FedSSA: Reducing Overhead of Additive Cryptographic Methods in Federated Learning with Sketch

Zirui Ou[†], Shujie Han[‡], Qihuan Zeng[§], and Qun Huang[†]
[†]*School of Computer Science, Peking University*
[‡]*School of Computer Science, Northwestern Polytechnical University*
[§]*Peking University*
{zirui.ou, huangqun}@pku.edu.cn, shujiehan@nwpu.edu.cn, 2000013149@stu.pku.edu.cn

***Abstract*—Federated Learning (FL) has been applied across diverse domains as a powerful technique but faces critical challenges in privacy protection. *Secure aggregation* and *additive homomorphic encryption* are two of the most commonly used cryptographic methods to protect model updates. To provide a strong privacy guarantee, both methods satisfy the additivity and require the integer form to encrypt model updates. However, they still suffer from a non-negligible overhead of the computation and communication. To mitigate such overhead, existing approaches of lossy compression (e.g., gradient compression) have been explored but exhibit the inapplicability of FL with additive cryptographic methods. In this paper, we propose FEDSSA, a novel compression framework to reduce the overhead of additive cryptographic methods in FL based on two new techniques: (i) QSRHT Sketch, a sketch-based compression method that supports large compression ratios with a bounded error with the integer requirement, and (ii) periodic rehashing, which ensures the unbiasedness of QSRHT Sketch. Our evaluation shows that FEDSSA achieves a high compression ratio ($\times$160) with a low model accuracy degradation (less than 5%). For additive homomorphic encryption, FEDSSA reduces the average computation time per round by up to 58.15% compared to state-of-the-art compressor that support additive homomorphic encryption, with a low test accuracy drop (within 2.2%).**

I. INTRODUCTION

Federated Learning (FL) is a machine learning paradigm to maintain data privacy via training models on decentralized data sources. It has been widely applied in various domains, including intelligent personal assistants [1], healthcare [2], and financial support [3]. A typical training procedure of FL has two parties [4], i.e., a *logically centralized server* (server for short) and multiple *decentralized clients* (clients for short). The training procedure iterates for multiple rounds. In each round, clients train a shared model using their local data and send only *model updates* (usually gradients) to the server, instead of raw data as in traditional machine learning.

The concern of privacy leakage is crucial in FL. Despite clients keeping their local data away from the server, the server can still infer sensitive information from individual model updates. Specifically, the server can reconstruct the client's sensitive data from individual model updates through data reconstruction attacks or gradient inversion attacks [5]–[8]. This poses a privacy vulnerability in the client's local dataset. To this end, a large body of existing research has tackled this issue with privacy-preserving methods using cryptographic schemes [9]–[12]. These methods are provably secure and can provide a strong privacy guarantee. Among them, *secure aggregation* [10], [13], [14] and *additive homomorphic encryption* [11], [15], [16] are two of the most commonly used methods. Both methods are referred to as the *additive cryptographic methods*, where the additivity means the sum operation can be directly applied to encrypted model updates and yield meaningful aggregated results. Such additive cryptographic methods execute encryption and decryption procedures in integer fields, meaning that the plaintext (i.e. model updates) should be in the integer form [15], [17].

While additive cryptographic methods provide strong privacy guarantees, they still incur significant overhead during computational and communication procedures. Specifically, computational overhead in these methods primarily results from complex modular operations during encryption or decryption processes [16], while the communication overhead is mainly caused by the slow communication connection between clients and the server. Even worse, secure aggregation and additive homomorphic encryption cause the *data inflation* problem, where the ciphertext may be larger than the raw model updates [16], [17]. This further exacerbates the communication overhead and slows down the FL training procedure. Notably, the overhead of these methods are proportional to the size of the plaintext (i.e. the model updates of each client).

To mitigate these overheads, adopting *gradient compression* to reduce the size of model updates seems intuitive. Unfortunately, existing gradient compression methods [18]–[22] exhibit the following limitations which makes them unsuitable for FL with additive cryptographic methods. First, many of them lack additivity, which renders them incompatible with additive cryptographic methods [18]–[20], [23], [24]. Second, certain methods only support a low compression ratio [16], [25]. Therefore, these methods can only offer a limited overhead reduction for additive cryptographic methods. Third, some of the gradient compression methods generate biased model updates, which compromises the model accuracy [26].

In this paper, we propose FEDSSA, which adopts the sketch-based compression method to address the above limitations. Sketch belongs to a family of randomized data structures that compress large datasets using compact arrays and random hash functions. We adopt the *Subsampled Randomized*

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

Hadamard Transform (SRHT) Sketch [27], [28], a well-established technique in traditional machine learning. SRHT Sketch is additive and supports a large compression ratio with bounded error. However, SRHT Sketch faces compatibility challenges with additive cryptographic methods for two main reasons. First, to meet the integer requirement of additive cryptographic methods, SRHT Sketch should convert its fractional elements into integers with bounded error. When compressing model updates, SRHT Sketch takes fractional model updates as input and generates fractional elements. Converting each fractional element into an integer using operations like floor or ceiling functions may seem straightforward. However, these methods may introduce unbounded errors due to the precision loss of directly discarding the fractional parts. Second, it is important to efficiently refresh hash functions to guarantee independence. SRHT Sketch relies on the independence of these hash functions to generate unbiased model updates in each round. While refreshing the hash functions can maintain independence, it is challenging to do so efficiently and correctly. The clients can send requests for the latest hash functions. However, this would cause additional processing overhead on the server side when multiple clients send requests simultaneously. Also, it is crucial to update hash functions correctly on the client side. If the hash function is updated before the client decompresses using the old one, it leads to incorrect model updates and introduces errors.

FEDSSA overcomes the above challenges with two new techniques. For the first challenge, FEDSSA adopts QSRHT Sketch, which enhances SRHT Sketch [27], [28] with a quantization step. In this step, QSRHT Sketch maps each fractional element to an integer with unbiased estimation and a bounded error guarantee. For the second challenge, FEDSSA proposes an efficient protocol called *periodic rehashing* that refreshes the hash functions of QSRHT Sketch every round and maintains the correctness of hash function usage. We conduct a comprehensive evaluation by comparing FEDSSA to 7 state-of-the-art baselines including 2 secure aggregation-based compressors [29], 1 additive homomorphic encryption compressor [16], and 4 gradient compressors without privacy protection [22], [30], [31]. Our evaluation on two tasks (i.e., ResNet9 [32] on CIFAR-10 and ResNet18 [33]on CIFAR-100) shows that FEDSSA achieves a high compression ratio of up to $\times 160$ with no more than 5% model accuracy degradation. When comparing to baseline compressors, FEDSSA achieves comparable test accuracy under the same compression ratio. Moreover, FEDSSA with a larger compression ratio can reduce the average computation time per round by up to 58.15% compared to the additive homomorphic encryption-based compressor, with a test accuracy within 2.2%.

II. BACKGROUND AND MOTIVATION

A. Federated Learning

Federated Learning (FL) is a collaborative machine-learning approach including two parties: a centralized server and multiple decentralized clients [4]. The server coordinates the training of a global model among clients with multiple rounds. In each round, each client trains a local model using its local data and sends *model updates* (i.e., gradients) to the server for aggregation. The server aggregates local models into the global model and sends them back to clients for training in the next round. The process iterates until the desired number of rounds or the target model accuracy is achieved.

Privacy leakage and threat model. Despite clients keeping their local data away from the server, the server can still infer sensitive information from individual model updates [5]–[8]. For example, it can accurately reconstruct a client's training data through a *data reconstruction attack* based on gradient updates. In this paper, we consider a threat model where both the server and clients are *honest-but-curious*, which is standard in the context of FL [5], [10], [16], [34]. In this model, the server faithfully aggregates model updates from clients, but it attempts to infer the private information of each client. Clients execute local training faithfully and do not collude with the server or other clients. Although some existing works consider the potential privacy leakage caused by a client dropout or straggler [34], [35], in our threat model, we do not consider these scenarios as most of these situations can be managed by off-the-shelf privacy protection mechanisms [10]. We leave the exploration of these issues for future work.

B. Secure Aggregation and Additive Homomorphic Encryption

To protect the privacy, *secure aggregation* [10], [13], [14] and *additive homomorphic encryption* [11], [15], [16], [36] are two widely used cryptographic methods in FL to ensure the confidentiality of individual model updates. They require the following properties in use.

- **Additivity.** The property ensures that the additive operation can directly apply to the encrypted model updates for generating aggregated results. Also, decrypting the aggregated results yields the corresponding aggregated model updates. Thus, the plaintext (i.e., model updates) should also be additive. Without additivity, it is impossible to obtain aggregated model updates by decrypting the aggregated results.
- **Integer arithmetic.** Both methods perform encryption and decryption procedures on the integer field. Thus, the model updates should be in integer form.

