Exploiting the Exponent of Floating-Point: A Novel Pathway to Efficient Federated Learning

Goran Saman Nariman [1, 2 *], Hozan Khalid Hamarashid [3,4]
[1] Department of Computer Science, College of Science, Charmo University, Sulaimani, Iraq
[2] Department of Information Technology, College of Science and Technology, University of Human Development, Kurdistan Region, Iraq
[3] Information Technology Department, Computer Science Institute, Sulaimani Polytechnic University, Sulaimani 46001, Iraq
[4]Computer Engineering Department, Tishk International University, Kurdistan Region, Iraq
E-mail: goran.saman@chu.edu.iq, goran.nariman@uhd.edu.iq, Hozan.Khalid@spu.edu.iq
[*]Corresponding author

Keywords: federated learning, communication efficiency, model compression, parameter sparsification, exponent-based pruning

Received: April 9, 2025

Federated Learning (FL) enables decentralized model training by having clients process local data and transmit only learned updates to a central server, but faces significant communication bottlenecks due to frequent exchanges of high-dimensional model parameters. While existing compression techniques such as sparsification and quantization reduce overhead, they fail to fully exploit the structural properties of floating-point representations. This paper introduces a novel compression strategy that innovatively exploits the exponent field of float16 representations to encode sequences of negligible parameter updates, achieving substantial communication reduction while preserving model integrity. The proposed method operates through four main steps: (1) client-side model subtraction between local and global parameters, (2) downcasting to float16, (3) threshold-based pruning to remove insignificant values, and (4) exponent-based encoding to compactly represent sequences of negligible updates, followed by server-side decompression through exponent extraction and sequence reconstruction. On MNIST, the method achieves an 88.4% size reduction (threshold 0.001) with merely 0.2% accuracy loss versus baseline, while a lighter threshold (0.0001) improves accuracy by 0.1% at 64.7% compression; on CIFAR-10, it yields 64% compression with maintained accuracy (+0.1%) and 42% faster convergence through critical weight preservation, with lighter thresholds (0.0001) achieving 51.4% size reduction and 1.2% accuracy improvement. The technique's computational efficiency and compatibility with existing FL frameworks make it particularly suitable for resource-constrained edge environments, bridging a critical gap in communication-efficient FL through innovative use of floating-point exponent manipulation for scalable, privacy-preserving distributed learning.

Povzetek: Prispevek predstavlja kompresijsko metodo za federativno učenje, ki izkorišča eksponentno strukturo 16-bitnih plavajočih števil. Predlagani pristop omogoča znatno zmanjšanje prenosa podatkov med odjemalci in strežnikom, pri tem pa ohranja natančnost modela in ne potrebuje dodatnih metapodatkov.

1 Introduction

Federated Learning (FL) has emerged as a transformative paradigm for training machine learning models across decentralized devices while preserving data privacy [1]. In FL, multiple clients collaboratively train a shared global model under the coordination of a central server, without sharing their raw data. This approach is particularly crucial for domains such as healthcare, where patient data privacy is paramount [2], with recent studies demonstrating hierarchical FL frameworks tailored to sensitive medical and pharmacy data [3]; Agriculture, where sensitive data such as crop yields, soil conditions, and farming practices must remain protected [4]; and the Internet of Things (IoT), where edge devices generate vast amounts of data but have limited computational and communication resources [5, 6]. Due to its privacy-preserving nature, FL has been explored in recommendation systems [7, 8], transportation and Internet of Vehicles (IoV) [9-12], including security applications such as trust-based attack detection in vehicular networks.

While FL addresses critical privacy concerns, it introduces significant communication overhead, as model updates, often comprising millions of parameters, must be transmitted repeatedly between clients and the server during training, a challenge extensively reviewed in recent surveys that classify communication reduction strategies and highlight their trade-offs [13, 14]. This communication bottleneck is exacerbated in resource-constrained environments, such as mobile devices or edge networks [15], where bandwidth and energy are limited [16]. Consequently, reducing communication overhead has become a central challenge in scaling FL to real-world applications.

To address the communication bottleneck in distributed learning, according to [14, 17], the techniques addressing this issue can be broadly classified into three main strategies: **round reduction**, **compression**, and **hybrid methods**.

- Round reduction techniques aim to minimize the number of communication rounds by optimizing client selection, reducing the frequency of updates, or leveraging local computations to limit the need for frequent synchronization.
- Compression techniques focus on reducing the size of the data transmitted in each round, thereby decreasing the overall communication overhead.
- Hybrid methods combine elements of both round reduction and compression to achieve even greater efficiency by simultaneously reducing the number of rounds and the volume of data exchanged.

Among these, compression techniques have garnered significant attention due to their ability to directly reduce the volume of data exchanged between clients and the server. Compression methods can be further categorized into three types [16, 17]:

- **Sparsification**: This technique reduces the number of non-zero elements in the model updates, transmitting only the most significant gradients or parameters.
- **Quantization**: This approach reduces the precision of the parameters, often by representing them with fewer bits, thereby decreasing the size of each transmitted value.
- **Hybrid compression**: These methods combine sparsification and quantization to achieve both sparsity and reduced precision simultaneously, further enhancing communication efficiency.

In this work, we focus on quantization and sparsification, two of the most widely used compression techniques in FL. Specifically, our approach falls under structured sparsification, where we target the reduction of redundant or near-zero parameters in the model updates. While existing methods have explored various forms of sparsification and quantization, none have fully exploited the potential of the exponent field in floating-point representations, particularly in the float16 format. This represents a significant gap in the literature, as the exponent field offers a compact yet versatile structure for encoding numerical data, which can be leveraged to further reduce communication overhead. In the current work, the model parameters are exchanged between clients and the server, and compression is applied directly to these parameters. Alternatively, other forms of information such as gradients, federated features, or distilled knowledge can be transmitted, as explored in [18].

