


Fig. 2. An example of feature encoding. ① Feature sorting. Each client $i \in [1, N]$ generates the shares of her/his private dataset D_i and sends one piece of the shares (i.e., $\langle D_i \rangle_s$) to the cloud server. ② Encoding sorted feature as a complete binary search tree. For each feature, the client constructs a binary search tree, in which all nodes in the left subtree have feature values that are smaller than the value of its root node, while all nodes in the right subtree have values that are greater than the value of its root node. ③ Encoding tree as array. The array records all information about each sample, including its index, index of the root of the left subtree, index of the root of the right subtree, feature value, and label, where $\star$ represents the value that can be set randomly, and $\#$ denotes the value sampled randomly from $[1, \phi]$, where ϕ is the total number of classes.

and $p_z(I_{\geq\theta}) = \frac{|I_{\geq\theta} \cap L_z|}{|I_{\geq\theta}|}$. Li et al. [3] proposed the Mean-Split Gini scheme, which uses the mean of the feature values as a good approximation for an optimal splitting threshold θ, which reduces the number of expensive comparison operations from $O(m^2)$ to $O(m)$. We further present how to achieve the mean partition using $O(\log m)$ comparison operations. Here, we omit the subscripts that identify the client and feature vectors, as all clients perform the same procedure for each continuous feature. The fundamental intuition is that considering a sorted feature vector (assuming in ascending order), the threshold value θ always appears at a critical position (denoted as $pidx$ in the following). This critical index has the following two properties: (1) for all data whose indexes are less than or equal to $pidx$, their values must be less than or equal to the threshold; (2) for all data whose indexes are greater than $pidx$, the corresponding values must be greater than the threshold. This also implies that $pidx$ equals to the dimensionality of set $I_{<\theta}$ (i.e., $|I_{<\theta}| = pidx$, and $|I_{\geq\theta}| = m - pidx$).

To learn $pidx$ of each continuous feature efficiently, each client encodes locally the features as an array $\mathcal{T}$ as shown in Fig. 2, which records the data structure of a Balanced Binary Tree (BBT) [55] with ordered feature values. The client modifies the standard BBT as shown in ② of Fig. 2. First, the client complements the BBT as a complete binary tree guided by Table II and allocates a virtual index to these supplemented nodes. Taking the example given in Fig. 2, since the node with $idx = 5$ in the $(\lceil \log(m+1) \rceil - 1)$-th layer is located at the right subtree, its missing left child should be set to its parent node (i.e., the node with value 40) with the virtual index of 6 while its missing right child should be itself with the virtual index of 7. The second modification is that we redirect each leaf node of the complete binary tree by setting its right child index to itself while its left child index to its prior index. Note that the supplemented nodes in the left subtree are linked to their parent's index while the supplemented nodes in the right subtree need to be linked to their own actual index. Then, ③ in Fig. 2 presents the procedure of encoding the modified BBT into an array $\mathcal{T}$. A node in the tree is stored in $\mathcal{T}$ in the depth-first search order, where the index of the root is denoted as $tidx$. The row with index idx in $\mathcal{T}$ (also denoted as $\mathcal{T}[idx]$) consists of all information about the node including (1) the index of the node in the sorted features; (2) the index of its left child; (3) the index of its right child; (4) the feature value of the node; (5) the corresponding label; (6) the identification of whether it is a real feature or a supplemented node. Similarly, any feature vector from clients can be naturally represented as an array $\mathcal{T}$ of length $2^{\lceil \log(m+1) \rceil} - 1$. In such a way, locating $pidx$ requires at most $\lceil \log(m+1) \rceil$ instances of secure comparison protocol.

TABLE II
COMPLEMENTING THE BALANCED BINARY TREE.

	Left Child	Right Child
Left Subtree	–	Grandparent Node
Right Subtree	Parent Node	Itself

Benefiting from the structure of this binary tree encoding, we also design an efficient method to count the classification of each set, i.e., $|I_{<\theta} \cap L_z|$ and $|I_{\geq\theta} \cap L_z|$. We define two ϕ-length arrays $\mathbf{q}_{<\theta}$ and $\mathbf{q}_{\geq\theta}$, in which the z-th entry of vector $\mathbf{q}_{<\theta}[z]$ (resp., $\mathbf{q}_{\geq\theta}[z]$) stores the number of samples classified as z-th label L_z in the set $I_{<\theta}$ (resp., $I_{\geq\theta}$). By assuming that the root node of the tree is the threshold, the client sets two initial sets $A_<$ and $A_\geq$. Similar to the definition of the set I, each element in the set $A_<$ (resp $A_\geq$) is less than (resp greater than or equal to) the current root node. Besides, the client initializes two ϕ-length arrays $\mathbf{a}^0_<$ and $\mathbf{a}^0_\geq$ to count the classification in the initial sets $A^0_<$ and $A^0_\geq$. Let $\mathcal{T}^{(0)} = \mathcal{T}$ be the whole evaluation space. Given the threshold θ, the i-th evaluation (computing $b^{(i-1)} = \mathbf{1}\{\mathcal{T}[cidx^{(i-1)}].value < \theta\}$) allows us to relocate the current root node $cidx^{(i)}$ and narrow down the evaluation space to a new tree $\mathcal{T}^{(i)}$ with $cidx^{(i)}$ as the root node. Let the left child and right child of $cidx^{(i)}$ as $l^{(i)}$ and $r^{(i)}$, we compute $cidx^{(i)}$ as below:

$$cidx^{(i)} \leftarrow l^{(i-1)} + b^{(i-1)} \cdot (r^{(i-1)} - l^{(i-1)}) \tag{8}$$

Then, the sets $A_<$ and $A_\geq$ can be repartitioned by the following three guidelines: (1) the parent node of the current root node $cidx^{(i)}$ would be placed to set $A_<$ if $cidx^{(i)} = r^{(i-1)}$; (2) the current root node $cidx^{(i)}$ is kept in the set $A_\geq$; and (3) the nodes in the right subtree of $\mathcal{T}^{(i)}$ should be moved to set $A_\geq$ if $cidx^{(i)} = l^{(i-1)}$, or the nodes in the left subtree of $\mathcal{T}^{(i)}$ are moved to set $A_<$ if $cidx^{(i)} = r^{(i-1)}$. Accordingly, we update $\mathbf{a}^{(i)}_<$ and $\mathbf{a}^{(i)}_\geq$ so that they strictly record the classification of the sets $A^{(i)}_<$ and $A^{(i)}_\geq$, respectively. At the end, we have $\mathbf{q}_< = \mathbf{a}^{(\lceil \log(m+1) \rceil)}_<$ and $\mathbf{q}_\geq = \mathbf{a}^{(\lceil \log(m+1) \rceil)}_\geq$. We formally describe our algorithm for evaluating the BBT in Algorithm 1.

It becomes more challenging when evaluating Algorithm 1 in a secret domain. As the basic requirement is to seal all values from both parties, we use existing protocols [44] for secure additions and multiplications, and design an efficient protocol for secure comparison in Section IV-D. However, it is not enough to hide intermediate values to avoid all leaks. For example, if the secure computation leaks the memory access pattern across each evaluation, the client can obtain $pidx$ directly. What is more, if the client captures the modification

Algorithm 1 $\text{pidx}, \mathbf{q}_{<\theta}, \mathbf{q}_{\geq\theta} \leftarrow \text{BBT}(\mathcal{T}, \theta, \text{tidx}, \mathbf{a}_{<}^{(0)}, \mathbf{a}_{\geq}^{(0)})$