Secure aggregation. Secure aggregation (SA) is a widely employed privacy protection method in FL [10], [13], [14]. In each round, SA involves clients generating cancellable integer masks that mutually cancel out when summed. After that, clients encrypt their model updates by adding up the corresponding masks to obtain ciphertexts and send the ciphertexts to the server. The server then directly sums the ciphertexts to obtain aggregated model updates without decrypting individual contributions. This works because the integer masks cancel each other out when the ciphertexts are summed, which makes the aggregated sum identical to the sum of the original model updates. This procedure ensures that the server cannot learn individual model updates beyond their aggregated sum. Note that the server's access to the aggregated result does not compromise each client's privacy for two reasons.

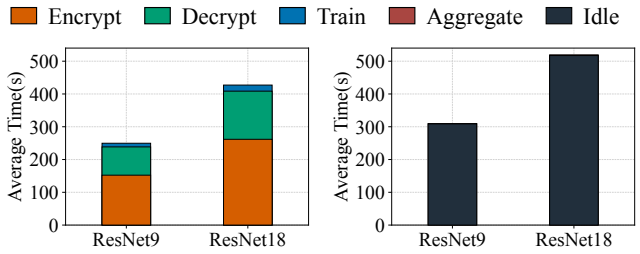


Figure 1: Average time per round breakdowns for client and server.

First, SA is designed so that the server can only access the aggregated model updates instead of the individual updates [10], [14]. This aggregation inherently prevents the server from reconstructing individual contributions. Second, SA can be combined with differential privacy techniques to further enhance privacy protection by adding controlled noise to the results [10], [35], [37], which is complementary to our work.

Additive homomorphic encryption. Additive homomorphic encryption (AHE) is a variant of homomorphic encryption which supports the additive operation on ciphertexts [15], [38]. In each round, clients first agree on a pair of public key and private key. Subsequently, each client encrypts their model updates with the agreed public key and sends the ciphertext to the server. The server then sums the ciphertexts and dispatches the results back to all clients. After that, each client decrypts the results locally with the private key and obtains the corresponding aggregated model updates. Throughout the entire aggregation process, the intermediate results are protected by AHE and cannot be learned by the server.

C. Motivation

The additive cryptographic methods provide strong security guarantees but still incur a significant overhead of computation and communication, which makes them impractical in use.

Overhead of AHE. AHE encounters significant computation and communication overhead. The computation overhead is mainly caused by complex modular operations performed during the encryption and decryption procedures [16], [36], [39], [40]. We evaluate the average time per round of two tasks with AHE on both the client side and server side (see Section V for more details). Figure 1(a) shows that the encryption and the decryption procedure of AHE takes up most of the computation time on the client side, while Figure 1(b) shows that the server spent most of the time (i.e., idle time) waiting for clients to finish computation, which takes up most of the average time per round. The aggregation procedure on the server side is negligible, which is not shown in the figure. Thus, the computation of AHE on clients becomes the bottleneck of the training procedure. Also, the communication overhead of AHE is mainly attributed to the *data inflation* problem [16]. For instance, the Paillier Homomorphic Encryption requires the use of a 2048-bit public key for security, which results in a large increase in message size. When encrypting a gradient of 32 bits with a 2048-bit public key, the message size inflates by a factor of 64. Given that the communication often occurs over slow network connections with limited bandwidth [21], [22], the data inflation problem further exacerbates the communication overhead.

Table I: Requirements of additivity (A), integer (I), large compression ratios (L), and unbiasedness (U).

	Requirements			
	A	**I**	**L**	**U**
SA	✔	✔		
AHE	✔	✔		
sketch-based methods	✔	✘	✔	✘
SRHT Sketch	✔	✘	✔	✔

Overhead of SA. SA also faces non-negligible communication overhead mainly due to data inflation. To avoid overflow in the aggregation procedure, SA reserves additional bandwidth, which results in the data inflation issue similar to AHE [17].

D. Compression of Model Updates

As the computation and communication overhead of additive cryptographic methods is proportional to the size of model updates, reducing it can alleviate such overhead. Several methods such as traditional gradient compression [18]–[22] and sketch-based gradient compression [22], [37], [41], [42] have been applied in distributed machine learning and FL.

Traditional gradient compression. We classify traditional gradient compression methods into two categories, namely *gradient sparsification* and *gradient quantization*. Gradient sparsification approaches [19], [20], [23], [43] reduce the size of model updates by reserving only part of the model updates. And gradient quantization approaches [16], [25], [44] conduct lossy compression by reducing the precision of each element. However, both gradient sparsification and gradient quantization methods have some limitations. First, many existing gradient compression methods lack of additivity, which may lead to erroneous aggregated results when summing the ciphertexts of compressed model updates [19], [20], [23], [25], [43], [45]. Second, both methods struggle for the balance between the reduction of model size and the model accuracy. To reduce the model size, some existing gradient sparsification methods [18], [19], [24], [46] introduce biased model update estimations with biased operators like top-k [24], which leads to a significant model accuracy drop. Also, to maintain the model accuracy, gradient quantization methods [16], [25], [44] only achieve a limited compression ratio of no more than $\times 8$.

Sketch-based methods. Sketch-based gradient compression methods [22], [37], [41], [42] reduces the size of model updates based on *sketch*. Sketch belongs to a family of randomized data structures that compress large datasets using compact arrays and random hash functions [47], [48]. Typically, sketches are associated with three operations, namely *compression*, *decompression*, and *aggregation*. Many sketches are naturally additive and support a large compression ratio with bounded error [27], [47], [48]. Among them, the *Subsampled Randomized Hadamard Transform* (SRHT) Sketch [27], [28] is widely used in traditional machine learning. However, existing sketch-based methods still have several limitations. As shown in Table I, although these methods [22], [37], [41]

inherit the strengths of sketches, they do not satisfy the integer requirement and lack unbiasedness. While SRHT Sketch is attractive because of its unbiasedness, it still falls short of meeting the integer requirement.

III. Overview

A. Design Goals and Challenges

FedSSA is a compression framework of FL that supports additive cryptographic methods with these design goals:

- **Generality (G1).** It should directly support additive cryptographic methods for privacy protection.
- **Efficiency (G2).** It should significantly reduce the overhead.
- **Accuracy (G3).** It should retain a high model accuracy.

Specifically, FedSSA is built on SRHT Sketch (see Section II-D). SRHT Sketch can compress model updates with large compression ratios, therefore reducing the model size and the associated overhead (G2). Also, SRHT Sketch is unbiased, which can retain a high model accuracy (G3). However, SRHT Sketch does not meet the integer requirement. To make it compatible with additive cryptographic methods, the challenge is to maintain high model accuracy in two different aspects.

Challenge 1 (C1): Error-bounded integer conversion. Recall in Section II-B that additive cryptographic methods require the plaintext to be in integer form. However, SRHT Sketch takes fractional model updates as input and generates fractional elements. This discrepancy poses a challenge since fractional elements cannot be encrypted by additive cryptographic methods. While taking operations like floor or ceiling function seems straightforward, these operations are biased and compromise the unbiasedness of SRHT Sketch. Moreover, they may introduce unbounded errors due to precision loss from simply discarding fractional parts. Therefore, it is crucial to convert the sketch elements into integers while maintaining the unbiasedness and ensuring a bounded-error guarantee.

Challenge 2 (C2): Efficient refreshing to generate independent hash functions. Recall in Section II-D that SRHT Sketch compresses model updates with hash functions. The independence of hash functions across rounds is crucial when they are treated as random variables. Specifically, this independence ensures the unbiasedness of SRHT Sketch [27]. Using the same hash functions throughout the training procedure breaks the independence of hash functions and leads to a notable decline in model accuracy. While refreshing the hash functions of SRHT Sketch is an intuitive idea to maintain their independence, it is challenging to do so efficiently and correctly. In particular, the refreshed hash functions should be synchronized across the server and participating clients. Clients can actively send requests to the server for the latest hash functions, but this approach is inefficient and can burden the server with additional processing overhead. Also, it is crucial to update hash functions correctly on the client side. If a client updates the hash functions before decompressing with the old ones, it will result in incorrect model updates and introduce errors into the federated learning procedure.