The float16 format allocates 16 bits as follows: 1 bit for the sign, 5 bits for the exponent, and 10 bits for the mantissa [19, 20]. The 5-bit exponent field, capable of representing raw values from 0 to 31 (biased to yield actual exponents from -14 to $+15$, with 0 and 31 reserved for special cases such as zero, denormalized numbers, infinity, and NaN), provides a compact yet versatile structure for encoding numerical data. We exploit this structure to encode sequences of up to 15 consecutive zero or near-zero parameter updates, a strategy that, to the best of our knowledge, has not been previously explored for enhancing communication efficiency in FL. By embedding the count of negligible values into the exponent of significant parameters, our approach uncovers an underutilized aspect of the float16 representation, achieving substantial reductions in transmitted data size while preserving the integrity of model updates.

The proposed method operates in two main phases: client-side compression and server-side decompression. On the client side, model updates are compressed by identifying and encoding sequences of insignificant values (those below a predefined threshold) into the exponent bits of significant parameters. This process involves model subtraction, downcasting to float16, flattening of parameters, and compression of insignificant values. On the server side, the compressed updates are decompressed by reconstructing the original parameter sequences, reshaping the model, upcasting to the original precision, and aggregating the updates to refine the global model. This end-to-end framework ensures that the benefits of compression are realized without compromising the accuracy or convergence of the global model.

The contributions of this work are as follows:

1. **Novel compression technique**: We introduce a novel compression strategy that exploits the exponent bits of the float16 format to encode sequences of negligible parameter updates, significantly reducing communication overhead in FL.
2. **Efficient encoding:** By embedding the count of insignificant values into the exponent of significant parameters, our method achieves a compact representation of model updates without requiring additional metadata or complex encoding schemes.
3. **Preservation of model integrity**: The proposed technique preserves the sparsity pattern and precision of model updates, ensuring that the global model converges accurately.
4. **Scalability**: The method is designed to be computationally lightweight, making it suitable for resource-constrained environments such as mobile and edge devices.

Research goals and hypotheses

The overarching goal of this study is to design a communication-efficient FL compression technique that reduces transmission overhead while preserving model accuracy and convergence. To guide the work, we formulate the following research questions:

RQ1: Can exponent encoding in float16 be exploited to compactly represent sparsity information in model updates without requiring external metadata?

RQ2: How does the proposed approach compare with state-of-the-art compression techniques in terms of communication savings and accuracy retention across different datasets?

RQ3: How do dataset complexity and model architecture influence the achievable compression ratio when using exponent encoding?

From these questions, we hypothesize that:

H1: Embedding run-length counts into the float16 exponent can achieve substantial communication reduction while preserving model fidelity.

H2: The proposed method will yield compression rates competitive with or superior to existing approaches (e.g., ResFed, FLCP) while avoiding their metadata or cryptographic overhead.

H3: Compression performance will vary by dataset, with simpler datasets (e.g., MNIST) yielding higher sparsity and thus higher compression than more complex datasets (e.g., CIFAR-10).

To evaluate these hypotheses, we assess three outcome metrics: (1) test accuracy delta relative to baseline models, (2) convergence speed measured by the number of rounds to reach target accuracy, and (3) communication efficiency expressed as percentage size reduction of transmitted updates.

This paper is organized as follows: Section 2 reviews related work in communication-efficient FL and floating-point compression techniques. Section 3 presents the proposed methodology, including client-side compression and server-side decompression. Section 4 evaluates the performance of the proposed approach, and Section 5 concludes the paper with a discussion of future directions.

2 Literature review

The issue of communication overhead in FL has led to extensive research on compression techniques aimed at reducing the size of transmitted model updates. Numerous studies, such as [14, 17], classify these methods into categories including sparsification, quantization, tensor decomposition, and hybrid approaches, each balancing compression efficiency, computational complexity, and accuracy retention. Below, we review representative methods most closely related to our work.

The work in [21] introduces model compression via dynamic rank reduction in tensor decomposition. By constraining deep networks to operate within low-rank spaces, the method achieves a 13.96× reduction in communication while preserving accuracy, making it suitable for resource-constrained devices. Similarly, magnitude-based pruning [20], removes parameters at 30%, 50%, and 70% levels, yielding size reductions of approximately 0.23×, 0.40×, and 0.58×, though at the cost of accuracy drops of 1–3%. To improve pruning adaptivity, the PruneFL framework [22] integrates client-side and in-training pruning, achieving up to 50% communication reduction while keeping accuracy within 0.5% of baseline performance.

Beyond pruning, quantization techniques offer another path. The Adaptive Quantized Gradient (AQG) method [23] improves FL efficiency by adaptively tuning the number of quantization bits according to local gradient updates, thereby exploiting data heterogeneity among clients. Unlike fixed-bit schemes such as QSGD, AQG dynamically lowers communication for slowly varying gradients while allocating more precision to informative updates. The authors further propose an augmented AQG variant robust to client dropouts, which uses gradient amplification to maintain unbiased aggregation. Experimental results confirm 18–50% reduction in transmission cost compared to QSGD and LAQ, without loss of convergence, even under non-IID distributions and dropout rates up to 90%.

Complementary strategies include Adaptive Parameter Freezing (APF) [24] , which freezes stable weights during training and reduces data transfer by over 60%, and ResFed [25] , which compresses residuals rather than raw weights, reporting >700× per-round compression on CIFAR-10 and reducing total communication volume by ~99% with minimal accuracy loss. Meanwhile, the FLCP framework [26] integrates adaptive weight compression with strong privacy-preserving measures such as homomorphic encryption and differential privacy. By tailoring compression rates per client and encrypting updates before aggregation, FLCP simultaneously reduces bandwidth consumption and mitigates risks of gradient leakage. Results show that FLCP achieves 87–89% reduction in communication cost, while preserving accuracy and offering formal privacy guarantees under non-IID conditions, demonstrating its suitability for secure large-scale FL deployments.