1: $\text{cidx}^{(0)} \| l^{(0)} \| r^{(0)} \| v^{(0)} \| L^{(0)} \| d^{(0)} \leftarrow \text{Read}(\mathcal{T}, \text{tidx})$
2: **for** $0 < i \leq \lceil \log(m+1) \rceil - 1$ **do**
3: $\quad b^{(i-1)} \leftarrow \mathbf{1}\{v^{(i-1)} < \theta\}$
4: $\quad \text{cidx}^{(i)} \leftarrow l^{(i-1)} + b^{(i-1)} \cdot (r^{(i-1)} - l^{(i-1)})$
5: $\quad \text{cidx}^{(i)} \| l^{(i)} \| r^{(i)} \| v^{(i)} \| L^{(i)} \| d^{(i)} \leftarrow \text{Read}(\mathcal{T}, \text{cidx}^{(i)})$
6: $\quad \mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i-1}, L^{(i-1)}, d^{(i-1)} b^{(i-1)} (2b^{(i-1)} - 1))$
7: $\quad \mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i-1}, L^{(i-1)}, d^{(i-1)} b^{(i-1)} (1 - 2b^{(i-1)}))$
8: $\quad \mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i}, L^{(i)}, d^{(i)} (1 - b^{(i-1)}) (2b^{(i-1)} - 1))$
9: $\quad \mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i}, L^{(i)}, d^{(i)} (1 - b^{(i-1)}) (1 - 2b^{(i-1)}))$
10: $\quad \text{nidx}^{(i)} \leftarrow r^{(i)} + b^{(i-1)} \cdot (l^{(i)} - r^{(i)})$
11: $\quad \text{Temp}^{(i)} = \{\text{nidx}^{(i)}\}$
12: $\quad$ **for** $i \leq j < \lceil \log(m+1) \rceil - 2$ **do**
13: $\quad\quad$ **for** $x \in \text{Temp}^{(j)}$ **do**
14: $\quad\quad\quad x \| l \| r \| v \| L \| d \leftarrow \text{Read}(\mathcal{T}, x)$
15: $\quad\quad\quad \mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i}, L, d \cdot (2b^{(i-1)} - 1))$
16: $\quad\quad\quad \mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i}, L, d \cdot (1 - 2b^{(i-1)}))$
17: $\quad\quad\quad \text{Temp}^{(j)} \leftarrow \text{Temp}^{(j)} - \{x\}$
18: $\quad\quad\quad \text{Temp}^{(j+1)} \leftarrow \{l, r\}$
19: $\quad\quad$ **end for**
20: $\quad$ **end for**
21: **end for**
22: $b^{(i)} \leftarrow \mathbf{1}\{v^{(i)} < \theta\}$
23: $\text{pidx} \leftarrow l^{(i)} + b^{(i)} \cdot (r^{(i)} - l^{(i)})$
24: $\mathbf{a}_{<}^{i+1} \leftarrow \text{Write}(\mathbf{a}_{<}^{i}, L^{(i)}, d^{(i)} b^{(i)} (2b^{(i)} - 1))$
25: $\mathbf{a}_{\geq}^{i+1} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i}, L^{(i)}, d^{(i)} b^{(i)} (1 - 2b^{(i)}))$
26: $\mathbf{q}_{<\theta} = \mathbf{a}_{<}^{i+1}; \mathbf{q}_{\geq\theta} = \mathbf{a}_{\geq}^{i+1}$
27: **return** pidx, $\mathbf{q}_{<\theta}$, $\mathbf{q}_{\geq\theta}$

of $\mathbf{a}_{<}$ or $\mathbf{a}_{\geq}$ at the last evaluation, then it would infer the location of $pidx$ with a high probability since the input space is limited in the range of $\{-1, 0, 1\}$. Both of these examples enable the client to learn the evaluation path, which is not allowed according to our security requirements. Our goal here is to obliviously access or read all information of $\mathcal{T}[idx]$ with secret shared index idx and obliviously modify the item in $\mathbf{a}_{<}$ or $\mathbf{a}_{\geq}$. Thus, we formalize the above tasks as oblivious reading and writing functionality. More details can be found in Section IV-C.

With both of these tools, we read the specified item by $\text{Read}(\mathcal{T}, idx)$ at the beginning of each evaluation and then update the array $\mathbf{a}_{<}$ and $\mathbf{a}_{\geq}$ following the first guideline as:

$$\mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i-1}, L^{(i-1)}, d^{(i)} b^{(i-1)} (2b^{(i-1)} - 1)) \quad (9)$$

$$\mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i-1}, L^{(i-1)}, d^{(i)} b^{(i-1)} (1 - 2b^{(i-1)})) \quad (10)$$

where $d^{(i)}$ represents the identification of whether $cidx^{(i)}$ is a real feature. Given the current root node $cidx^{(i)}$ and its classification $L^{(i)}$, we remove its record from $\mathbf{a}_{<}$ and add 1 to the $L^{(i)}$-th item in $\mathbf{a}_{\geq}$ if and only if $cidx^{(i)} = l^{(i-1)}$ as:

$$\mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i}, L^{(i)}, d^{(i)} (1 - b^{(i-1)}) (2b^{(i-1)} - 1)) \quad (11)$$

$$\mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i}, L^{(i)}, d^{(i)} (1 - b^{(i-1)}) (1 - 2b^{(i-1)})) \quad (12)$$

Besides, for all nodes in the subtree, we update $\mathbf{a}_{<}$ and $\mathbf{a}_{\geq}$ as:

$$\mathbf{a}_{<}^{i} \leftarrow \text{Write}(\mathbf{a}_{<}^{i}, L, d \cdot (1 - 2b^{(i-1)})) \quad (13)$$

$$\mathbf{a}_{\geq}^{i} \leftarrow \text{Write}(\mathbf{a}_{\geq}^{i}, L, d \cdot (2b^{(i-1)} - 1)) \quad (14)$$

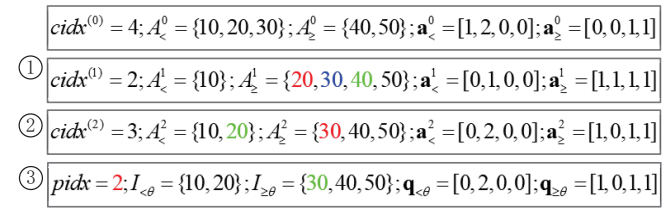


Fig. 3. An example of BBT evaluation. The red values are the current root node, the green nodes represent the parent of the current node and the blue ones indicate the nodes in the subtree that need to be moved.

To facilitate understanding, we give an example in Fig. 3 to show how to obtain $pidx$, $\mathbf{q}_{<\theta}$ and $\mathbf{q}_{\geq\theta}$ intuitively. Given the threshold of the feature samples given in Fig. 2 to be $\theta = 30$, the first evaluation moves the current root node with values of 20 and the node with the value of 30 in the subtree to the set $A_{\geq}$. The second evaluation obtains the current root node $cidx^{(2)} = 3$. Its parent node should be moved from $A_{\geq}$ to $A_{<}$ due to $cidx^{(2)} = r^{(1)}$. Since the current root node is a leaf node, the sets $A_{<}$ and $A_{\geq}$ do not need to be modified. In the end, the node $pidx = 2$ is determined to be set $I_{\geq\theta}$ in the third evaluation.

C. Oblivious Reading and Writing via Improved FLORAM

Line 1 of Algorithm 1 shows that each interaction starts by reading the item with the given index from the array without revealing the accessed item. Similar functionality also works in the oblivious selection of k features as outlined in Section II-E: given the indexes of features with the lowest Gini scores, the functionality refines the raw dataset such that the clean dataset only stores the features with the given indexes. Besides, Algorithm 1 requires a large number of calls to $\text{Write}(\cdot)$ so that secret-shared values are written into the specified items of the array. Inspired by FLORAM [40], we propose new techniques to improve efficiency for oblivious reading and writing.

In FLORAM, given the particular index idx, performing such oblivious reading and writing has to generate tokens $[\mathbf{e}]_s$ and $[\mathbf{e}]_c$ using Function Secret Sharing (FSS) [56] in the online phase, such that $[\mathbf{e}_{idx}]_s \oplus [\mathbf{e}_{idx}]_c = 1$ and $[\mathbf{e}_j]_s \oplus [\mathbf{e}_j]_c = 0$ for all $j \neq idx$. Intuitively, we can improve the online efficiency of our construction if we generate the token in a data-independent phase. To run FSS, the parties iteratively execute the GC protocol $\log n$ times, which requires performing $\log n$ OTs. Since the implementation of concurrent OTs significantly slows down the FSS evaluation, we utilize the silent OT extension technique built upon the vector oblivious linear evaluation [29], which enables us to generate a large amount of random correlated OT correlations (COTs) before the FSS evaluation with only a small amount of additional communication for each GC instance [46].

Assume that FSS generates the unit vector $\mathbf{e}$ related to a random index $ridx$ sampled from the corresponding index space in the offline phase. In the online phase, given the dataset D shared in the form of $\langle \cdot \rangle$, the additive secret-shared index $[idx]$, and a command op, the parties first reveal an offset β that records the difference between the random index and

Protocol 2 $\mathcal{D}_{\text{idx}} \leftarrow \text{Read}(\mathcal{D}, \text{idx})$

Require: A unit vector $\mathbf{e}$ consisting of zero all except $\mathbf{e}[\text{ridx}] = 1$, where ridx is a random index. S holds $\Delta_{\mathbf{e}} = \mathbf{e} + [\delta_{\mathbf{e}}]_s + [\delta_{\mathbf{e}}]_c$ and $[\delta_{\mathbf{e}}]_s$ while C has $\Delta_{\mathbf{e}}$ and $[\delta_{\mathbf{e}}]_c$.

1: Party i for $i \in \{S, C\}$ computes $[\beta]_i = [\text{ridx}]_i - [\text{idx}]_i$ and sends the result to each other;
2: **for** $1 \le j \le |\mathcal{D}|$ **do**
3: Party i for $i \in \{S, C\}$ locally computes $[\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}} \cdot \mathcal{D}_j]_i = i \cdot \Delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} \cdot \Delta_{\mathcal{D}_j} - \Delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} \cdot [\delta_{\mathcal{D}_j}]_i - \Delta_{\mathcal{D}_j} \cdot [\delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}}]_i + [\delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} \cdot \delta_{\mathcal{D}_j}]_i$;
4: Party i for $i \in \{S, C\}$ locally computes $[b]_i \leftarrow [b]_i + [\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}} \cdot \mathcal{D}_j]_i$;
5: **end for**
6: Party $i \in \{S, C\}$ locally sets $[\mathcal{D}_{\text{idx}}]_i \leftarrow [b]_i$;
7: **return** $\mathcal{D}_{\text{idx}}$

special index, i.e., $\beta = ridx - idx$. Then, we correct the unit vector with random index $ridx$ to obtain the unit vector with special index idx by using $\mathbf{e}_{j+\beta \pmod{|D|}}$ for $j \in [1, |D|]$.