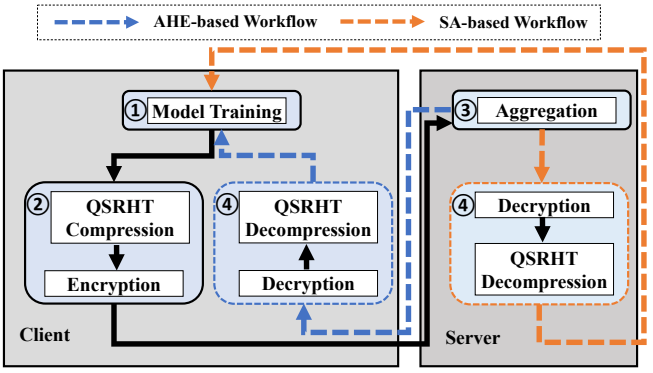


Figure 2: FedSSA workflow.

B. Our Solution

To address the aforementioned challenges, FedSSA is designed based on the following two techniques:

QSRHT Sketch. To address C1, we propose a new sketch called QSRHT Sketch. QSRHT Sketch enhances SRHT Sketch [27] with a *quantization step*. Specifically, it scales the fractional elements to preserve the precision of QSRHT Sketch with a parameter called the *quantization scaling factor*. To meet the theoretical guarantee of unbiased estimation, it adopts *probabilistic quantization* to map each fractional element into an integer. The overall introduced error of QSRHT Sketch can be bounded with a theoretical guarantee. Also, we prove that QSRHT Sketch maintains the additivity in expectation, even though the probabilistic quantization is a non-linear operation.

Periodic rehashing. To tackle C2, we present a new protocol called the *periodic rehashing*, which refreshes the hash functions in QSRHT Sketch every round to maintain the independence of hash functions. Specifically, the server sends the latest hash functions *along* with the latest model parameters back to clients. This can avoid the requests from clients for new hash functions. After receiving the latest hash functions, the client delays the hash function update procedure until it correctly decompresses with old hash functions, which ensures the correctness of hash function usage.

C. Architecture

FedSSA includes two parties, namely the *server* and multiple *clients*. FedSSA supports two workflows, namely *SA-based workflow* and *AHE-based workflow*. SA-based workflow supports secure aggregation while AHE-based workflow supports additive homomorphic encryption.

FedSSA workflow. As shown in Figure 2, the workflow of FedSSA consists of four steps:

- **Step (1): Model training.** At the beginning of each round, the server randomly selects a subset of available clients to participate. These clients then acquire the global model and train the model for multiple iterations using the local dataset. In the SA-based workflow, clients fetch the global model from the server, whereas in the AHE-based workflow, clients utilize the locally stored global model.
- **Step (2): Client processing.** The client first updates the hash functions of QSRHT Sketch. Then, the client compresses

Notation	Meaning
Defined in Section IV-A	
S	counter array in QSRHT Sketch
m	number of columns in S
D	sign hash function $\{0,1,...,d-1\} \to \{-1,1\}$
R	index function $\{0,1,...,m-1\} \to \{0,1,...,d-1\}$
$\mathbf{g}$	original model update vector before compression
$\mathbf{g}'$	model update vector after decompression
d	size of the model update vector $\mathbf{g}$
i	the i-th element in a vector of size d $(0 \le i \le d-1)$
j	the j-th element in a vector of size m $(0 \le j \le m-1)$
$\mathbf{H}$	normalized Walsh-Hadamard matrix
α	quantization scaling factor
Q	quantization function
r	compression ratio $r = d/m$
K	number of selected clients
Defined in Section IV-B	
t	t-th round in communication
D_t	sign hash function for round t
R_t	index function for round t
Defined in Section IV-C	
$\mathcal{C}_\alpha(\mathbf{g})$	the estimation of $\mathbf{g}$ obtained by QSRHT Sketch
Defined in Section IV-D	
$\mathbf{w}$	model parameters
E	number of iterations for local training
$F(\mathbf{w})$	loss function with $\mathbf{w}$
$\mathbf{g}_{t,v}^u$	the v-th gradient computed by client u in round t

Table II: Major notation used in Section IV.

local model updates with QSRHT Sketch and encrypts QSRHT Sketch using the corresponding cryptographic method.

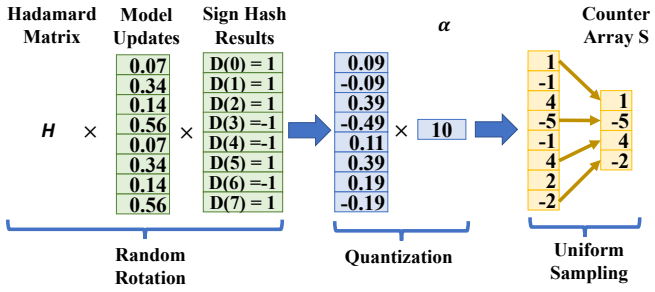
- **Step (3): Server aggregation.** The server aggregates the encrypted QSRHT Sketches from participating clients by summing them together to obtain the aggregated results.
- **Step (4): Post-processing.** In the SA-based workflow, the post-processing step occurs on the server side, while in the AHE-based workflow, each client executes this step individually. In this step, the corresponding party first decrypts the results obtained in Step (3) to generate the aggregated QSRHT Sketch. Then the party decompresses the aggregated QSRHT Sketch to obtain an unbiased estimation of aggregated model updates. The estimated model updates are then used to refine the global model.

IV. Design

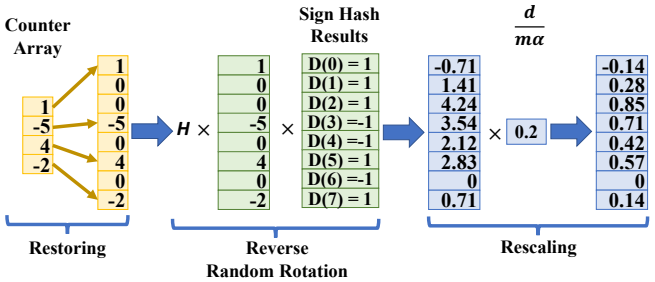
A. QSRHT Sketch

We first introduce the data structure of QSRHT Sketch. Then we introduce the operations of QSRHT Sketch, namely *compression*, *decompression* and *aggregation*. We summarize the major notations in Table II used in Section IV.

Data structure. A QSRHT Sketch comprises a counter array (denoted by S) with m columns and two random hash functions, including the sign hash function (denoted by D) and the index hash function (denoted by R). Let $\mathbf{g}$ be a one-dimensional vector of model updates with d elements, the *compression ratio* of QSRHT Sketch is defined by $r = d/m$. QSRHT Sketch compresses the model updates $\mathbf{g}$ into S via D and R. The function D maps the sign of each element in $\mathbf{g}$ to ± 1 uniformly at random based on the index of the element, i.e., $\{0,1,...,d-1\} \to \{-1,1\}$. The function R uniformly samples m elements from $\mathbf{g}$ with replacements, i.e., $\{0,1,...,m-1\} \to \{0,1,...,d-1\}$.



(a) Compression of QSRHT Sketch.



(b) Decompression of QSRHT Sketch.

Figure 3: The compression and decompression of QSRHT Sketch.

Compression. Figure 3(a) shows the compression procedure of QSRHT Sketch, which compresses $\mathbf{g}$ into S by three steps.

- **Random rotation.** QSRHT Sketch first randomly flips the sign of each element in $\mathbf{g}$ and gets an intermediate vector $\mathbf{g}_r = (D(0) \times g_0, ...D(i) \times g_i, ..., D(d-1) \times g_{d-1})$, where g_i denotes the i-th element in $\mathbf{g}$. It then multiplies $\mathbf{g}_r$ by the normalized Walsh-Hadamard matrix [27] $\mathbf{H}$ and obtains the rotated vector $\mathbf{h} = \mathbf{H}\mathbf{g}_r$. The matrix $\mathbf{H}$ is defined as $\mathbf{H} = \frac{1}{\sqrt{d}}\mathbf{H}_d$, where $\mathbf{H}_d$ is computed recursively via $\mathbf{H}_d = \begin{pmatrix} \mathbf{H}_{\frac{d}{2}} & \mathbf{H}_{\frac{d}{2}} \\ \mathbf{H}_{\frac{d}{2}} & -\mathbf{H}_{\frac{d}{2}} \end{pmatrix}$ from $\mathbf{H}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.
- **Quantization.** To preserve the precision of $\mathbf{h}$, QSRHT Sketch first scales $\mathbf{h}$ by multiplying the quantization scaling factor (defined in Section III-B) (denoted by α) and obtains a scaled vector $\alpha\mathbf{h}$. QSRHT Sketch then applies the probabilistic quantization on the scaled vector $\alpha\mathbf{h}$ element-wisely to obtain the quantized result $\mathbf{h}_q$. Specifically, let Q be the quantization function and h_i be the i-th element in $\mathbf{h}$. The i-th element in $\mathbf{h}_q$ is computed by $Q(\alpha h_i)$, where

$$Q(x) = \begin{cases} \lfloor x \rfloor + 1 & \text{with the probability } p(x) = x - \lfloor x \rfloor, \\ \lfloor x \rfloor & \text{with the probability } 1 - p(x), \end{cases}$$

  and $\lfloor x \rfloor$ denotes the integer no more than an element x. Thus, each element in $\mathbf{h}_q$ is an integer and $\mathbf{h}_q$ is an unbiased estimation of the vector $\alpha\mathbf{h}$.
- **Uniform sampling.** To reduce the size of model updates (i.e., d), QSRHT Sketch performs uniform sampling with the index hash function R to choose m element from $\mathbf{h}_q$ and stores the sampled elements into S. Specifically, for the j-th element in S, $R(j)$ outputs the index of the selected element in $\mathbf{h}_q$. Note that all elements in S are now integers.