In summary, while prior methods such as pruning, quantization, residual compression, and privacy-preserving frameworks achieve 15–99% communication savings, they often require auxiliary metadata, introduce lossy coding, or add cryptographic overhead. As shown in Table 1, none exploit the exponent field of float16 to encode run-lengths of negligible parameters. Our method fills this gap by embedding compression information directly into float16 exponents, thereby achieving substantial reductions without accuracy degradation or metadata, and offering a lightweight approach compatible with existing FL frameworks.

3 Methodology

The proposed FL framework introduces a communication-efficient strategy by compressing model updates transmitted from clients to the server. This compression technique leverages the inherent sparsity in parameter updates and exploits the structure of floating-point representations to encode information about negligible values. The methodology is divided into client-side and server-side procedures, which are detailed below, and Figure 1 provides a visual overview of the entire process.

Figure 1 illustrates the end-to-end workflow of the proposed framework, including client-side compression (model subtraction, downcasting, flattening, and negligible value compression) and server-side decompression (sequence reconstruction, model reshaping, upcasting, client-specific model restoration,

Table 1: Comparison of FL communication reduction techniques.

Approaches and Ref.	**Key mechanism**	**Compression Rate**	**Accuracy Retention**	**Dataset**	**# Clients**	**Notable features**
Rank Reduction in Tensor Decomposition [18]	Enforces a low-rank constraint via iterative rank adaptation	13.96× reduction	Reporting gaining 0.88% accuracy	MNIST, CIFAR-10/100, ImageNet.	N/R	Low-rank adaptation; reduces storage and inference time
Magnitude-Based Pruning [19]	Prunes model parameters at levels of 30%, 50%, and 70% based on magnitude	Size reductions of ~0.23×, 0.40×, and 0.58×	CIFAR-10: 86.02% vs 86.98% baseline (Δ≈-0.96%)	CIFAR-10 (wireless channel testbed)	N/R	Simple pruning; channel-aware evaluation
PruneFL [20]	Combines initial client-side pruning with adaptive in-training pruning	Up to 50% reduction	Maintains baseline accuracy within ~0.5%	FEMNIST, CIFAR-10, ImageNet-100	10	Dynamic client pruning during training
APF [22]	Freeze/unfreeze stabilized parameters	>**60%** less data transferred	"Without compromising convergence"; sometimes improved	LeNet-5, LSTM, ResNet-18	**50**	Freezes stable parameters; long-term efficiency
ResFed [23]	Residual-based compressed updates (sparsify + quantize)	15% higher compression ratio than QSFL	Minimal accuracy impact	CIFAR-10 (non-IID)	**10**	Residual compression; balances accuracy and comms
FLCP [24]	Integrates adaptive weight compression with hybrid privacy (HE + DP)	87–89% reduction	Accuracy ≥ baseline	MNIST, CIFAR-10	N/R	Adds privacy with strong comm. reduction
Proposed Approach	Embed run-length into float16 exponent (metadata-free)	**88.4%** (MNIST, τ=0.001), **64.7%** (CIFAR-10, τ=0.001)	MNIST: 96.5% vs 96.7% (Δ≈-0.2%); CIFAR-10: 55.39% vs 54.73% (Δ≈+0.66%)	MNIST, CIFAR-10	**10**	Metadata-free; high compression with minimal accuracy loss

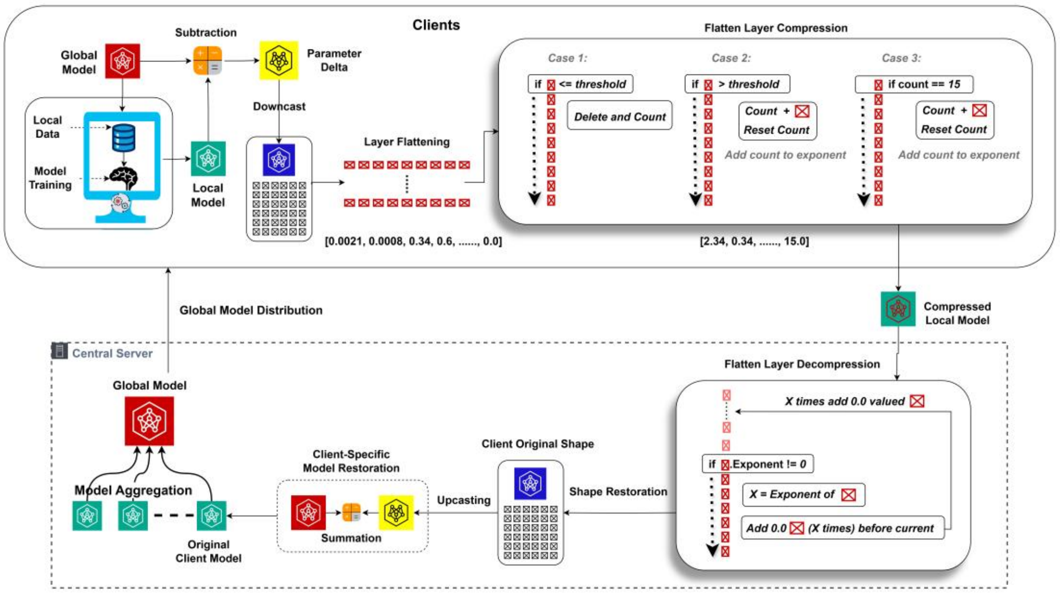

Figure 1: Graphical abstract of the proposed compression technique in FL.