For the command $op = \text{Read}$, the core idea of FLORAM is to calculate the dot product of the unit vector and the dataset. Following this, parties have to perform the secure computation to take off the mask of $[\Delta_{D_{idx}}]_i$ to obtain $[D_{idx}]_i$, which inevitably leads to a communication overhead for each invocation of Read(). Motivated by this, we further reduce the overhead of the online phase by introducing a new non-interactive computation. In detail, we require the unit vector $\mathbf{e}$ shared in the form of $\langle \cdot \rangle$. With the observation that $\langle \text{v} \rangle = \langle [\text{v}]_s \rangle + \langle [\text{v}]_c \rangle$, it can be easily obtained by sharing $[\mathbf{e}]_i$ in the form of $\langle \cdot \rangle$ to each other after running the GC-based FSS. Let $\tilde{j} = j + \beta \pmod{|D|}$. Party $i \in \{S, C\}$ performs the local computation as follows:

$$\begin{aligned}[D_{idx}]_i &= [\sum_{j=1}^{|D|} (\mathbf{e}_{j+\beta \pmod{|D|}} \cdot D_j)]_i \\ &= \sum_{j=1}^{|D|} (i \cdot \Delta_{\mathbf{e}_{\tilde{j}}} \Delta_{D_j} - \Delta_{\mathbf{e}_{\tilde{j}}} [\delta_{D_j}]_i - \Delta_{D_j} [\delta_{\mathbf{e}_{\tilde{j}}}]_i + [\delta_{\mathbf{e}_{\tilde{j}}} \delta_{D_j}]_i)\end{aligned} \tag{15}$$

where the last item $[\delta_{\mathbf{e}_{\tilde{j}}} \cdot \delta_{\mathcal{D}_j}]_i$ can be computed via Beaver's multiplication protocol in a data-independent phase as both $\delta_{\mathbf{e}_{\tilde{j}}}$ and $\delta_{\mathcal{D}_j}$ are random values. By doing this, the online phase only requires local computation without interaction between participants, thus resulting in a great gain in efficiency.

Along with the design of FLORAM, the command Write in the Arithmetic world for any $j \in [1, |D|]$ can be executed as:

$$[D_j^{'}]_i \leftarrow [D_j]_i + x \cdot [\mathbf{e}_{j+\beta \pmod{|D|}}]_i \tag{16}$$

However, a slightly different protocol is required to implement the more complex functionality in our scheme where the input x keeps secret-shared. The Beaver's multiplication protocol can be integrated into Eq.(16) directly, but resulting in a communication overhead linear in the size of the dataset. To get over this, party $i \in \{S, C\}$ takes as inputs $\langle x \rangle$ and $\langle \mathbf{e} \rangle$. Similar with Eq.(15), parties obtain the additive share of $x \cdot \mathbf{e}_{\tilde{j}}$ for $j \in [1, |D|]$ by non-interactive local computation as below:

$$[x \cdot \mathbf{e}_{\tilde{j}}]_i = i \cdot \Delta_x \Delta_{\mathbf{e}_{\tilde{j}}} - \Delta_x [\delta_{\mathbf{e}_{\tilde{j}}}]_i - \Delta_{\mathbf{e}_{\tilde{j}}} [\delta_x]_i + [\delta_x \delta_{\mathbf{e}_{\tilde{j}}}]_i \tag{17}$$

where $[\delta_x \delta_{\mathbf{e}_j}]_i$ is generated in the data-independent phase.

Protocol 3 $\mathcal{D}^{'} \leftarrow \text{Write}(\mathcal{D}, \text{idx}, x)$

Require: An unit vector $\mathbf{e}$ which consists of all zero except $\mathbf{e}[\text{ridx}] = 1$, where ridx is a random index. S holds $\Delta_{\mathbf{e}} = \mathbf{e} + [\delta_{\mathbf{e}}]_s + [\delta_{\mathbf{e}}]_c$ and $[\delta_{\mathbf{e}}]_s$ while C has $\Delta_{\mathbf{e}}$ and $[\delta_{\mathbf{e}}]_c$. Besides, party i for $i \in \{S, C\}$ has $[\delta_x \cdot \delta_{\mathbf{e}}]_i$ generated by the Beaver's multiplication protocol.

Ensure: $\mathcal{D}^{'}_{\text{idx}} = \mathcal{D}_{\text{idx}} + x$ and $\mathcal{D}^{'}_j = \mathcal{D}_j$ for all $j \in [1, |\mathcal{D}|] \backslash \{\text{idx}\}$.

1: Party i for $i \in \{S, C\}$ computes $[\beta]_i = [\text{ridx}]_i - [\text{idx}]_i$ and sends the result to each other;
2: **for** $1 \le j \le |\mathcal{D}|$ **do**
3: S locally computes $[\mathcal{D}^{'}_j]_s \leftarrow [\mathcal{D}_j]_s + \Delta_x \cdot \Delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} - \Delta_x \cdot [\delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}}]_s - \Delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} \cdot [\delta_x]_s + [\delta_x \cdot \delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}}]_s$;
4: C locally computes $[\mathcal{D}^{'}_j]_c \leftarrow [\mathcal{D}_j]_c - \Delta_x \cdot [\delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}}]_c - \Delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}} \cdot [\delta_x]_c + [\delta_x \cdot \delta_{\mathbf{e}_{j+\beta \pmod{|\mathcal{D}|}}}]_c$;
5: **end for**
6: **return** $\mathcal{D}^{'}$

Protocols 2 and 3 describe our design details for oblivious reading and writing, which are built upon FLORAM but with a more efficient implementation. To sum up, the introduction of data-independent computation and non-interactive online computation allows a blazing-fast online evaluation.

D. Lightweight Comparison Protocol

As mentioned above, our work is based on the GC. We advance the state-of-the-art secure two-party computation for comparison [11], [44], by developing a more efficient circuit representation. For standard functionalities, we use the size-optimized circuit constructions summarized in [26]. Importantly, our size-optimized comparison circuit can replace any secure comparison circuit, leading to improved efficiency and performance gains. Examples include but are not limited to the secure implementation of activation function ReLU[1] in privacy-preserving machine learning tasks, which has gained substantial attention in current privacy preservation research [46], [47], [57].

To compare two secret-shared values $[a]$ and $[b]$ with ℓ-bit length, the naïve method [11] with a circuit proceeds as shown in Fig. 4(1) involves the following steps: (1) Compute $a = [a]_0 + [a]_1 \mod 2^\ell$ and $b = [b]_0 + [b]_1 \mod 2^\ell$ by obtaining $x = [a]_0 + [a]_1$, $x - 2^\ell$, $y = [b]_0 + [b]_1$ and $y - 2^\ell$ using four ℓ-bit adder/subtractor modules (ADD/SUB). (2) Use a multiplexer (MUX) to check whether x (resp. y) overflows and output x (resp. y) if there is no overflow, otherwise $x - 2^\ell$ (resp. $y - 2^\ell$). (3) Compare a and b using an ℓ-bit comparator (CMP). If the output of the comparison circuit is demanded to be secret-shared, it would be implemented by performing an ADD module with a random value r, which requires an additional ℓ-bit ADD gate. This design results in 16ℓ table entries per comparison [34]. Such a huge overhead motivates us to further reduce the overhead incurred by comparison operations as it has a direct impact on the feature selection process.

The key insight behind our first optimization is that the comparison can be reformulated as $sign(a - b)$ [58], which equals 1 if $a < b$ and 0 otherwise. Fig. 4(2) shows its improvement by only assembling $sign(a - b)$ into the circuit. By

[1] $\text{ReLU}(x) = \max\{0, x\}$.

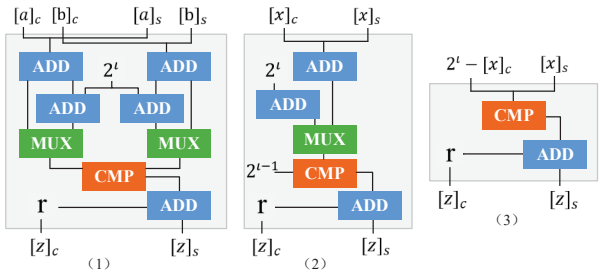


Fig. 4. Circuit construction for comparison. (1) Naïve construction. It consists of five ℓ-bit ADDs, two ℓ-bit MUXs, and one ℓ-bit CMP. (2) Simplified comparison. It consists of three ℓ-bit ADD, one ℓ-bit MUX, and one ℓ-bit CMP. (3) Simplified comparison with the limited regime. It consists of one ℓ-bit CMP and one ℓ-bit ADD.

having two parties pre-compute and provide $[x]_i = [a]_i - [b]_i$ as inputs, the circuit computes $x = [x]_s + [x]_c \mod 2^\ell$ using two ADD/SUB modules and a MUX, and check x against $2^{\ell-1}$ using a CMP. Despite the savings of two ADD/SUB modules and a MUX (i.e., 6ℓ tables entries), it still needs to perform expensive modulo additions to exactly reconstruct x. Our second optimization takes into account the data regime without overflow, significantly simplifying the circuit but introducing an additional fault. As shown in Fig. 4(3), this optimization avoids the reconstruction of x within the circuit by having one party (assumed to be C) input $2^\ell - [x]_c$ instead of $[x]_c$. This change only requires one comparator to check if $[x]_s \leq 2^\ell - [x]_c$, reducing the table entries to 4ℓ. In Appendix C, we give the implementation in Fig. 8 and discuss the practical advantages of our size-optimized circuits in multiple aspects.