Decompression. The decompression procedure is to restore the original model update vector $\mathbf{g}$ from the counter array S. Figure 3(b) shows the corresponding decompression procedure of QSRHT Sketch, which consists of the following three steps:

- **Restoring.** QSRHT Sketch first restores the counter array of m columns back to an intermediate vector (denoted by $\mathbf{h}'$) with d elements. The i-th element in $\mathbf{h}'$ is computed by $h'_i = \sum_{R(j)=i} S[j]$.
- **Reverse random rotation.** This step can be viewed as the reverse operation of the random rotation step. In this step, QSRHT Sketch first multiplies the corresponding sign hash function to obtain the intermediate result $\mathbf{g}'_r = (D(0) \times h'_0, ...D(i) \times h'_i, ...D(d) \times h'_d)$. Then it multiplies the matrix $\mathbf{H}$ with $\mathbf{g}'_r$ to obtain $\mathbf{g}_h = \mathbf{H}\mathbf{g}'_r$.
- **Rescaling.** Finally, QSRHT Sketch rescales $\mathbf{g}_h$ to obtain the unbiased estimation of the original vector $\mathbf{g}' = \frac{d}{m\alpha}\mathbf{g}_h$, which is the final result of the decompression operation.

Aggregation. The aggregation operation of QSRHT Sketch is used by the server to aggregate QSRHT Sketches from clients. When two QSRHT Sketches have the same parameters (i.e. d, m, and α), they can be aggregated by summing the corresponding values in the counter arrays (see Theorem 2 in Section IV-C).

Selection of quantization scaling factor α. The quantization scaling factor α determines the precision preserved in model updates. A larger α preserves more precision for the floating-point values. However, an excessively large α may cause numerical overflow in the scaled vector ($\alpha\mathbf{h}$) and introduce errors. Denote the number of participating clients by K. To determine a proper α, we select α such that $\alpha > \sqrt{\frac{dr}{K}}$ at the beginning of training and use the same values during training. We provide the analysis of α selection in Section IV-D.

B. Periodic Rehashing

Protocol. The periodic rehashing is designed to refresh the sign hash functions and the index hash function per round with low computation overhead. In each round, the server and clients execute the following steps.

- **Server generation.** When sending the latest global model (or aggregated QSRHT Sketch) to clients, the server independently and uniformly at random samples the hash functions D and R. For each message sent to participating clients that contains the global model (or aggregated QSRHT Sketch), the server piggybacks D and R.
- **Client saving.** When each client receives the latest global model (or aggregated QSRHT Sketch) from the server, it also receives the latest hash functions D and R. Instead of directly updating, the client saves D and R.
- **Client updating.** When each client begins its model training, it updates QSRHT Sketch with D and R.

Benefits. The server generation can efficiently synchronize hash functions among clients. The client saving and the client updating can make periodic rehashing compatible with two different workflows and ensure the correctness of hash functions usage (see Section III-C). In the SA-based workflow, the client saving is followed by the client updating, which

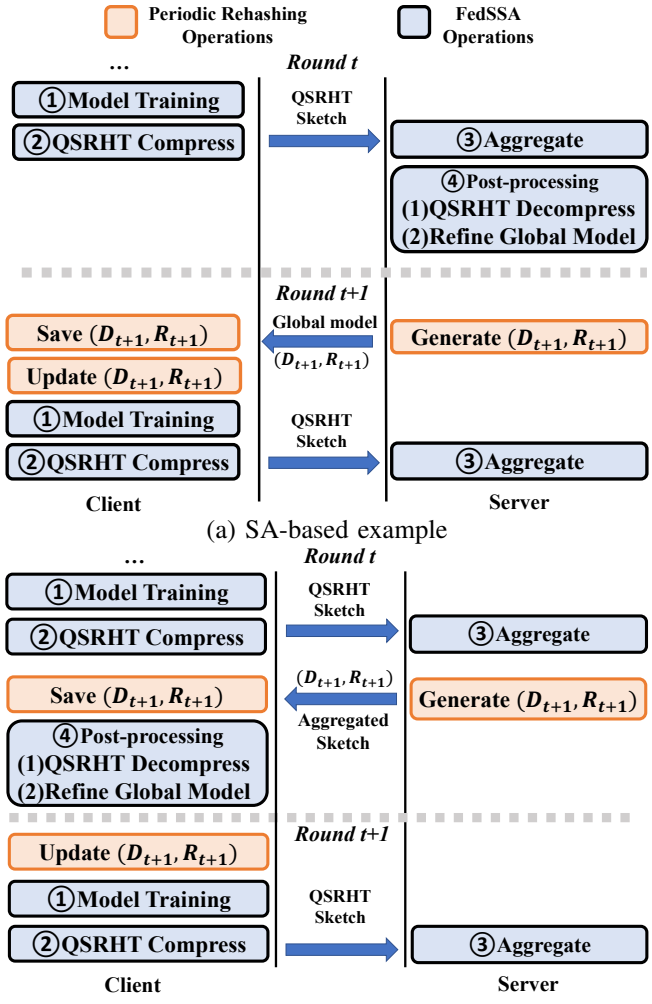


Figure 4: Workflow of periodic rehashing. Encryption and decryption are omitted for simplicity.

is equivalent to an immediate update. However, in the AHE-based workflow, the client should decompress the aggregated QSRHT Sketch with old hash functions. If the client updates hash functions immediately, the corresponding decompression operation would use the new hash functions and result in wrong model updates.

Workflows. Let D_t and R_t denote the sign hash function and index hash function at round t. We show the SA-based and AHE-based workflow of periodic rehashing.

- **SA-based workflow.** Figure 4(a) illustrates the SA-based workflow. In round t, the client trains the model, compresses the updates, and sends them to the server. The server aggregates the QSRHT Sketch and decompresses the updates to refine the global model. Note that in round t, both sides use hash functions (D_t, R_t) for compression and decompression. In round $t+1$, the server sends the updated global model and new hash functions (D_{t+1}, R_{t+1}) to the client. Then the client executes the client saving and the client updating to update new hash functions.
- **AHE-based workflow.** Figure 4(b) shows the AHE-based workflow. In round t, the client sends its QSRHT Sketch to

the server. The server aggregates and sends the aggregated results along with new hash functions (D_{t+1}, R_{t+1}) to each client. The client saves the new hash functions, and decompresses the updates using the old hash functions to refine the global model. At the start of round $t+1$, the client updates its hash functions before local training.

Discussion. While periodic rehashing samples new hash functions on the server side, it does not compromise clients' privacy for two reasons. First, in an honest-but-curious setting, the server is assumed to generate the hash functions correctly rather than deliberately creating incorrect ones to manipulate the system maliciously. Second, FEDSSA's privacy protection does not rely solely on the randomness of the hash functions. Instead, it mainly employs additive cryptographic methods that provide strong privacy guarantees. This ensures that even with periodic rehashing, the clients' privacy remains intact.

C. Analysis of QSRHT Sketch

We provide the theoretical guarantees of QSRHT Sketch, including unbiasedness, additivity and bounded variance property. For a model update vector $\mathbf{g} \in \mathbb{R}^d$ and a given scaling factor α, we denote the estimation of $\mathbf{g}$ as $\mathcal{C}_\alpha(\mathbf{g})$.

Theoretical guarantees. Theorem 1 provides the unbiasedness of QSRHT Sketch with periodic rehashing. Theorem 2 provides the additivity guarantee of QSRHT Sketch, which enables the aggregation operation of QSRHT Sketch. Theorem 3 offers a guarantee on the variance of model update estimations for QSRHT Sketch. In the interest of space, we put the proofs of Theorem 1–Theorem 3 into our technical report [49].