Algorithm 1: Client-Side Compression

Input:

- θ_{global}Global model parameters (dict of tensors, float32 precision)
- θ_{local}: Locally trained client model parameters (dict of tensors, float32 precision)
- : Threshold for insignificant values (scalar, e.g., 0.001)
- M: Maximum zero sequence length (scalar, e.g., 15)

Output:

- : Compressed subtracted model updates (dict of lists of float16-encoded values)

Algorithm Client_Compression($\boldsymbol{\theta_{global}}$, $\boldsymbol{\theta_{local}}$, τ, M)

```
1.  Initialize SM = {}  // Empty dict for subtracted model
2.  For each layer key k in $\theta_{local}$:
3.      SM[k] ← $\theta_{local}$[k] - $\theta_{global}$[k]  // Element-wise subtraction (float32)
4.      SM[k] ← SM[k].to(float16)  // Downcast to float16
5.      flat_SM[k] ← flatten(SM[k])  // Flatten to 1D tensor
6.      compressed_layer ← []  // List for compressed values
7.      zero_count ← 0
8.      For each value v in flat_SM[k]:
9.          If |v| < τ:
10.             zero_count ← zero_count + 1
11.             If zero_count >= M:
12.                 Append float(zero_count) to compressed_layer
13                  zero_count ← 0
14.             Else:
15.                 If zero_count > 0:
16.                     Append float(zero_count) to compressed_layer
17.                     zero_count ← 0
18.                 Append v to compressed_layer
19.     If zero_count > 0:
20.         Append float(zero_count) to compressed_layer  // Encode trailing zeros
21.     SM_comp[k] ← compressed_layer  // Store as list of float16-encoded values
22. Return SM_comp
```

and aggregation). The process highlights the communication-efficient exchange of compressed model updates between clients and the server.

3.1 Client side

On the client side, the following steps are executed in each round of the FL process to compress and prepare model updates for transmission to the server. The complete client-side procedure is summarized in Algorithm 1, which outlines the step-by-step process of local model subtraction, threshold-based pruning, and exponent-encoded run-length compression of negligible updates before transmission to the server.

3.1.1 Model subtraction

At the beginning of each round, each client computes the difference between its locally trained model parameters and the global model parameters received from the server. This difference, referred to as the Subtracted Model (SM), represents the updates made during local training. The subtraction is performed element-wise across all layers of the model, including both weights and biases. Mathematically, for each parameter tensor W_{local} in the locally trained model and the corresponding tensor W_{global} in the global model, the SM is computed as $W_{SM} = W_{local} - W_{global}$.

This step ensures that only the updates made by the client are captured, effectively nullifying non-updated or minimally updated parameters based on the most recent global model. The resulting SM is then passed to the next steps for further processing and compression.

3.1.2 Down casting to Float16

After computing the SM, it is converted from its original high-precision format (float32) to a lower-precision 16-bit floating-point representation (float16). This downcasting is performed after subtraction to preserve the accuracy of the update computation. Performing the subtraction in float32 avoids precision loss that could occur due to rounding errors or the limited range of float16.
Downcasting beforehand might degrade the quality of the SM, as critical differences between the local and global models could be lost. By maintaining full precision during subtraction, the SM accurately reflects the updates, ensuring the integrity of the learning process. The conversion to float16 then reduces the memory footprint of the updates, enabling efficient compression and transmission to the server while retaining sufficient precision for effective aggregation.

3.1.3 Flattening model parameters and biases

To facilitate the subsequent compression process, each layer of the SM, including both weights and biases, is flattened into a one-dimensional sequence. This transformation converts multidimensional parameter tensors (e.g., convolutional kernels or fully connected weights) and bias vectors into linear arrays.

Flattening simplifies the handling of the data, enabling a uniform compression procedure to be applied across all parameters, irrespective of their original structure. This step is crucial as it transforms the updates into a continuous stream of values, making it easier to identify and process sequences of negligible parameters during the subsequent compression phase.

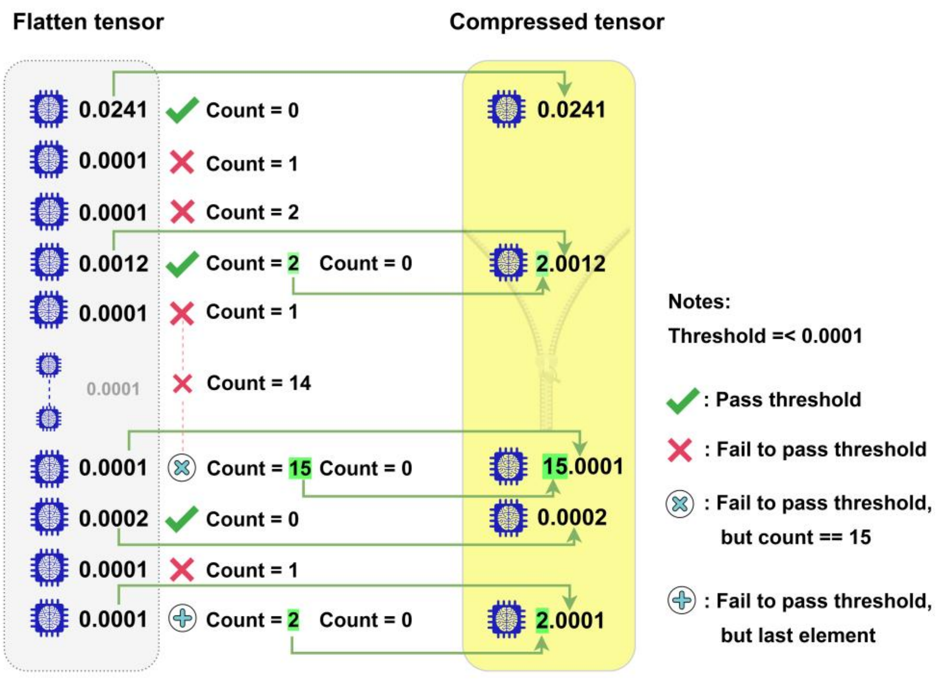

Figure 2: Visualization of the parameter compression process exploiting the floating-point exponent section.

3.1.4 Compression of insignificant values

The core of the proposed compression strategy involves processing each flattened layer to identify and encode sequences of insignificant values—those with an absolute magnitude below a predefined threshold. This step is critical for reducing the size of model updates while preserving the integrity of significant updates. The process works as follows, and Figure 2 visually depicts the parameter compression process, exploiting the exponent section of the floating-point representation.