Theorem IV.1. *For any $x \in \mathbb{Z}_{2^\ell}$ and v sampled uniformly at random from $\mathbb{Z}_{2^\ell}$, the additive shares of x are $[x]_c = 2^\ell - v$ and $[x]_s = x + v \mod 2^\ell$, respectively. We have:*

$$Pr\{\widetilde{G}_{sign}([x]_s, 2^\ell - [x]_c, r) \neq sign([x]_s, 2^\ell - [x]_c, r)\} = \frac{|x|}{2^\ell}$$

Proof. We defer the proof to Appendix D due to limited space. The error probability $Pr_\epsilon = \frac{|a-b|}{2^\ell}$ has a limited impact since in practice $|x| \ll 2^\ell$ for typical choices of the data field. We also experimentally verify the claim in V.

E. Top-k Selection via Garbled Circuit

The traditional top-k selection functionality $\binom{n}{k}$-MIN in [59] is defined as follows: given a list of n numbers $\{x_1, x_2, \cdots, x_n\}$, the function outputs $k \leq n$ smallest elements in the list . It is also considered that an augmented functionality $\binom{n}{k}$-$\overline{\text{MIN}}$ takes n pairs of value and its index (x_i, idx_i) and returns k smallest elements and their corresponding indexes. In this study, we consider a slight variation of this functionality, referred to as $\binom{n}{k}$-MIN^2, where the output is limited to only the k indexes with the smallest list elements. This can be easily derived from $\binom{n}{k}$-$\overline{\text{MIN}}$ as described in Algorithm 4. The algorithm keeps a sorted list of the current k minimum values and uses a "for" loop to insert each (x_i, idx_i) into its correct location in the sorted arrays.

Algorithm 4 $\{\text{idx}_i\}_k \leftarrow \binom{n}{k}\text{-MIN}^2\ \{(x_i, \text{idx}_i)\}_n,\ k)$

```
OPT = {∅}_k, idxlist = {∅}_k
for 1 ≤ i ≤ n do
    T_opt ← x_i, T_idx ← idx_i
    for 1 ≤ j ≤ k do
        b ← 1{T_opt < OPT[j]}
        x_i = x_i ⊕ b · (OPT[j] ⊕ T_opt)
        OPT[j] = OPT[j] ⊕ b · (OPT[j] ⊕ T_opt)
        idx_i ← idx_i ⊕ b · (idx_i ⊕ idxlist[j])
        idxlist[j] ← idxlist[j] ⊕ b · (idxlist[j] ⊕ T_idx)
    end for
end for
return idxlist
```

Algorithm 5 $\{\text{idx}_i\}_k \leftarrow \text{ATopk}(\{(x_i, \text{idx}_i)\}_n,\ \pi,\ k,\ b)$

```
{(x_π(i), idx_π(i))}_n ← π{(x_i, idx_i)}_n
for 1 ≤ i ≤ b do
    (x'_i, idx'_i) ← (n/b 1)-MIN{(x_π(i·n/b+j), idx_π(i·n/b+j))}_{n/b}
end for
idxlist ← (b k)-MIN^2{(x'_i, idx'_i)}_b
return idxlist
```

Whichever functionality it is, computing top-k naïvely would require a circuit made of $O(nk)$ comparisons and thus results in the circuit size of $O(\ell nk)$, where ℓ is denoted as the data length. Inspired by secure approximate k-nearest neighbors search [59] for $\binom{n}{k}$-MIN, we provide an efficient algorithm for an approximate $\binom{n}{k}$-MIN^2, which enables a relatively smaller circuit size. The intuition of [59] is that the optimal circuit size $O(\ell n)$ of $\binom{n}{k}$-MIN^2 is achieved when k is a constant. Accordingly, if we construct α bins large enough to store the data shuffled uniformly and randomly over the permutation choice π, evaluating the minimum value of each bin (i.e., $(x'_i, \text{idx}'_i) \leftarrow \binom{n/\alpha}{1}$-$\overline{\text{MIN}}\{(x_{\pi(1)}, \text{idx}_{\pi(1)}), \cdots, (x_{\pi(n/\alpha)}, \text{idx}_{\pi(n/\alpha)})\}$) would consume $O(\ell \cdot n/\alpha)$ gates. As such, the circuit size can be reduced from $O(\ell nk)$ to $O(\ell \cdot (n + k\alpha))$. Moreover, $k \cdot \alpha = O(n)$ achieves the optimized circuit size $O(\ell n)$. Along with our lightweight comparison protocol, we expect that $\binom{\alpha}{k}$-MIN$^2\{(x'_{\pi(i)}, \text{idx}'_{\pi(i)}), \cdots, (x'_{\pi(\alpha)}, \text{idx}'_{\pi(\alpha)})\}$ is exactly $\binom{n}{k}$-MIN$^2\{(x_1, \text{idx}_1), \cdots, (x_n, \text{idx}_n)\}$ with the probability of at least $(1-\lambda)(1 - \frac{|\max\{x_i\}|}{2^\ell})^{(n+k\alpha)}$.

We give a concrete illustration for the approximate Top-k selection in Algorithm 5. Notably, our scheme is of independent interest and can effectively utilize its efficiency advantage for Top-k selection based on garbled circuits. This approach can be applied to various applications, such as secure k-nearest neighbor classifiers [59].

Theorem IV.2. *Given a uniformly random permutation π for any n data-index pairs $\{(x_i, idx_i)\}_n$, $\lambda_0 > 0$ and a positive function f_+, for $0 < \lambda < \lambda_0$ and $k = f_+(\lambda)$, the number of bins α can be set to k^r/λ such that the output of $ATopk(\{(x_i, idx_i)\}_n, \pi, k, \alpha)$ with the lightweight comparison protocol is exactly $\binom{n}{k}$-MIN$^2\{(x_1, idx_1), \cdots, (x_n, idx_n)\}$ with the probability of at least $(1-\lambda)(1 - \frac{|\max\{x_i\}|}{2^\ell})^{(n+k\alpha)}$.*

Proof. We defer the proof to Appendix E due to limited space.

F. Putting All Together

By combining the mentioned ingredients, we present our main protocol, which includes a more in-depth explanation of our algorithms for both components, specifically the evaluation of feature scores and dimensionality reduction. We consider a federated setting with N clients and a single cloud server. When securely realizing each component, we maintain the following invariant: participants take the arithmetic shares of the input to each cryptographic gadget and end with the arithmetic shares of the output of the gadget after the protocol. Semi-honest security of the protocol will follow trivially from the composability of individual sub-protocols. Besides, the inputs to secure feature selection are floating-point numbers, encoded as fixed-point integers.

At the beginning of the evaluation of the feature score, the server and N clients perform n instances of the secure aggregation protocol for n features, where client $i \in [1, N]$ takes as inputs $\{(\sum \mathbf{f}_j - \texttt{PRG}(\text{Seed}_j))\}_n$. The protocol enables each client i holds one piece of shares of threshold $[\theta_j]_i = \texttt{PRG}(\text{Seed}_j)$ for the j-th feature. Then the server locally generates another piece of threshold by computing: $[\theta_j]_s = \frac{\sum_i^N (\sum \mathbf{f}_j - \texttt{PRG}(\text{Seed}_j))}{m}$ so that $[\theta_j]_s + [\theta_j]_i = \theta_j$ for any $i \in [1, N]$. Following that, client $i \in [1, N]$ encodes locally each feature into a BBT and records the BBT into an array $\mathcal{T}$. We incorporate both secure reading and writing protocols, arithmetic sharing based secure computation, and garbled circuit based comparison protocol to evaluate our modified BBT evaluation algorithm. Intuitively, all intermediate values are secret-shared between client i and cloud server while the participants keep invisible to traverse the BBT. The evaluation of BBT returns to the server and client i the additive shares of $|I^i_{<\theta}|$, $|I^i_{\geq\theta}|$, $\{|I^i_{<\theta} \cap L_z|\}_\phi$ and $\{|I^i_{\geq\theta} \cap L_z|\}_\phi$. Next, all participants run $2\phi+2$ instances of the secure aggregation protocol to generate the additive shares of $\sum_i^N |I^i_{<\theta}|$, $\sum_i^N |I^i_{\geq\theta}|$, $\{\sum_i^N |I^i_{<\theta} \cap L_z|\}_\phi$ and $\{\sum_i^N |I^i_{\geq\theta} \cap L_z|\}_\phi$ between server S and client C. Note that here client C takes as input the required aggregated value masked with a random value (i.e., $\texttt{PRG}(\text{Seed})$), while the remaining $(N-1)$ clients input their secret values. According to Eq. (3) and (4), client C and server S compute the Gini scores of n features by running $2n$ GC based division protocol as [44] following the two Beaver's triple multiplication protocols.