Theorem 1. *Given $\alpha \in \mathbb{R}$ and $\mathbf{g} \in \mathbb{R}^d$, QSRHT Sketch is an unbiased estimator of $\mathbf{g}$, i.e. $\mathbb{E}[\mathcal{C}_\alpha(\mathbf{g})] = \mathbf{g}$. The expectation is over the sign hash function and the index hash function.*

Theorem 2. *Given $\alpha \in \mathbb{R}$, for any $\mathbf{g_1}, \mathbf{g_2} \in \mathbb{R}^d$, QSRHT Sketch satisfies $\mathbb{E}[\mathcal{C}_\alpha(\mathbf{g_1} + \mathbf{g_2})] = \mathbb{E}[\mathcal{C}_\alpha(\mathbf{g_1}) + \mathcal{C}_\alpha(\mathbf{g_2})]$.*

Theorem 3. *Given quantization scaling factor $\alpha \in \mathbb{R}$, $\forall \mathbf{g} \in \mathbb{R}^d$, we have $\mathbb{E}[\|\mathcal{C}_\alpha(\mathbf{g}) - \mathbf{g}\|_2^2] = \frac{(d+m-1)d}{4m\alpha^2} + \frac{d-1}{m}\|\mathbf{g}\|_2^2$.*

The necessity of periodic rehashing. Theorem 1 relies on the randomness of hash functions to guarantee the unbiasedness of QSRHT Sketch. However, if the same hash functions are used throughout the training procedure, the unbiasedness of QSRHT Sketch no longer holds.

Variance analysis of QSRHT Sketch. Theorem 3 establishes a theoretical bound for the variance of the model update estimation. The term $\frac{d-1}{m}||g||_2^2$ can be viewed as the *sampling variance*. The coefficient of the sampling variance $\frac{d-1}{m} \approx \frac{d}{m}$ represents the compression ratio of QSRHT Sketch, which captures the trade-off between compression ratio and variance.

D. Convergence Analysis of FEDSSA

Notations. We first introduce some notations. The FL optimization problem involves N clients to learn the model parameter (denoted by $\mathbf{w}$) collaboratively. The federated optimization problem is defined as $\min_w F(w) = \frac{1}{N}\sum_{i=1}^{N} F_i(\mathbf{w})$, where $F_i(\mathbf{w})$ denotes the local loss functions of the i-th client. In each round, FEDSSA server randomly selects K clients without replacement and each client performs E iterations for local training. In each round t, we denote the current model parameter by $\mathbf{w}_t$, the learning rate by η_t, and the j-th iteration gradient of the client i by $\mathbf{g}_{t,v}^u$.

Assumptions. We make the following assumptions which are standard in FL convergence analysis [50]. Specifically, Assumption 1 assumes Lipschitz continuous gradients on loss functions. Assumption 2 and 3 assume the stochastic gradient of each client to be unbiased and bounded.

Assumption 1. *There is a constant $L > 0$, such that $\|\nabla F_i(\mathbf{u}) - \nabla F_i(\mathbf{v})\| \leq L\|\mathbf{u} - \mathbf{v}\|$, $\forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, $\forall 1 \leq i \leq N$.*

Assumption 2. $\mathbb{E}\left[\mathbf{g}_{t,v}^u\right] = \nabla F_i(\mathbf{w}_t)$, $\forall 1 \leq i \leq N$.

Assumption 3. *There exist two constants $\sigma_L > 0, \sigma_G > 0$, such that $\mathbb{E}\left[\|\mathbf{g}_{t,v}^u - \nabla F_i(\mathbf{w}_t)\|^2\right] \leq \sigma_L^2$, $\forall 1 \leq i \leq N$. and $\|\nabla F_i(\mathbf{w}_t) - \nabla F(\mathbf{w}_t)\|^2 \leq \sigma_G^2$, $\forall 1 \leq i \leq N$, $\forall 1 \leq t \leq T$.*

Convergence analysis. When assuming that all clients participate in each round (i.e. $N = K$), the following theorem provides a convergence bound for FEDSSA. We leave the proof of the Theorem 4 to our technical report [49].

Theorem 4. *Suppose η is a constant and $\eta \leq \min\left(\frac{1}{8EL}, \frac{1}{EL\left(1+\frac{d-1}{m}\right)}\right)$. When using a constant learning rate $\eta_t = \eta$ for each round, the following bound holds for* FEDSSA*:*

$$\min_{1\leq t\leq T}\{\mathbb{E}\|\nabla F(\mathbf{w}_t)\|^2\} \leq \frac{F(\mathbf{w}_0) - F(\mathbf{w}_*)}{c\eta ET} + M, \quad (1)$$

where $F(\mathbf{w}_*) := \min_{\mathbf{w}}\{F(\mathbf{w})\}$,

$$M := \frac{1}{c}\left[\frac{L\eta}{E}c_2 + \frac{L\eta}{2K}\left(1 + \frac{d-1}{m}\right)\sigma_L^2 + \frac{5E\eta^2L^2}{2}c_3\right],$$

where c is a constant, $c_2 := \frac{d(1+r)}{4K\alpha^2}$ and $c_3 := \sigma_L^2 + 6E\sigma_G^2$.

Quantization scale selection. The term $\frac{d(1+r)}{4K\alpha^2}$ in equation (1) can be interpreted as the quantization error of QSRHT Sketch. Theorem 4 suggests that by carefully selecting the quantization scaling factor, the quantization step in QSRHT Sketch has a minimal impact on the final model accuracy. In the probabilistic quantization step, QSRHT Sketch chooses a quantization scale factor such that $\alpha \geq \left(\sqrt{\frac{d\times r}{K}}\right)$.

V. EVALUATION

A. Experiment Settings

Implementation details. To implement the FEDSSA prototype, we first develop a single-machine simulator in Python (∼300 LoC) to simulate the training procedure. Then we develop the FEDSSA server module (∼1,600 LoC) and the FEDSSA client module (∼160 LoC) on top of the simulator with Python. We also develop QSRHT Sketch with PyTorch on GPU (∼180 LoC). For AHE, we adopt the Paillier's scheme [15]. We use Intel's Paillier Cryptosystem Library [51] to implement the encryption and decryption procedure.

Datasets, models, and data allocation. We conduct image classification tasks (CV tasks) on CIFAR-10 and CIFAR-100, and a next-character prediction task (NLP task) on the Shakespeare dataset [4], [52]. The models and data allocation details are summarized in Table III.

Table III: Datasets, Models, and Data Allocation Summary.

Dataset	Model	Parameters	Data Allocation
CIFAR-10	ResNet9 [32]	6.5M	Dirichlet
CIFAR-100	ResNet18 [33]	11M	Dirichlet
Shakespeare	LSTM [53]	4M	real-world non-i.i.d.

For both CV tasks, we simulate non-independent and identically distributed (non-i.i.d.) data distributions among N clients using the Dirichlet distribution $Dir(\beta = 0.1, 0.3, 0.5)$, where smaller β values represent higher non-i.i.d. levels. For the NLP task, we use a subset that contains 143 clients as suggested in [52]. In both ResNet models, batch norm [54] is replaced with group norm [55] as suggested in [56].

Metrics. We use the following two metrics for evaluation, namely the *test accuracy* and *average computation time*. The test accuracy is the percentage of correctly classified samples in the test dataset. The average computation time represents the average time spent on computation by the client throughout the procedure. We use the average computation time to evaluate the computation overhead when AHE is in use [16] (Exp#7).

FL settings. We evaluate FEDSSA under two federated learning settings, namely *SA setting* and *AHE setting*. We summarize the hyperparameters in Table IV.

Table IV: Hyperparameter Summary for FL Settings.

Settings	SA setting	AHE setting
Rounds (T)	1000	100
Clients (N)	100 (143 for Shakespeare)	10
Clients per round (K)	12 (10 for Shakespeare)	10
Learning rate (η)	0.1	0.01

For the SA setting, we adopt a cosine annealing learning rate scheduler to stabilize the training procedure. For the AHE setting, we set the key size of Paillier encryption to 2048 bits. We use the following hyperparameters for both settings. Each client performs $E = 3$ epochs of local updates with a batch size of $B = 64$ using the SGD optimizer, which is consistent with [4]. For FEDSSA, we use $\alpha = 10^6$ by default.

Baseline algorithms. We evaluate the test accuracy of FEDSSA with 7 baseline algorithms from three categories:

- **SA-based compressor.** We compare FEDSSA with 2 SA-based compression methods, namely KVSAgg-Minmax (KVS-MM) and KVSAgg-GSpar (KVS-GS) [29]. As they support a large compression ratio, we vary the compression ratio r from 20 to 160 with a step size of 20, and we refer to them as *large compression ratios*. To ensure correct decompression under compression ratio r, we configure KVS-MM and KVS-GS with specific parameters $W = \frac{d}{11.25 \times K \times r}$ and $m = \frac{d}{9 \times r}$ as suggested in [29].
- **AHE-based compressor.** We compare FEDSSA with 1 AHE-based algorithm, namely BatchCrypt (BC) [16]. BC is a quantization-based method and only supports a small compression ratio (i.e., 2 and 4). We set $r = 2, 4$ for their comparison and refer to them as *small compression ratios*, while we also validate the test accuracy of FEDSSA under large compression ratios.
- **Non privacy-preserving compressor.** To show that FEDSSA can retain test accuracy under large compression ratios, we further compare FEDSSA with 4 baseline gradient compression methods without privacy protection in the SA setting. We choose 3 sparsification methods, namely Minmax sampling (MM) [30], GSpar sampling (GS) [31] and random-k sampling (RK) [57] with shared randomness (i.e., choosing the same elements among clients). We also compare FEDSSA against FetchSGD (FSGD) [22], which is the state-of-the-art sketch-based method in FL.