1. **Iteration through flattened sequence:** The method iterates through each value in the flattened sequence of parameters and biases.
2. **Identification of insignificant values:** If a value's absolute magnitude is below the threshold, it is considered insignificant and removed from the sequence. A counter tracks the number of consecutive insignificant values removed.
3. **Encoding significant values:** When a param value exceeds the threshold, the accumulated count of consecutive insignificant values is encoded into the representation of this significant value. Specifically, the count is embedded into the floating-point exponent section of the significant value.

- Float16 uses 1 sign bit, 5 exponent bits, and 10 mantissa bits. We embed the count (0–15) of skipped values into the exponent field by repurposing 4 bits, enabling run-length encoding without external metadata.
- If the count of consecutive insignificant values reaches 15, the number 15 is added to the exponent of the current value (even if it is below the threshold). The counter is then reset to zero, and the process continues iterating through the remaining values.
- If the count is less than 15 when interrupted by a significant value, the count is embedded into the floating-point exponent section of the significant value.

4. **Final check for remaining count:** Before the loop ends, the counter is checked one final time. If the counter value is greater than 0 (but less than 15), it is added to the exponent section of the last parameter in the sequence, even if the last parameter is below the threshold. This ensures that no remaining insignificant values are left unprocessed.

This approach efficiently compresses sequences of insignificant values into a compact form. By embedding the count of insignificant values into the exponent of significant values, the method reduces the data size while preserving the relative positions of significant updates in the sequence. This ensures that the server can accurately reconstruct the original updates during decompression.

Although this embedding mechanism effectively captures the position of insignificant parameters, it is important to note that the process is not strictly lossless. Two controlled sources of information reduction occur.

First, all parameters whose absolute magnitude is below the predefined threshold τ are reconstructed as exact zeros during decompression; these elements are intentionally discarded to reduce communication overhead while having negligible impact on model accuracy. Second, the act of overwriting the float16 exponent field with the run-length count slightly alters the numeric value of the specific parameter whose exponent carries the count. This modification affects only that individual parameter—its adjacent values retain their original exponent–mantissa composition. The perturbation introduced by this overwriting remains within the quantization scale of the float16 format and was empirically found to have no measurable effect on convergence or final accuracy.

Overall, the encoding can therefore be characterized as a controlled lossy compression scheme that trades minimal numerical deviation for substantial communication savings while maintaining faithful reconstruction of the significant model structure.

To ensure stable encoding and decoding, anomalous values such as NaNs, infinities, and subnormals are clipped and replaced with zeros prior to compression. This prevents disruptions in tensor layout and maintains decoder stability.

3.2 Server side

When the server receives the compressed SMs from the participating clients, it performs the following steps to reconstruct the client updates, restore the original model shapes, and aggregate the updates to produce a new global model. Algorithm 2 presents the corresponding server-side operations, including decompression of received updates, reconstruction of client models, and global aggregation through the FedAvg strategy.

3.2.1 Receiving compressed updates

The server begins by decompressing the flattened layers of the received SMs to restore the removed parameters to their proper locations. This process involves iterating through the compressed sequence and reconstructing the original number of elements in the flattened tensor. The decompression algorithm works as follows:

1. **Iteration through compressed sequence**: The server iterates through each value in the compressed sequence.
2. **Exponent extraction, zero insertion, and reconstruction:**

- For each parameter, the server checks the absolute value of its exponent. If the exponent is greater than zero, it is extracted, and the exponent of the parameter is reset to zero.
- The extracted exponent value indicates the number of zero parameters that were removed before the current parameter during compression.
- The server inserts the corresponding number of zero values into the sequence before the current parameter. This ensures that the original number of elements is restored, and the arrangement of

Algorithm 2: Server-side decompression and aggregation

Input:

- $\{SM_{comp}^{(1)}, \ldots, SM_{comp}^{C}\}$: Set of C compressed updates from clients (list of dicts of lists)
- θ_{global}:Current global model parameters (dict of tensors, float32 precision)
- Shapes: Original tensor shapes for each layer (dict)**Output:**
- θ_{global}^{new}: Updated global model parameters (dict of tensors, float32 precision)
- $\{\theta_{client}^{(i)}\}_{i=1}^{C}$: Restored client models (list of dicts, for synchronization)

Algorithm Server_Decompression_Aggregate($\{SM_{comp}^{i}\}$, θ_{global}, Shapes)

```
1.  1. Initialize $\theta_{clients}$ = []          // List for restored client models
2.  For each client i in 1 to C:
3.      Initialize $SM_{decomp}$ = {}          // Empty dict for decompressed updates
4.      For each layer key k in $SM_{comp}^{(i)}$:
5.          decompressed flat = []          // List for reconstructed flat tensor
6.          j ← 0          // Index in compressed list
7.          While j < length($SM_{comp}^{(i)}$[k]):
8.              v ← $SM_{comp}^{(i)}$ [k][j]
9.              If isinstance(v, float) and v.is_integer() and v >= 1.0:
10.                 z_count ← int(v)
11.                 Append z_count * [0.0] to decompressed_flat
12.             Else:
13.                 Append v to decompressed_flat
14.             j ← j + 1
15.         orig_size ← product(Shapes[k])
16.         If length(decompressed_flat) ≠ orig_size:
17.             Raise Error("Decompression mismatch")
18.         $SM_{decomp}$ [k] ← reshape(decompressed_flat, Shapes[k]).to(float32)
19.     $\theta_{client}^{(i)}$ ← {$k$: $\theta_{global}$[k] + $SM_{decomp}$[k] for each $k$}  // Restore client model
20.     Append $\theta_{client}^{(i)}$ to $\theta_{clients}$
21. Perform FedAvg aggregation to update $\theta_{global}^{new}$ and synchronize $\theta_{clients}$
22. Return $\theta_{global}^{new}$, $\theta_{clients}$
```

zeros and non-zero values (i.e., the sparsity pattern) is preserved.