In the dimensionality reduction process, we implement our approximate top-k selection using a GC to extract k features with the minimum Gini scores. We make further optimization to improve the performance by embedding our optimized GC based comparison module. To achieve this, we develop our GC implementation with the majority of the standard optimizations [60], allowing us to save more than several orders of magnitude in both runtime and memory usage compared to [11]. Finally, once received the k feature indexes, client $i \in [1, N]$ updates the dataset via the secure reading protocol.

Theorem IV.3. *SeiFS provides a secure data pre-processing protocol to realize the ideal functionality of Gini impurity based feature selection with the assistance of a single cloud server in the presence of semi-honest admissible adversaries.*

Proof. We defer the proof to Appendix F due to limited space.

V. Evaluation

A. Experiment Setup

We perform the evaluation with Amazon EC2 instances (c5.4xlarge) under different network settings (LAN and WAN). In the LAN setting, we use multiple instances from the "Asia-Pacific Southeast" region. Here, we set the bandwidth between cloud instances to 3GBps and achieved a round-trip time (RTT) of 0.3ms by configuring Traffic Control in the Linux kernel. Each instance is designated for clients and the cloud server. We simulate the WAN setting with instances hosted in the "Asia-Pacific Southeast" and "US West" regions with a latency of about 20ms and throughput of about 400MBps. We implement our protocols on top of the ABY library [11], but substantially modify the way the secret sharing and the comparison protocol are implemented. Also, we extend the Ferret protocol [29] in the EMP toolkit to support the VOLE-based OT extension [29]. The reported values are the average of ten trials and do not include the one-time setup.

Recall that N is the number of clients, m is the number of records in the private database that each client holds, n stands for the number of continuing attributes, and ϕ denotes the number of classifications. The running time of our algorithm is guaranteed to be independent of the input data values because each component is designed to be oblivious. We test our method using dummy data for various settings of the aforementioned parameters and use 4 threads for all implementations. Since the primary focus of our work is on continuous attributes, we do not include discrete ones in our benchmarks. Besides, the algorithms in this work do introduce errors into the final result, from the lightweight comparison circuit and approximated top-k selection. These errors have been theoretically proven to be controllable below a reasonably small value. To validate this claim, we examine the accuracy of the proposed methods by running our protocols for binary classification tasks over three real-world datasets: Cognitive Load Detection[2] (COG), Lee Silverman Voice Treatment[3] (LSVT), and Speed Dating[4] (SPEED).

B. End-to-End Evaluation

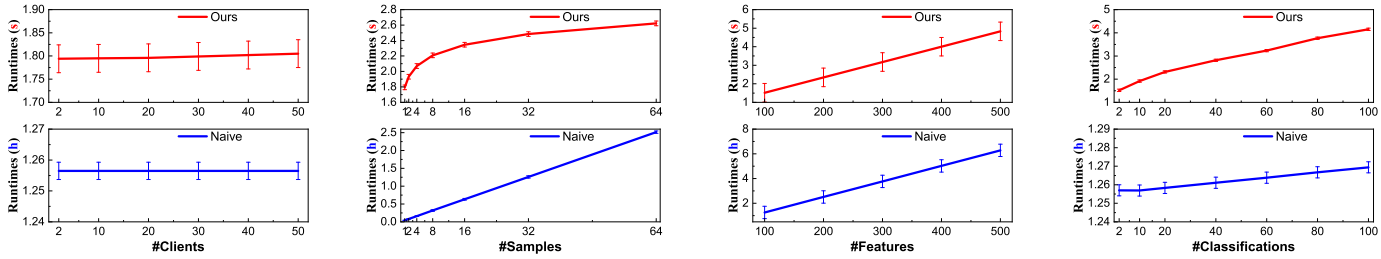
Multi-client. Considering the different numbers of clients ($N = \{2, 10, 20, 30, 40, 50\}$) in our setting, we evaluate the runtime with a fixed $m = 32K$, $n = 100$, $k = 10$ ($\lambda = 0.2$) and $\phi = 2$. Fig. 5(a) and Fig. 6(a) plot the results over different network environments. We observe that in both settings, our algorithm achieves about two orders of magnitude more efficient in terms of runtime than a naïve garbled circuit implementation. SeiFS achieves $2.6 \sim 3.1\times$ runtime improvements, which are greatly attributed to the fact that our feature encoding method reduces the number of secure comparison protocols required from $O(m)$ to $O(\log m)$.

[2]https://www.ubittention.org/2020/data/Cognitive-load%20challenge%20description.pdf

[3]https://archive.ics.uci.edu/ml/datasets/LSVT+Voice+Rehabilitation

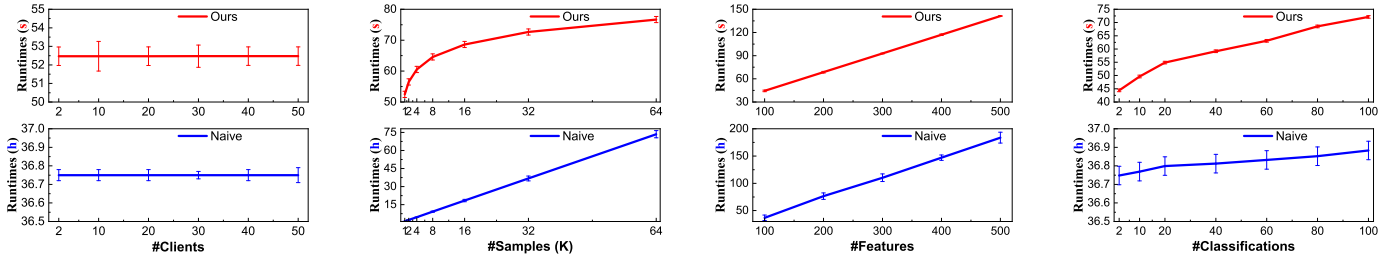
[4]https://www.openml.org/search?type=data&sort=runs&id=40536&status=active

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3422850



(a) Multi-clients (b) Multi-samples (c) Multi-features (d) Multi-classifications

Fig. 5. Comparison of SeiFS and naïve garbled circuit-based approach for secure feature selection tasks in the LAN setting. We document that our protocol completes in a runtime ranging from a few seconds to an hour, depending on the size of the dataset. Compared to the hours the naïve protocol requires, our protocol demonstrates a significant efficiency gain.



(a) Multi-clients (b) Multi-samples (c) Multi-features (d) Multi-classifications

Fig. 6. Comparison of SeiFS and naïve garbled circuit-based approach for secure feature selection tasks in the WAN setting.

Multi-sample. Fig. 5(b) and Fig. 6(b) compare the runtime over different samples in the baseline and our improved algorithms under the LAN and WAN settings. Here, we fix the number of clients to 10, and other variants are the same as in previous experiments. According to the results, the proposed algorithm significantly improves the bottleneck in the naïve garbled circuit implementation for secure feature selection tasks. It is noticeable that the runtime of the proposed method grows slowly with the increase in the number of samples in both figures, which is in close accordance with the curve of the logarithmic function. This comes from two main aspects: (1) comparison operations take up a significant portion of the overhead of the task, and (2) reducing the number of comparisons and the circuit size for implementing secure comparisons as we have done is extremely effective in improving the efficiency of the secure feature selection task.

Multi-feature. In Fig. 5(c) and Fig. 6(c) we summarize how the performance of SeiFS depends on the number of features. We include in the figures the runtime of our complete protocols as well as the naïve garble circuit with the same and fixed $m = 32K$, $N = 10$, $k = 10$ ($\lambda = 0.2$) and $\phi = 2$. With the increasing number of features, SeiFS shows its advantages in runtime, while naïve garbled circuit implementations suffer more involved protocols than SeiFS. Under WAN with 20ms high network latency, the online runtime of SeiFS for high-dimensional datasets is around 60% higher than the naïve garbled circuit. This gap becomes progressively larger as the number of features increases, since our approximate top-k algorithm decouples the multiplicative relationship between n and k, reducing the circuit size for top-k selection implementation from $O(\ell nk)$ to $O(\ell(n + kb))$, where ℓ represents the data length and b is the number of bins required by our top-k.

Multi-classification. We further report the runtime of both SeiFS and the naïve garbled circuit under different classifications. As seen in Fig. 5(d) and Fig. 6(d), SeiFS shows a slightly more distinct change than the naïve garbled circuit implementation. This is mainly because the comparison operations occupy most of the computational resources in the naïve garbled circuit implementation. Other functions used to accommodate changes in the number of classifications (e.g., secure multiplication and division) are far less than the overhead of expensive comparisons, even if they require more computation, The situation is completely different in SeiFS. The increase in the number of other functions required by our scheme is clearly identifiable due to the greatly reduced overhead of the comparison operations.