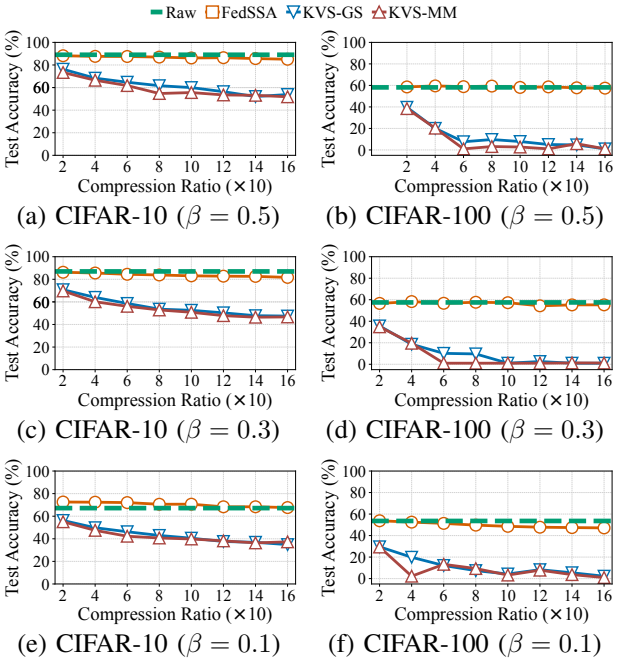


Figure 5: (Exp#1) Accuracy comparison in the SA setting.

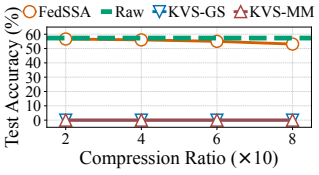


Figure 6: (Exp#1) Accuracy comparison on Shakespeare.

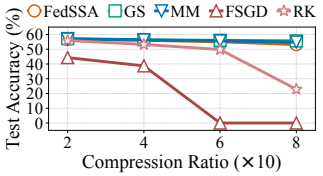


Figure 7: (Exp#3) Accuracy comparision on Shakespeare.

B. Experiment Results

(Exp#1) Accuracy comparison in the SA setting. We compare the test accuracy of FEDSSA with KVS-MM and KVS-GS under the SA setting on 7 tasks: CIFAR-10 ($\beta = 0.5, 0.3, 0.1$), CIFAR-100 ($\beta = 0.5, 0.3, 0.1$), and Shakespeare. We also evaluate tasks without compression (Raw).

Figures 5 and 6 show that FEDSSA outperforms KVS-MM and KVS-GS across all tasks, with improvements up to 34.25% for the CV tasks and up to 56% for the NLP task. Notably, KVS-MM and KVS-GS fail to converge at $r = 20$ in the NLP task, whereas FEDSSA maintains stability across varying compression ratios. Moreover, FEDSSA achieves test accuracies close to those of the corresponding tasks without compression. For instance, as shown in Figures 5(a), 5(c), and 5(e), FEDSSA achieves 88.24%, 86.24%, and 72.56% accuracy on CIFAR-10 tasks at $r = 20$, compared to 89.05%, 86.97%, and 67.17% without compression. Even as

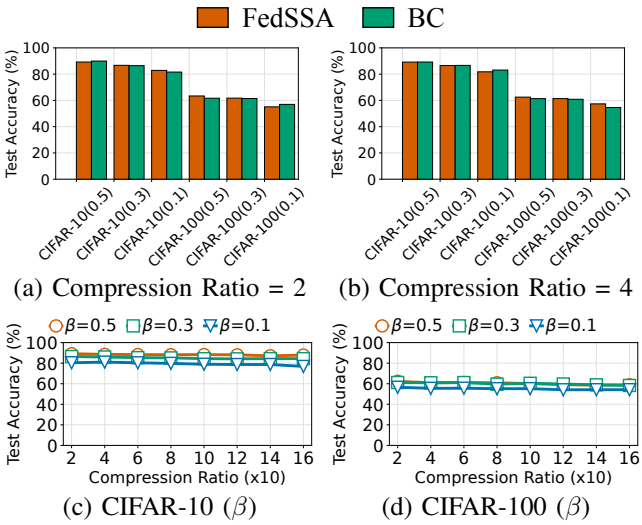


Figure 8: (Exp#2) Accuracy comparison in the AHE setting.

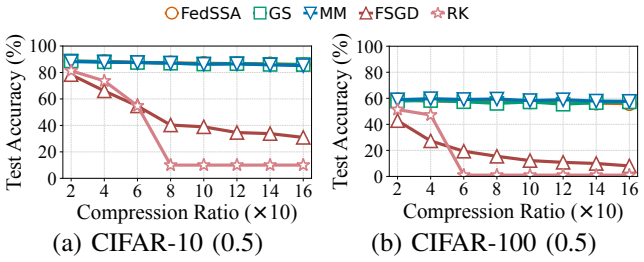


Figure 9: (Exp#3) Accuracy comparison with non-privacy compressors.

the compression ratio increases to $r = 160$, FEDSSA only experiences minor accuracy reductions of 3.10%, 4.50%, and 4.91%, respectively. In contrast, KVS-MM and KVS-GS suffer significant degradations of over 17.76% when $r = 160$.

(Exp#2) Accuracy comparison in the AHE setting. We first compare the test accuracy of FEDSSA with BC under the AHE setting on CIFAR-10 ($\beta = 0.5, 0.3, 0.1$) and CIFAR-100 ($\beta = 0.5, 0.3, 0.1$). Figures 8(a) and 8(b) show the accuracy for compression ratios $r = 2$ and $r = 4$, respectively. The results indicate that FEDSSA achieves a comparable test accuracy to BC under the same compression ratio. Specifically, the differences between FEDSSA and BC are within 1.86% and 1.33% across different datasets for $r = 2, 4$, respectively.

We next validate the test accuracy of FEDSSA for large compression ratios under the AHE setting. Figures 8(c) and 8(d) show that when r varies from 20 to 160, FEDSSA achieves a stable model accuracy with no more than 4.19% and 3.8% of accuracy drop on CIFAR-10 and CIFAR-100, respectively. Also, we compare the test accuracy of FEDSSA with $r = 20$ to that of BC with $r = 4$. For CIFAR-10, the test accuracy of FEDSSA is lower than that of BC within 2.2%, while for CIFAR-100, the test accuracy of FEDSSA is higher than that of BC by up to 2.76%.

(Exp#3) Accuracy comparison with non-privacy compressors. We evaluate the test accuracy of FEDSSA and 4 baseline compressors without privacy protection. We evaluate Shakespeare, CIFAR-10 (0.5), and CIFAR-100 (0.5).

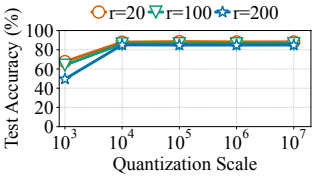


Figure 10: (Exp#4) Impact of α.

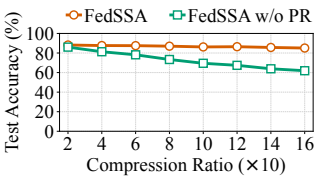


Figure 11: (Exp#5) Effect of periodic rehashing.

Figure 7 and Figure 9 show the results on the NLP task and the CV tasks, respectively. It shows that FEDSSA achieves comparable test accuracy compared to baselines on these training tasks. For the Shakespeare task, as shown in Figure 7, MM, GS, and FEDSSA maintain stable test accuracies ranging from 57.23% to 53.12%. In contrast, the test accuracy of FSGD and RK quickly drops to 0% (due to Nan) and 22.89% when $r = 80$. For the CIFAR-10 ($\beta = 0.5$) task, MM, GS, FSGD, RK, and FEDSSA achieve test accuracies of 88.7%, 88.53%, 78.35%, 81.31%, and 88.24%, respectively for $r = 20$. As the compression ratio increases, the test accuracy decreases by 2.6%, 2.7%, 47.42%, and 3.10% for MM, GS, FSGD, and FEDSSA, respectively. Meanwhile, the test accuracy of RK rapidly decreases to 1% when $r = 60$. For the CIFAR-100 ($\beta = 0.5$) task, MM, GS, and FEDSSA maintain stable test accuracies ranging from 59.69% to 55.86%. In contrast, the test accuracy of FSGD and RK achieves only 42.82% and 51.33% when $r = 20$ and quickly drops to 12.15% and 1% when $r = 120$. Compared to FS, FEDSSA retains higher model accuracy due to its unbiasedness. Similarly, FEDSSA outperforms RK because of the bounded-error guarantee.