3. **Continuation of the process:** The server moves to the next parameter in the sequence and repeats the process until the entire sequence is decompressed.

By completing this step, the server reconstructs the client's SM in its flattened form, restoring the original number of elements and the sequence of parameters and zeros.

3.2.2 Reconstruction of model shapes

The decompressed SM from the previous step is in a flattened (1D) form. To align with the original structure of the global and client models, the server reshapes the flattened tensor back into its original multidimensional form. This step ensures that the weights and biases are correctly aligned with their corresponding layers in the model.

The reshaping process is performed layer-by-layer, using the original shapes of the model parameters (e.g., convolutional kernels, fully connected weights, and biases). This restores the structural integrity of the SM, preparing it for the next steps.

3.2.3 Upcasting to original format

The reshaped SM is currently in a 16-bit floating-point (float16) format, which was used during compression to reduce communication overhead. To match the original precision of the global and client models, the server upcasts the SM to its original format (typically 32-bit floating-point, or float32).

This upcasting ensures that the precision of the model updates is preserved, maintaining the accuracy of the learning process. The upcasted SM is now ready for integration into the client-specific model.

3.2.4 Client-specific model restoration

At this stage, the SM represents only the differences between the client's locally trained model and the global model shared at the start of the round. To restore the complete client model, the server adds the decompressed and upcasted SM to the global model parameters. Mathematically, this is expressed as:

$$W_{original\ client} = W_{received} + W_{global}$$

where:

- $W_{original\ client}$ is the restored client model
- $W_{received}$ is the decompressed and upcasted SM, and
- W_{global} is the global model shared at the start of the round.

This step ensures that the server accurately reconstructs the client's locally trained model, including all updates made during local training.

3.2.5 Aggregation of client updates

Finally, the server aggregates the reconstructed client models to update the global model. This is achieved by averaging the parameters across all participating clients for each layer. The aggregation process is mathematically expressed as:

$$W_{new\ original} = \frac{1}{N}\sum_{i=1}^{N} W_{original\ client_i}$$

where N is the number of participating clients. The resulting averaged parameters form the new global model, which is then broadcast to all clients for the next round of FL. This step ensures that the compressed updates are fully integrated into the global learning process, maintaining the collaborative nature of FL while benefiting from reduced communication overhead.

4 Implementation

This section provides an overview of the implementation of the proposed communication-efficient FL framework, which leverages parameter compression to reduce communication overhead. The implementation is structured into the following subsections: dataset preparation, client distribution, model architecture, threshold tuning, and evaluation metrics. The framework is designed to operate in a realistic FL environment, where data is distributed in a non-IID manner across clients, and communication efficiency is critical.

4.1 Dataset preparation and client distribution

To evaluate the proposed FL framework under realistic and diverse conditions, two benchmark datasets are employed: MNIST and CIFAR-10. The MNIST dataset comprises 60,000 training images and 10,000 test images of handwritten digits (0–9), each represented as a 28x28 grayscale image. In contrast, CIFAR-10 consists of 50,000 training images and 10,000 test images of natural objects across 10 classes (e.g., airplane, automobile, bird), originally provided as 32x32 RGB images with three color channels.

The MNIST dataset is partitioned in a pathological non-IID manner across 100 clients to mimic real-world data distribution, where each client has data from only a subset of classes. Specifically, the dataset is divided such that each client receives data from only 2 out of the 10 possible classes. This non-IID distribution is achieved by splitting the dataset into 5 groups, where each group contains data from 2 classes, and further dividing each group into 20 clients. This ensures that each client has data from only those 2 classes, creating a challenging and heterogeneous data distribution. The FL system consists of 100 clients, with 10 clients selected at random in each communication round to participate in training. This random selection ensures diversity in the updates received by the server and prevents bias toward specific clients. Each selected client trains its local model for 5 epochs using a batch size of 32. The local updates are then compressed and transmitted to the server for aggregation.

For the CIFAR-10 dataset, we adopted a partitioning strategy inspired by the Flower framework [27], widely recognized as an effective framework for FL [28]. The dataset is split in a pathological non-IID manner over 50 communication rounds, with all 10 clients participating in each round. Each client receives data from only two distinct classes, achieved by dividing the dataset into five subsets (each with two classes) and distributing them across two clients. This setup reflects real-world FL challenges, ensuring heterogeneity while maintaining computational efficiency and accuracy.

4.2 Model architecture

The model architecture used in this work is a Convolutional Neural Network (CNN) designed for the MNIST dataset. The architecture begins with three convolutional layers: the first layer has 1 input channel and 32 output channels, the second layer has 32 input channels and 64 output channels, and the third layer has 64 input channels and 128 output channels. All convolutional layers use a kernel size of 3, a stride of 2, and padding of 1, followed by ReLU activation functions. The output of the final convolutional layer is flattened and passed through two fully connected layers: the first with 2048 input features and 128 output features, and the second with 128 input features and 10 output features (one for each class).

The model uses a log-SoftMax output layer for classification. This architecture is lightweight yet effective for the MNIST and CIFAR-10 classification task, making it suitable for resource-constrained FL environments. The model is implemented using the PyTorch framework, and its parameters are updated using Stochastic Gradient Descent (SGD) with a learning rate of 0.1.

4.3 Threshold tuning

The compression technique relies on a predefined threshold to identify insignificant parameter updates. Parameters with absolute values below this threshold are considered negligible and are compressed. To determine the optimal threshold, multiple values were evaluated, including 0.01 to 0.00001. These thresholds were chosen to explore the trade-off between compression efficiency and model accuracy. A higher threshold (e.g., 0.01) results in more aggressive compression, as a larger number of parameters are considered insignificant. Conversely, a lower threshold (e.g., 0.00001) retains more parameters, improving model accuracy but reducing compression efficiency. The selection of the threshold is a critical aspect of the compression process, as it directly impacts the balance between communication efficiency and model performance.