Multi-thread. We also benchmark the proposed methods in Table III, by running naïve garbled circuit implementations and SeiFS with different threads under the LAN and WAN settings. The performance accounts for one call to the whole protocol with $m = 32K$, $N = 10$, $n = 100$, $k = 10$ ($\lambda = 0.2$) and $\phi = 2$. We only measure the runtime excluding the data-independent phase. We observe that the speed-ups obtained in this way are up to $64.5\times$ for the LAN setting and up to $46\times$ for the WAN setting, even if they are not strictly non-linear with the number of threads. This is mainly due to the memory and network constraints in our approach. Overall, the multi-threaded mode produces a runtime of less than 2.3 seconds in

TABLE III
EVALUATION OF SEIFS IN THE MULTI-THREAD MODE.

# threads	LAN / WAN. Runtime (s)		
	naïve	Ours	Speedup
1	142.48 / 3013.91	2.21 / 65.51	64.5 / 46.0
2	131.58 / 2910.96	2.09 / 62.74	62.9 / 46.4
4	118.01 / 2761.84	1.94 / 60.02	60.8 / 46.0
8	107.36 / 2685.33	1.80 / 57.85	59.6 / 46.4
16	93.32 / 2577.81	1.63 / 53.48	57.3 / 48.2
32	82.19 / 2433.78	1.61 / 51.82	51.1 / 47.1
64	77.05 / 2329.26	1.51 / 50.77	51.0 / 45.9
80	72.01 / 2072.60	1.49 / 47.48	48.3 / 43.7

the LAN setting and 65.51 seconds in the WAN setting.

C. Comparison against Naïve Approaches

In Table IV, we demonstrate that the proposed algorithm works well for real-world feature selection problems. One issue for a fair comparison with the prior work [3] is that it outsources all the calculations for the feature selection task to multiple (three or four in their experiments) non-collusive cloud servers in the LAN. That is why the speedup factor in the WAN setting of [3] is empty. Besides, we re-run the naïve garbled circuit implementation under our environment. From the table, SeiFS is $2.3 \sim 112.9\times$ faster than [3] and $62.7 \sim 169.3\times$ faster than the naïve method in LAN. It is $40.1 \sim 164.2\times$ faster than the naïve implementation in WAN by virtue of slightly less accuracy. Even so, the accuracy does not drop much compared to [3] in which all protocols provide faithful implementations for functionalities.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present SeiFS, an efficient and secure implementation for a practical machine learning data preprocessing pipeline. Unlike prior solutions that rely on multiple non-colluding servers, SeiFS employs a single-server-aided secure function evaluation, improving efficiency, security, and scalability. To achieve these objectives, we incorporate state-of-the-art cryptographic primitives such as secret-sharing, distributed oblivious RAM, and optimized garbled circuits, as well as a newly developed feature encoding method specifically tailored for secure feature selection computations. Performance evaluations show that our solution scales well to massive datasets with up to one million samples. In the future, we plan to explore new methods to speed up the computation of additional feature selection techniques, as well as extend our protocols to protect against malicious deviations.

REFERENCES

[1] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[2] B. Venkatesh and J. Anuradha, "A review of feature selection and its methods," *Cybernetics and information technologies*, vol. 19, no. 1, pp. 3–26, 2019.

[3] X. Li, R. Dowsley, and M. De Cock, "Privacy-preserving feature selection with secure multiparty computation," in *Proceedings of ICML*, 2021, pp. 6326–6336.

[4] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," in *Proceedings of PETs*, 2021, pp. 167–187.

[5] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[6] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning," in *Proceedings of IEEE S&P*, 2022, pp. 1354–1371.

[7] V. Rao, Y. Long, H. Eldardiry, S. Rane, R. Rossi, and F. Torres, "Secure two-party feature selection," *arXiv preprint arXiv:1901.00832*, 2019.

[8] S. Ono, J. Takata, M. Kataoka, T. I, K. Shin, and H. Sakamoto, "Privacy-preserving feature selection with fully homomorphic encryption," *Algorithms*, vol. 15, no. 7, p. 229, 2022.

[9] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of CCS*, 2017, pp. 1175–1191.

[10] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "{BatchCrypt}: Efficient homomorphic encryption for {Cross-Silo} federated learning," in *Proceedings of USENIX ATC*, 2020, pp. 493–506.

[11] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *Proceedings of NDSS*, 2015, pp. 1–15.

[12] W. Zheng, T. Eilam-Stock, T. Wu, A. Spagna, C. Chen, B. Hu, and J. Fan, "Multi-feature based network revealing the structural abnormalities in autism spectrum disorder," *IEEE Transactions on Affective Computing*, vol. 12, no. 3, pp. 732–742, 2019.

[13] C. Khanji, L. Lalonde, C. Bareil, M.-T. Lussier, S. Perreault, and M. E. Schnitzer, "Lasso regression for the prediction of intermediate outcomes related to cardiovascular disease prevention using the transit quality indicators," *Medical Care*, vol. 57, no. 1, pp. 63–72, 2019.

[14] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" in *Proceedings of ICML*, 2015, pp. 1689–1698.

[15] T. M. Khoshgoftaar, A. Fazelpour, H. Wang, and R. Wald, "A survey of stability analysis of feature subset selection techniques," in *Proceedings of IRI*. IEEE, 2013, pp. 424–431.

[16] F. Yang and K. Mao, "Robust feature selection for microarray data based on multicriterion fusion," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 4, pp. 1080–1092, 2010.

[17] M. Bennasar, Y. Hicks, and R. Setchi, "Feature selection using joint mutual information maximisation," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8520–8532, 2015.

[18] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers & security*, vol. 24, no. 4, pp. 295–307, 2005.

[19] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[20] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation," *Cryptology ePrint Archive*, 2011.

[21] S. Kamara, P. Mohassel, and B. Riva, "Salus: a system for server-aided secure function evaluation," in *Proceedings of CCS*, 2012, pp. 797–808.

[22] F. Baldimtsi, D. Papadopoulos, S. Papadopoulos, A. Scafuro, and N. Triandopoulos, "Server-aided secure computation with off-line parties," in *Proceedings of ESORICS*. Springer, 2017, pp. 103–123.

[23] H. Ren, H. Li, D. Liu, G. Xu, N. Cheng, and X. Shen, "Privacy-preserving efficient verifiable deep packet inspection for cloud-assisted middlebox," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1052–1064, 2022.

[24] M. O. Rabin, "How to exchange secrets with oblivious transfer," *Cryptology ePrint Archive*, 2005.

[25] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[26] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Proceedings of CANS*. Springer, 2009, pp. 1–20.

[27] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proceedings of Crypto*. Springer, 1991, pp. 420–432.

[28] Y. Ishai and E. Kushilevitz, "Private simultaneous messages protocols with applications," in *Proceedings of ISTCS*. IEEE, 1997, pp. 174–183.

[29] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast extension for correlated ot with small communication," in *Proceedings of CCS*, 2020, pp. 1607–1626.

[30] A. C.-C. Yao, "How to generate and exchange secrets," in *Proceedings of FOCS*. IEEE, 1986, pp. 162–167.

TABLE IV
PERFORMANCE OF SEIFS ON REAL-WORLD DATASETS.

Dataset	Details					Accuracy			Speed-up	
	m	n	Φ	k	λ	**Baseline**	**naïve**&[3]	**Ours**	**naïve**	[3]
COG	632	120	6	12	0.2	50.9%	52.5%	52.1%	62.7/41.0	18.5/-
LSVT	126	310	10	103	0.3	80.1%	86.2%	85.6%	91.1/40.1	2.3/-
SPEED	8378	122	10	67	Exact	95.2%	97.3%	97.1%	169.3/164.2	112.9/-

[31] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Tasty: tool for automating secure two-party computations," in *Proceedings of CCS*, 2010, pp. 451–462.

[32] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Proceedings of ICALP*. Springer, 2008, pp. 486–498.

[33] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *Proceedings of Eurocrypt*. Springer, 2015, pp. 220–250.

[34] M. Rosulek and L. Roy, "Three halves make a whole? beating the half-gates lower bound for garbled circuits," in *Proceedings of Crypto*. Springer, 2021, pp. 94–124.

[35] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, 2022, pp. 365–390.

[36] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[37] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proceedings of CCS*, 2014, pp. 844–855.

[38] X. S. Wang, Y. Huang, T. H. Chan, A. Shelat, and E. Shi, "Scoram: oblivious ram for secure computation," in *Proceedings of CCS*, 2014, pp. 191–202.

[39] X. Wang, H. Chan, and E. Shi, "Circuit oram: On tightness of the goldreich-ostrovsky lower bound," in *Proceedings of CCS*, 2015, pp. 850–861.

[40] J. Doerner and A. Shelat, "Scaling oram for secure computation," in *Proceedings of CCS*, 2017, pp. 523–535.

[41] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Proceedings of Eurocrypt*. Springer, 2015, pp. 337–367.

[42] X. Zhang, A. Mavromatics, A. Vafeas, R. Nejabati, and D. Simeonidou, "Federated feature selection for horizontal federated learning in iot networks," *IEEE Internet of Things Journal*, 2023.