(Exp#4) Impact of quantization scaling factor. We evaluate the test accuracy by varying the quantization scaling factor (i.e., α) on the CIFAR-10 (0.5) training task. We consider three different compression ratios: $r = 20, 100, 200$. For each compression ratio group, we vary α and analyze its impact on the test accuracy of FEDSSA. We also evaluate the final test accuracy of FEDSSA without the quantization step to understand the overall impact of quantization.

Figure 10 shows that for the CIFAR-10 (0.5) task, when $\alpha = 10^3$, the test accuracy of FEDSSA with quantization drops to 67.71%, 63.97%, and 49.53%, respectively. However, when setting $\alpha \geq 10^4$, the test accuracy of FEDSSA without quantization reaches 87.96%, 86.41%, and 84.60% for $r = 20, 100, 200$, respectively. When using an even larger α (e.g., 10^5, 10^6, and 10^7), the differences in the test accuracy are insignificant, i.e., within 0.4%.

(Exp#5) Effect of periodic rehashing. We evaluate the impact of periodic rehashing on CIFAR-10 (0.5) under the SA setting. We evaluate the test accuracy of FEDSSA and FEDSSA without periodic rehashing (FEDSSA w/o PR).

Figure 11 shows that the test accuracy of FEDSSA consistently outperforms FEDSSA w/o PR. The test accuracy of FEDSSA varies only from 88.24% to 85.08%, with a decrease of 3.16%. In contrast, FEDSSA w/o PR exhibits a significant accuracy drop, ranging from 86.01% to 61.89%, with a decrease of over 24%.

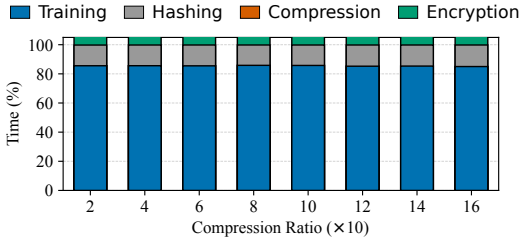
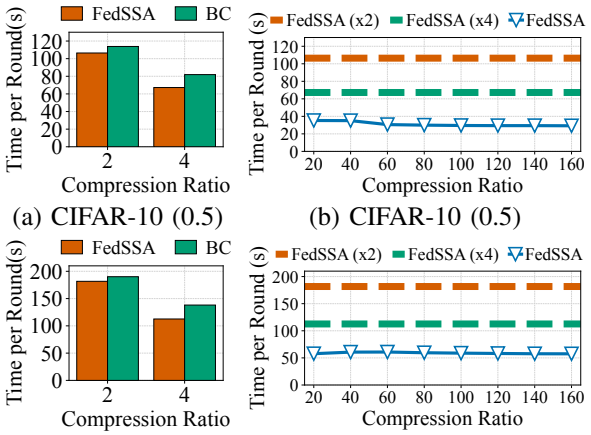


Figure 12: (Exp#6) FEDSSA overhead.



(a) CIFAR-10 (0.5) (b) CIFAR-10 (0.5)

(c) CIFAR-100 (0.5) (d) CIFAR-100 (0.5)

Figure 13: (Exp#7) AHE computational overhead reduction.

(Exp#6) FEDSSA overhead. In the SA setting, we analyze the percentage of local training (Training), compression with QSRHT Sketch, and encryption (Encryption) with SA. We break down the computation time of compression into hash function calculation (Hashing) and other operations except for hash calculation (Compression).

Figure 12 shows that the computation overhead of compression (not shown in the figure) and encryption are small across different r. The hash calculation overhead is small compared to local training, which is acceptable. The majority of computation time is consumed by local training and hash calculation, which takes up around 85% and around 14.8%, respectively.

(Exp#7) AHE computational overhead reduction. We compare the average computation time of FEDSSA and BC at compression ratios of $r = 2$ and $r = 4$. We focusing on CIFAR-10 (0.5) and CIFAR-100 (0.5). We also evaluate FEDSSA's performance under larger compression ratios.

Figures 13(a) and 13(c) show that FEDSSA consistently outperforms BC in terms of computation time at smaller compression ratios. Specifically, at $r = 4$, FEDSSA reduces the average computation time by 17.9% on CIFAR-10 and 18.49% on CIFAR-100 compared to BC. Moreover, Figures 13(b) and 13(d) highlight the trade-off between computation overhead reduction and model accuracy as the compression ratio increases. Higher compression ratios yield further reductions in computation time, but at the cost of accuracy. For example, at $r = 20$, FEDSSA reduces computation time by 56.71% on CIFAR-10 and 58.15% on CIFAR-100 compared to BC with $r = 4$, with only a 2.2% accuracy drop. This remains within acceptable limits (see Exp#2).

VI. RELATED WORK

Gradient quantization. These methods reduce the gradient size by reducing the precision of each gradient element [16], [25], [44], [58], [59]. However, these quantization methods are originally designed for model updates and may not be suitable for the quantization procedure of the sketch. In contrast, the quantization step of QSRHT Sketch can provide a bounded-error guarantee while maintaining the additivity of sketch.

Sketch-based gradient compression. Existing sketch-based compression methods mainly focus on two aspects: (i) assisting quantization or sparsification with sketch [22], [41], [60]; (ii) addressing the computation overhead of sketch in distributed machine learning [42]. Unfortunately, these methods inherit the limitations of gradient quantization or gradient sparsification, which is addressed by FEDSSA. The most similar work to ours is SQuaFL [61], which combines quantization with Count Sketch for compression [47]. However, FEDSSA addresses the following issues that SQuaFL does not consider: (i) adopts QSRHT Sketch with provable quantization scale factor which satisfies additivity (instead of lacking additivity); (ii) providing a bounded-error guarantee (instead of potential numerical overflow in the sketch compression procedure).

Differential privacy. Differential privacy (DP) protects client's model updates by introducing additional noise [62]–[64]. Some studies further codesign gradient compression with specific DP mechanisms [37], [65]–[68]. While these studies focus on alleviating the communication overhead of DP-based methods, FEDSSA primarily supports additivity cryptographic methods, which do not compromise model accuracy for privacy protection [10], [16], [40], [69].

VII. CONCLUSION

In this paper, we propose FEDSSA, a compression framework that can reduce the overhead of additive cryptographic methods in FL while retaining model accuracy. FEDSSA propose QSRHT Sketch to reduce the overhead of these methods with a large compression ratio. FEDSSA also proposes periodic rehashing to correctly fresh the hash functions. Our evaluation shows that FEDSSA can achieve comparable results to state-of-the-art SA-based compressors and AHE-based compressor under the same compression ratio. Furthermore, compared to state-of-the-art AHE-based compressors, FEDSSA decreases the computation time per round by up to 58.15% with an accuracy loss within 2.2%.

ACKNOWLEDGEMENTS

We thank our shepherd, Baochun Li, and the anonymous reviewers for their valuable comments. This work is supported by National Key Research and Development Program of China (2023YFB2904600), National Natural Science Foundation of China (62172007), and Research Grants Council of Hong Kong (GRF 14201523). Qun Huang is the corresponding author.

REFERENCES

[1] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[2] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, "Federated learning for healthcare informatics," *Journal of Healthcare Informatics Research*, vol. 5, pp. 1–19, 2021.

[3] G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated learning for open banking," in *Federated Learning: Privacy and Incentive*. Springer, 2020, pp. 240–254.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of AISTAT*. PMLR, 2017, pp. 1273–1282.

[5] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *arXiv preprint arXiv:1906.08935*, 2019.

[6] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Proc. of NeurIPS*, vol. 33, pp. 16 937–16 947, 2020.

[7] Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora, "Evaluating gradient inversion attacks and defenses in federated learning," *Proc. of NeurIPS*, vol. 34, pp. 7232–7241, 2021.

[8] D. I. Dimitrov, M. Balunović, N. Konstantinov, and M. Vechev, "Data leakage in federated averaging," *arXiv preprint arXiv:2206.12395*, 2022.

[9] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[10] 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 *Proc. of ACM CCS*, 2017, pp. 1175–1191.