4.4 Evaluation metrics

The performance of the proposed framework is evaluated using two key metrics: classification accuracy and size reduction.

1. **Test Accuracy:** The classification accuracy of the global model on the test set is measured after each communication round, reflecting the effectiveness of the FL process in learning from decentralized data.
2. **Compression efficiency:** The size reduction metric quantifies the communication efficiency achieved by the proposed compression technique. The percentage reduction in transmitted data size, calculated as:
3.

$$Reduction\ (\%) = \left(1 - \frac{Compressed\ Size\ (KB)}{Original\ Size\ (KB)}\right) \times 100$$

It is computed by comparing the size of the original client model updates (in float32) with the size of the compressed client updates (in float16 with exponent encoding). The average reduction percentage across all clients is reported for each round, providing a measure of the framework's communication efficiency.

4.5 Implementation validation and framework compatibility

To validate the proposed method and assess framework compatibility, two complementary implementations were developed.

The first integrates the exponent-encoded compression scheme into the Flower federated learning framework (v1.8), demonstrating its ability to operate within existing FL infrastructures. Only minor modifications were required to the client–server communication process by inserting a lightweight encoder–decoder module at the message-serialization stage. Each client compresses its model delta using the exponent-encoded float16 run-length scheme and transmits it as a binary payload, while the server decompresses, reconstructs the float32 model, and applies a customized FedAvg aggregation. This confirms full compatibility with standard FL orchestration routines.

A second stand-alone PyTorch implementation was also developed to provide a framework-independent setup and ensure full reproducibility. It includes the complete source code for both the proposed compression and baseline models on MNIST and CIFAR-10.

Both versions are publicly available for verification and replication:

- Flower-based: https://github.com/Goran1984/fl-exp16
- Stand-alone: https://github.com/Goran1984/fl-exp16-suite-pro.

4.6 Federated training protocol

The FL process consists of 100 and 50 communication rounds for the MNIST and CIFAR-10 datasets, respectively, with 10 clients randomly selected in each round. Each client trains its local model for 5 epochs using a batch size of 32. The server aggregates updates via Federated Averaging (FedAvg), with decompression and reshaping ensuring compatibility with the global model's architecture. Metrics are logged at each round, including average training loss, test accuracy, and compression efficiency.

5 Results and evaluations

This section presents an extensive evaluation of the proposed compression technique using the MNIST dataset. The analysis focuses on assessing the trade-offs between communication efficiency and model accuracy under different threshold settings for identifying negligible parameter updates. All experiments were repeated over ten runs under identical conditions. The observed variations in accuracy and compression ratio were within ±0.1 %, confirming the robustness of the results; hence, only average values are presented.

5.1 Results on MNIST

Table 2 and Figures 3 and 4 present the results for MNIST under different threshold settings. The "Base" configuration corresponds to standard float32 transmission without pruning or exponent encoding. As shown, increasing the pruning threshold leads to higher

Table 2: Accuracy and size reduction (%, remaining KB) of the proposed method on MNIST across pruning thresholds.

Thresholds	Size reduction (%), Size (KB)	Accuracy (%)	# Rounds
0.01	96.4, 48.96	91.9	91
0.001	88.4, 157.76	96.5	
0.0001	64.7, 480.32	96.8	86
0.00001	54.8, 613.28	96.7	
Base	0, 1360.0	96.7	

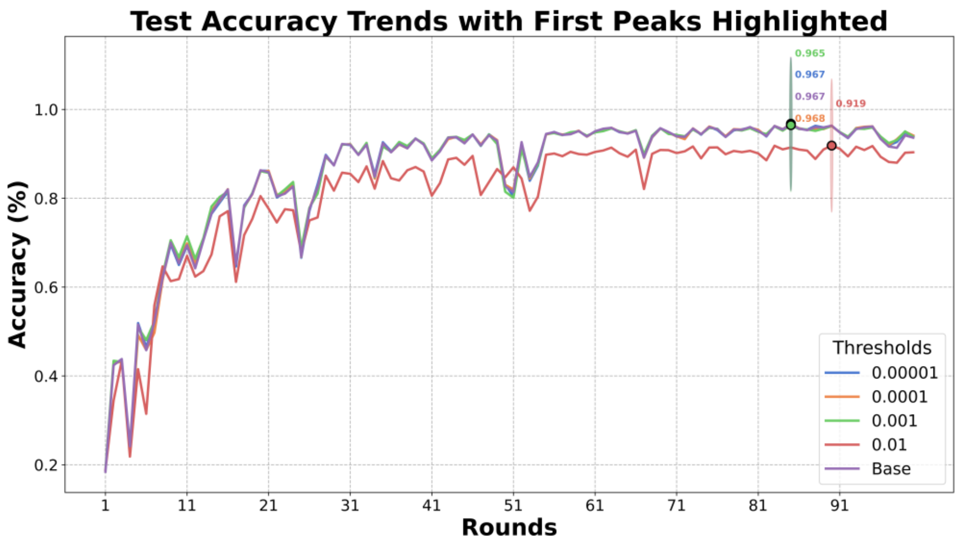

Figure 3: Test accuracy results across FL rounds for different used thresholds on MNIST dataset.