[43] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Proceedings of IEEE S&P*, 2017, pp. 19–38.

[44] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Aby2. 0: Improved mixed-protocol secure two-party computation," in *Proceedings of USENIX Security*, 2021, pp. 2165–2182.

[45] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[46] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *Proceedings of USENIX Security*, 2022, pp. 1–18.

[47] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of ACM CCS*, 2020, pp. 325–342.

[48] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "Simc: Ml inference secure against malicious clients at semi-honest cost," in *Proceedings of USENIX Security*, 2022, pp. 1361–1378.

[49] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. H. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1364–1381, 2020.

[50] Y. Yang, M. Hu, Y. Cao, J. Xia, Y. Huang, Y. Liu, and M. Chen, "Protect federated learning against backdoor attacks via data-free trigger generation," *arXiv preprint arXiv:2308.11333*, 2023.

[51] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 277–346, 2017.

[52] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of ACM CCS*, 2018, pp. 35–52.

[53] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Proceedings of Eurocrypt*. Springer, 1999, pp. 295–310.

[54] D. Beaver, "Precomputing oblivious transfer," in *Proceedings of CRYPTO*. Springer, 1995, pp. 97–109.

[55] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," in *Proceedings of PETs*, 2019, pp. 266–286.

[56] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *Proceedings of Eurocrypt*. Springer, 2014, pp. 640–658.

[57] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proceedings of USENIX Security*, 2020, pp. 2505–2522.

[58] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic relus for private deep learning," in *Proceedings of NeurIPS*, 2021, pp. 1–12.

[59] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi, "Sanns: Scaling up secure approximate k-nearest neighbors search," in *Proceedings of USENIX Security*, 2020, pp. 2111–2128.

[60] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *proceedings of IEEE S&P*, 2013, pp. 478–492.

[61] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[62] I. Damgard, M. Geisler, and M. Kroigard, "Homomorphic encryption and secure comparison," *International Journal of Applied Cryptography*, vol. 1, no. 1, pp. 22–31, 2008.

[63] Q. Zhang, C. Xin, and H. Wu, "Gala: Greedy computation for linear algebra in privacy-preserved neural networks," in *Proceedings of NDSS*, 2021, pp. 1–16.

[64] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Proceedings of USENIX Security*, 2018, pp. 1651–1669.

[65] P. Yang, Z. L. Jiang, S. Gao, J. Zhuang, H. Wang, J. Fang, S. Yiu, Y. Wu, and X. Wang, "Fssnn: communication-efficient secure neural network training via function secret sharing," *Cryptology ePrint Archive*, 2023.

[66] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of SIGSAC*, 2016, pp. 1292–1303.

[67] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Proceedings of EUROCRYPT*. Springer, 2021, pp. 871–900.

[68] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," in *Proceedings of PETs*, 2020, pp. 291–316.

APPENDIX A
BEAVER'S TECHNIQUE

A. Oblivious Transfer based Beaver Protocol

In oblivious transfer (OT) based Beaver protocol, both parties engage in ℓ OTs on ℓ-bit strings, that is, OT^{ℓ}_{ℓ}. In detail, the party C is assumed as sender with input $(m_{i,0}, m_{i,1})$ to i-th OT, where $m_{i,0} = r_i \xleftarrow{\$} \mathbb{Z}_{2^\ell}$ is sampled randomly and $m_{i,1} = r_i + 2^\ell \cdot [x]_C$. Party S is the receiver taking as input the choice bit $[y_i]_S$, where y_i refers to the i-th bit of $[y]_S$. The execution of OT ensures that the receiver S receives $r_i + 2^\ell \cdot [x]_C \cdot [y_i]_S$. That is followed by the step where party C sets $[[x]_C[y]_S]_C = -\sum_{i=0}^{\ell-1} r_i$ and party S sets $[[x]_C[y]_S]_S = \sum_{i=0}^{\ell-1}(r_i + 2^\ell \cdot [x]_C \cdot [y_i]_S)$.

B. Homomorphic Encryption based Beaver Protocol

To compute the shares of the product $[x]_c[y]_s$, party C encrypts $[x]_c$ and sends it to the server. Homomorphic encryption can be initiated using additive homomorphic encryption, such as Paillier [61] or Damgard-Geisler-Kroigaard (DGK) [62]. Party S then locally performs the multiplication of the ciphertext and the plaintext to get the ciphertext of $[x]_c[y]_s$. After masking this ciphertext with a random value r, party S sends the result back to the client to decrypt. At the end of the protocol, the parties hold the shares of $[x]_c[y]_s$, where the party S holds r as the share while the party C has $[x]_c[y]_s - r$.

APPENDIX B
PROTOCOL OF SECURE AGGREGATION

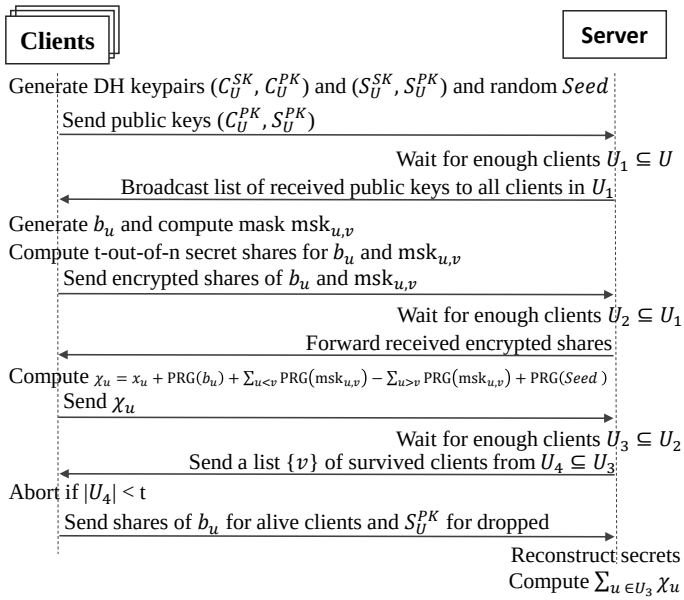


Fig. 7. Secure Aggregation

APPENDIX C
COMPARISON OF SIZE-OPTIMIZED COMPARISON CIRCUIT WITH OTHER WORKS

Here, we elucidate the interaction between client and server required for secure comparison, as depicted in Fig. 8. Subsequently, we extend our analysis to encompass a broader evaluation of schemes developed with sophisticated cryptographic technologies such as secret sharing and Function Secret Sharing.

First, we provide a comparison of communication overhead for secure comparison protocols against several state-of-the-art schemes based on secret sharing. [11], [57], [63], [64] use the naive GC implementation for comparison without optimization. CrypTFlow2 [47] has made considerable headway toward scalable secure neural network inference with secret sharing and OT. Cheetah [46] subsequently optimizes CrypTFlow2 with a more efficient OT technique for comparison operation. ABY2.0 [44] utilizes the offline-online paradigm and optimizes the comparison circuit with a new secret sharing

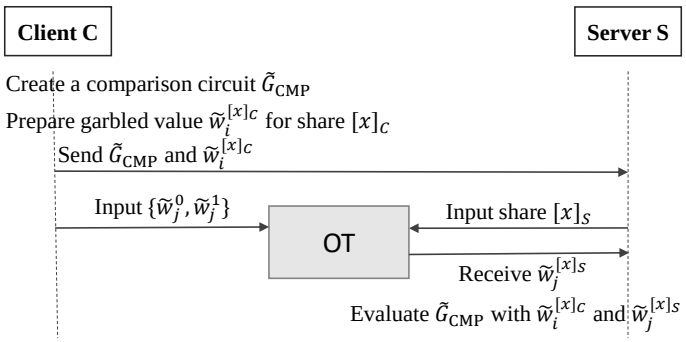


Fig. 8. Secure Comparison

primitives. As shown in Table V, our scheme achieves the same optimal number of communication rounds as GC-based schemes while reducing communication overhead by approximately $4\times$. In contrast to solutions [46], [47] that prioritize reducing communication costs, our scheme requires the fewest number of communication rounds. It is worth noting that federated participants are often geographically distributed, making it difficult to fully utilize the performance of [46] through LAN assistance. In the WAN setting, the latency of a single communication round becomes a bottleneck for running secure multi-party computation. Therefore, using a scheme with the fewest number of rounds offers a significant performance advantage. Especially in such a pay-as-you-go cloud computing scenario, we believe that there is realistic economic benefit in reducing the overall communication overhead and the number of communication rounds.

Then, we compare it with the Function Secret Sharing (FSS) based schemes [65], [66], [67], [68]. Although these methods present benefits regarding online communication costs and the complexity of rounds, they are predicated on the unrealistic premise of a trusted dealer for Distributed Comparison Function (DCF) key generation in the offline phase [68], or their applicability is confined to limited input domains ($\mathbb{Z}_{2^\ell}$ or smaller) [66], [67], thus significantly restricting their practical use. Moreover, the online phase under FSS based schemes is characterized by an increased computational burden for gate evaluation. To illustrate, FSS based models require 16ℓ AES calls for gate evaluation by both entities during the online phase, in stark contrast to our framework, which necessitates only 4ℓ AES calls for evaluating our size-optimized comparison circuit. This quadruple discrepancy in computational demand highlights the superior suitability of our model for federated environments, especially on edge devices with limited resources.