[11] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[12] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "Elsa: Secure aggregation for federated learning with malicious actors," in *Proc. of IEEE S&P*. IEEE, 2023, pp. 1961–1979.

[13] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," in *Proc. of ACM CCS*, 2020, pp. 1253–1269.

[14] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E Ali, B. Guler, and S. Avestimehr, "Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning," *Proc. of MLSys*, vol. 4, pp. 694–720, 2022.

[15] 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.

[16] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. of USENIX ATC*, 2020, pp. 493–506.

[17] K. Bonawitz, F. Salehi, J. Konečnỳ, B. McMahan, and M. Gruteser, "Federated learning with autotuned communication-efficient secure aggregation," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2019, pp. 1222–1226.

[18] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Proc. of INTERSPEECH*, vol. 2014, 2014, pp. 1058–1062.

[19] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[20] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," *Proc. of NeurIPS*, vol. 30, pp. 1508–1518, 2017.

[21] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. of AISTAT*. PMLR, 2020, pp. 2021–2031.

[22] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," in *Proc. of ICML*. PMLR, 2020, pp. 8253–8265.

[23] S. Agarwal, H. Wang, S. Venkataraman, and D. Papailiopoulos, "On the utility of gradient compression in distributed training systems," *Proc. of MLSys*, vol. 4, pp. 652–672, 2022.

[24] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," *arXiv preprint arXiv:1704.05021*, 2017.

[25] M. Li, R. B. Basat, S. Vargaftik, C. Lao, K. Xu, X. Tang, M. Mitzenmacher, and M. Yu, "Thc: Accelerating distributed deep learning using tensor homomorphic compression," *arXiv preprint arXiv:2302.08545*, 2023.

[26] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3252–3261.

[27] Y. Lu, P. S. Dhillon, D. Foster, and L. Ungar, "Faster ridge regression via the subsampled randomized hadamard transform," in *Proc. of NeurIPS*, 2013, pp. 369–377.

[28] J. Lacotte, S. Liu, E. Dobriban, and M. Pilanci, "Optimal iterative sketching with the subsampled randomized hadamard transform," in *Proc. of NeurIPS*, 2020, pp. 9725–9735.

[29] Y. Wu, S. Dong, Y. Zhou, Y. Zhao, F. Fu, T. Yang, C. Niu, F. Wu, and B. Cui, "Kvsagg: Secure aggregation of distributed key-value sets," in *Proc. of IEEE ICDE*. IEEE, 2023, pp. 1775–1789.

[30] Y. Zhao, Y. Zhang, Y. Li, Y. Zhou, C. Chen, T. Yang, and B. Cui, "Minmax sampling: A near-optimal global summary for aggregation in the wide area," in *Proc. of ACM SIGMOD*, 2022, pp. 744–758.

[31] S. Wang, "A practical guide to randomized matrix computations with matlab implementations," *arXiv preprint arXiv:1505.07570*, 2015.

[32] D. Page. (2019) How to train your resnet. [Online]. Available: https://myrtle.ai/how-to-train-your-resnet/

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] D. Huba, J. Nguyen, K. Malik, R. Zhu, M. Rabbat, A. Yousefpour, C.-J. Wu, H. Zhan, P. Ustinov, H. Srinivas *et al.*, "Papaya: Practical, private, and scalable federated learning," *Proc. of MLSys*, vol. 4, pp. 814–832, 2022.

[35] Z. Jiang, W. Wang, and R. Chen, "Taming client dropout for distributed differential privacy in federated learning," *arXiv preprint arXiv:2209.12528*, 2022.

[36] J. Zhang, X. Cheng, W. Wang, L. Yang, J. Hu, and K. Chen, "{FLASH}: Towards a high-performance hardware acceleration architecture for cross-silo federated learning," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1057–1079.

[37] W.-N. Chen, C. A. C. Choo, P. Kairouz, and A. T. Suresh, "The fundamental price of secure aggregation in differentially private federated learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 3056–3089.

[38] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[39] W. Jin, Y. Yao, S. Han, C. Joe-Wong, S. Ravi, S. Avestimehr, and C. He, "Fedml-he: An efficient homomorphic-encryption-based privacy-preserving federated learning system," *arXiv preprint arXiv:2303.10837*, 2023.

[40] Z. Zeng, Y. Du, Z. Fang, L. Chen, S. Pu, G. Chen, H. Wang, and Y. Gao, "Flbooster: A unified and efficient platform for federated learning acceleration," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 3140–3153.

[41] N. Ivkin, D. Rothchild, E. Ullah, V. Braverman, I. Stoica, and R. Arora, "Communication-efficient distributed sgd with sketching," in *Proc. of NeurIPS*, 2019, pp. 13 142–13 152.

[42] J. Gui, Y. Song, Z. Wang, C. He, and Q. Huang, "Sk-gradient: Efficient communication for distributed machine learning with data sketch," in *Proc. of IEEE ICDE*. IEEE, 2023, pp. 2372–2385.

[43] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, "Evaluation and optimization of gradient compression for distributed deep learning," *arXiv preprint arXiv:2306.08881*, 2023.

[44] K. Mishchenko, B. Wang, D. Kovalev, and P. Richtárik, "Intsgd: Adaptive floatless compression of stochastic gradients," *arXiv preprint arXiv:2102.08374*, 2021.

[45] S. Vargaftik, R. Ben-Basat, A. Portnoy, G. Mendelson, Y. Ben-Itzhak, and M. Mitzenmacher, "Drive: One-bit distributed mean estimation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 362–377, 2021.

[46] A. Beznosikov, S. Horváth, P. Richtárik, and M. Safaryan, "On biased compression for distributed learning," *Journal of Machine Learning Research*, vol. 24, no. 276, pp. 1–50, 2023.

[47] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. of ICALP*. Springer, 2002, pp. 693–703.

[48] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[49] (2024) Technical report of fedssa. [Online]. Available: https://github.com/fedssa/FedSSA-Technical-Report

[50] H. Yang, M. Fang, and J. Liu, "Achieving linear speedup with partial worker participation in non-iid federated learning," *arXiv preprint arXiv:2101.11203*, 2021.

[51] (2024) Python bindings and wrapper for intel paillier cryptosystem library. [Online]. Available: https://github.com/intel/pailliercryptolib_python

[52] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[53] J. Schmidhuber, S. Hochreiter *et al.*, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.

[54] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proc. of ICML*, 2015, pp. 448–456.

[55] Y. Wu and K. He, "Group normalization," in *Proc. of ECCV*, 2018, pp. 3–19.

[56] K. Hsieh, A. Phanishayee, O. Mutlu, and P. Gibbons, "The non-iid data quagmire of decentralized machine learning," in *Proc. of ICML*. PMLR, 2020, pp. 4387–4398.

[57] S. Shi, X. Chu, K. C. Cheung, and S. See, "Understanding top-k sparsification in distributed deep learning," *arXiv preprint arXiv:1911.08772*, 2019.

[58] A. T. Suresh, X. Y. Felix, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," in *Proc. of ICML*. PMLR, 2017, pp. 3329–3337.

[59] J. Xin, M. Canini, P. Richtárik, and S. Horváth, "Global-qsgd: Practical floatless quantization for distributed learning with theoretical guarantees," *arXiv preprint arXiv:2305.18627*, 2023.

[60] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proc. of ACM SIGMOD*, 2018, pp. 1269–1284.

[61] P. Prakash, J. Ding, M. Shu, J. Wang, W. Xu, and M. Pan, "Squafl: Sketch-quantization inspired communication efficient federated learning," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2021, pp. 350–354.

[62] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. of ACM CCS*, 2016, pp. 308–318.

[63] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 61–66.

[64] Y. Zhao, J. Zhao, M. Yang, T. Wang, N. Wang, L. Lyu, D. Niyato, and K.-Y. Lam, "Local differential privacy-based federated learning for internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8836–8853, 2020.

[65] T. Li, Z. Liu, V. Sekar, and V. Smith, "Privacy for free: Communication-efficient learning with differential privacy using sketches," *arXiv preprint arXiv:1911.00972*, 2019.

[66] B. Wang, F. Wu, Y. Long, L. Rimanic, C. Zhang, and B. Li, "Datalens: Scalable privacy preserving training via gradient compression and aggregation," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2146–2168.

[67] R. Kerkouche, G. Ács, C. Castelluccia, and P. Genevès, "Compression boosts differentially private federated learning," in *Proc. of IEEE EuroS&P*. IEEE, 2021, pp. 304–318.

[68] Y. Youn, Z. Hu, J. Ziani, and J. Abernethy, "Randomized quantization is all you need for differential privacy in federated learning," *arXiv preprint arXiv:2306.11913*, 2023.

[69] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1651–1669.