APPENDIX D
PROOF OF THEOREM IV.1

Proof. When $x < 2^{\ell-1}$, the sign is assigned a wrong result if $[x]_s \leq 2^\ell - [x]_c$. This case occurs when $[x]_s = x + v \mod 2^\ell \leq v$, that is, $(x + v)$ causes overflow. Given v sampled uniformly at random from $\mathbb{Z}_{2^\ell}$, the error probability $Pr\{\widetilde{G}_{\text{sign}}([x]_s, 2^\ell - [x]_c, r) \neq sign([x]_s, 2^\ell - [x]_c, r)\}$ is $\frac{x}{2^\ell}$ for $x < 2^{\ell-1}$. Consider another case where x is a negative value, that is, $x \geq 2^{\ell-1}$ in the fixed-point representation. Similarly,

TABLE V
COMPARISON WITH PRIOR WORKS FOR SECURE COMPARISON PROTOCOLS. κ IS THE CRYPTOGRAPHIC SECURITY PARAMETER ASSOCIATED WITH STANDARD CRYPTOGRAPHIC PRIMITIVES, TYPICALLY 128. n REFERS TO THE NUMBER OF FOUR-INPUT AND GATES. FOR ABY2.0 [44], WE HAVE $x_1 = 2n_2 + 8n_3 + 22n_4$ AND $x_2 = n_2 + n_3 + n_4$, WHERE n_2, n_3, n_4 DENOTE THE NUMBER OF AND GATES IN THE BIT EXTRACTION CIRCUIT WITH 2, 3, 4 INPUTS, RESPECTIVELY. FOR EXAMPLE, WHEN $\ell = 64$, THE CIRCUIT NEEDS $n_2 = 41$, $n_3 = 27$ AND $n_4 = 47$.

Protocol	Work	Setup	Online	
		Comm	Comm	Rounds
Secure Comparison over ring $\mathbb{Z}_{2^\ell}$	GC [11], [57], [63], [64]	–	$16\ell\kappa$	2
	CrypTFlow2 [47]	–	$\frac{3}{2}\kappa(\ell+1)+31\ell-13$	$\log\ell+2$
	Cheetah [46]	–	$4(13\kappa+712)+4\ell+2$	$\log\ell+2$
	ABY2.0 [44]	$x_1(\kappa+1)+\ell$	$\ell+2x_2$	$\log_4\ell+1$
	Our work	–	$4\ell\kappa$	2

the wrong sign occurs when $[x]_s = x + v \mod 2^\ell \leq v$. This case captures the error probability $Pr\{\widetilde{G}_{\text{sign}}([x]_s, 2^\ell - [x]_c, r) \neq sign([x]_s, 2^\ell - [x]_c, r)\} = \frac{2^\ell - x}{2^\ell}$ for $x \geq 2^{\ell-1}$.

In summary, for any $x \in \mathbb{Z}_{2^\ell}$, this design incurs the error probability of $\frac{|x|}{2^\ell}$. □

APPENDIX E
PROOF OF THEOREM IV.2

Proof. Given n data point $\{(x_1, \text{idx}_1), \cdots, (x_n, \text{idx}_n)\}$, the first step of Algorithm 5 is to distribute n/α data points over the choice of permutation π into each bin. We first consider the case of the number of bins $\alpha = k/\lambda$. Let A_1 represent the event that each bin contains at least one of the top-k elements. With the assumption that δ is sufficiently small and $k \to +\infty$, we have:

$$\begin{aligned} Pr[A_1] &= 1 - (1 - \frac{1}{\alpha})^k = 1 - e^{k \cdot \ln(1 - \frac{\lambda}{k})} \\ &= 1 - e^{-\lambda + O(1/k)} \geq \lambda - \frac{\lambda^2}{2} + O(1/k) \end{aligned}$$

where the third and fourth equations hold from the Taylor series of $\ln(1+x)$ and e^x, respectively. It is known that the calculation of each bin is independent of each other, the desired expectation of α bins can be computed as follows:

$$\begin{aligned} E &= \alpha \cdot Pr[A_1] = \frac{k}{\lambda} \cdot (\lambda - \frac{\lambda^2}{2} + O(1/k)) \\ &= k \cdot (1 - \frac{\lambda}{2}) + O(1) \end{aligned}$$

To fix the value of λ, the expectation is at least $k \cdot (1 - \lambda)$ due to the assumption of $k \to +\infty$.

To discuss in a more general case where $r > 1$, we denote A_2 as the event that the outputs of $\binom{\frac{n}{\alpha}}{1}$-$\overline{\text{MIN}}\{(x_{\pi(i \cdot \frac{n}{\alpha}+j)}, \text{idx}_{\pi(i \cdot \frac{n}{\alpha}+j)})\}_{\frac{n}{\alpha}}$ (the third step of Algorithm 5) for all $i \in [1, \alpha]$ and $j \in [1, n/\alpha]$ include all elements of $\binom{n}{k} - \text{MIN}^2\{(x_i, \text{idx}_i)\}_n, k)\}$. Now, we can compute the probability of the event A_2 as follows:

$$\begin{aligned} Pr[A_2] &= \underbrace{(1 - \frac{k}{\alpha})(1 - \frac{k-1}{\alpha}) \cdots (1 - \frac{1}{\alpha})}_{k} \\ &= e^{\sum_{c=1}^{k} \ln(1 - \frac{c \cdot \lambda}{k^r})} \\ &= e^{-\frac{\lambda}{k^r} \cdot \sum_{c=1}^{k} (c + O(\frac{1}{k^r}))} \\ &= 1 - \frac{\lambda}{2k^{r-2}} + O(\frac{1}{k^{r-2}}) \end{aligned}$$

where the second step comes from $\alpha = \frac{k^r}{\lambda}$, and the third and fourth steps are extended by the Taylor series of $\ln(1+x)$ and e^x. From this, we can conclude that in the case of $r > 1$, the upper limit of the probability of occurrence of A_2 is $1-\lambda$. With the increase of r, the result of $ATopk(\{(x_i, \text{idx}_i)\}_n, \pi, k, \alpha)$ is exactly $\binom{n}{k}$-MIN$^2\{(x_1, \text{idx}_1), \cdots, (x_n, \text{idx}_n)\}$ with higher probability.

On the other hand, each comparison component is substituted with our lightweight comparison protocol described in Section IV-D, instead of naïve garbled circuit implementation. Following with Theorem IV.1, $ATopk(\{(x_i, \text{idx}_i)\}_n, \pi, k, \alpha)$ consisting of $O(n + k\alpha)$ comparison circuits would introduce the error probability of at least $1-(1-\frac{|\max\{x_i\}|}{2^\ell})^{(n+k\alpha)}$. Thus, the proposed algorithm obtains exact results with probability at least $(1 - \lambda)(1 - \frac{|\max\{x_i\}|}{2^\ell})^{(n+k\alpha)}$.

□

APPENDIX F
PROOF OF THEOREM IV.3

Proof. Review the technical roadmap described in Fig.1, we use secure aggregation protocol proposed by [9] to securely compute the functionality $\mathcal{F}_{\text{Agg}}$, GC [11] and OT [29] for the division functionality $\mathcal{F}_{\text{Div}}$, as well as Beaver's triple technique [54] for the functionality $\mathcal{F}_{\text{Mul}}$. Naturally, security follows these works. For computationally intensive tasks, we modify them for more efficient representations but keep their implementation tools unchanged. For example, we design a new circuit representation for the secure comparison functionality $\mathcal{F}_{\text{CMP}}$, and use the approximated top-k algorithm to reduce the comparison circuit required. Both implementations are based on GC and OT as described in Section II-C3. This implies that the proposed protocols securely compute the functionalities $\mathcal{F}_{\text{CMP}}$ and $\mathcal{F}_{\text{ATopK}}$ in ($\mathcal{F}_{\text{GC}}$, $\mathcal{F}_{\text{OT}}$)-hybrid model against semi-honest adversary. Also, we optimize FLORAM for efficiency gains in the online phase by moving expensive generation work (which was implemented by GC and OT as discussed in Section IV-C) to the offline phase. Thus, our optimized protocols securely achieves the functionalities $\mathcal{F}_{\text{Read}}$ and $\mathcal{F}_{\text{Write}}$ in ($\mathcal{F}_{\text{Beaver}}$, $\mathcal{F}_{\text{GC}}$, $\mathcal{F}_{\text{OT}}$)-hybrid model under the semi-honest threat model.

To sum up, SeiFS securely performs the Gini impurity based feature selection in the ($\mathcal{F}_{\text{Agg}}$, $\mathcal{F}_{\text{GC}}$, $\mathcal{F}_{\text{OT}}$, $\mathcal{F}_{\text{Mul}}$, $\mathcal{F}_{\text{Beaver}}$)-hybrid model under the semi-honest threat model.

□