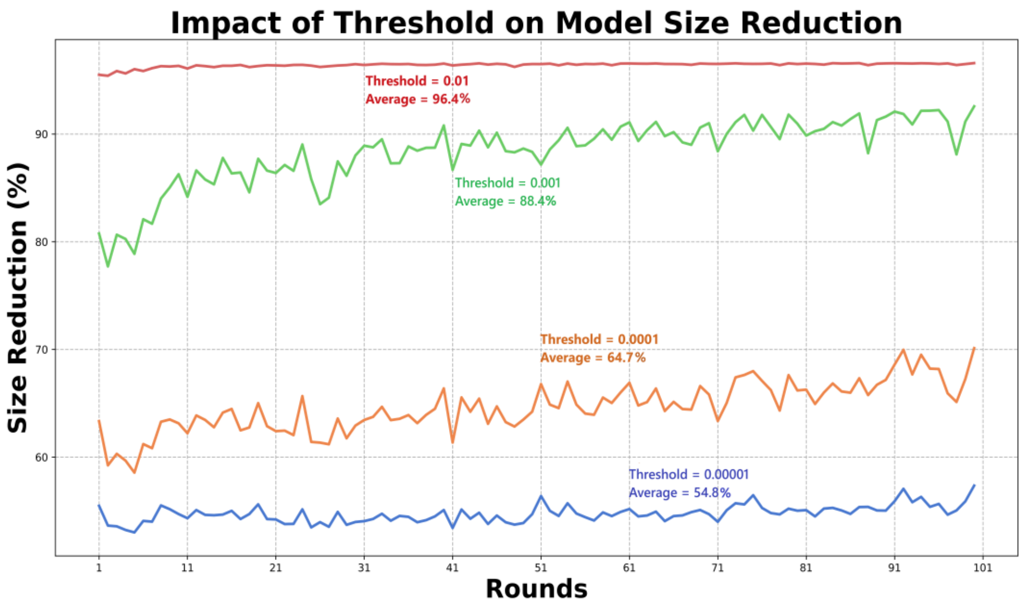

Figure 4: The percentage results of the size reduction of the proposed approach using different thresholds across FL rounds on MNIST dataset.

Table 3: Accuracy and size reduction (%, remaining KB) of the proposed method on CIFAR-10 across pruning thresholds.

Thresholds	Size reduction (%), Size (KB)	Accuracy (%)	# Rounds
0.01	95.6, 59.84	44.77	50
0.005	92, 108.8	52.5 44.9	29 50
0.003	85.1, 202.64	51.7 48.3	26 50
0.002	77, 312.8	53.3 48	44 50
0.001	64, 489.6	54.8	29
0.0005	56.8, 587.52	53.7	45
0.0001	51.4, 661.96	55.4	50
Base	0, 1360.0	54.73	50

size reduction but gradually decreases accuracy. At τ = 0.001, our method achieves an 88.4% reduction in communication with test accuracy of 96.5%, only 0.2% below the baseline (96.7%), demonstrating that exponent encoding preserves model fidelity while significantly reducing transmission. At higher thresholds (e.g., τ = 0.01), the reduction reaches over 96%, but accuracy falls by ~8%.

Convergence speed analysis further supports the validity of the approach. For τ = 0.001, the model reached baseline accuracy (96.7%) in 26 rounds, compared to 30 rounds for the Base model, indicating faster convergence despite reduced communication. Even at τ = 0.005, the method converged within 28 rounds, albeit at slightly reduced final accuracy. These results confirm that, for MNIST, exponent encoding achieves both high compression and accelerated convergence when the threshold is chosen conservatively.

To illustrate the contribution of each stage in the proposed compression pipeline, three configurations were examined:

i. float32 → float16 downcasting only,
ii. downcasting + threshold pruning, and
iii. the complete exponent-encoding method.

The first configuration directly halves the communication cost, each parameter requires 2 bytes instead of 4, while introducing a small accuracy reduction of about 2 % due to limited float16 precision.

The second configuration adds threshold pruning, which achieves the compression rates reported in Table 2; however, this intermediate representation is not aggregated by the server and therefore no accuracy value is associated with it. It serves to isolate the effect of pruning on communication volume.

The third configuration applies the proposed exponent-encoding to the pruned float16 updates. Because this encoding reuses the exponent field of float16 without adding new data, its compression rate remains identical to that of stage (ii). Yet, when aggregated in FedAvg, this version not only restores but slightly improves model accuracy compared with the pure float16 baseline, demonstrating that the encoding step stabilizes rounding effects while preserving significant information.

5.2 Results using Cifar-10

Our experiments on the CIFAR-10 dataset demonstrate that selective parameter pruning in FL can substantially reduce communication cost while retaining, and in some cases improving, model accuracy. The results highlight

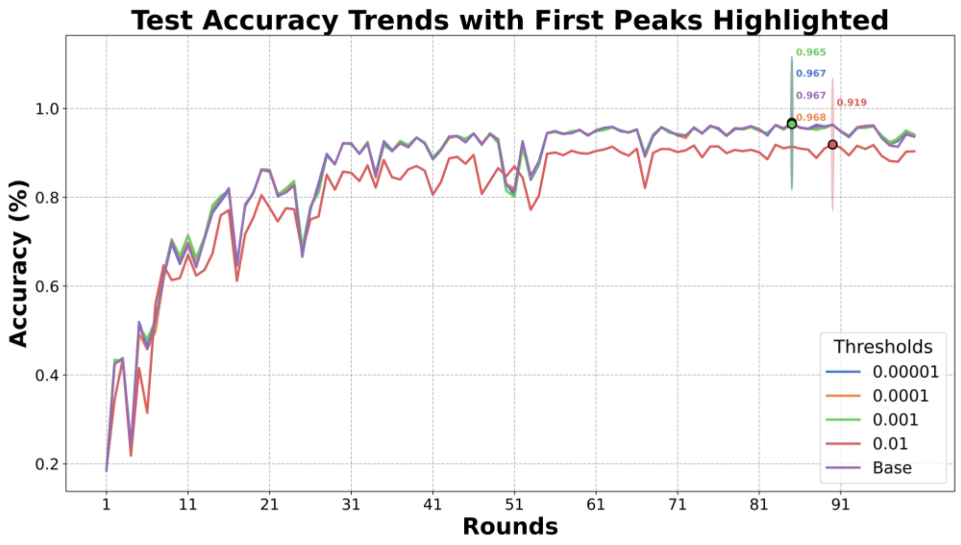

Figure 5: Test accuracy results across FL rounds for different used thresholds on CIFAR-10 dataset.

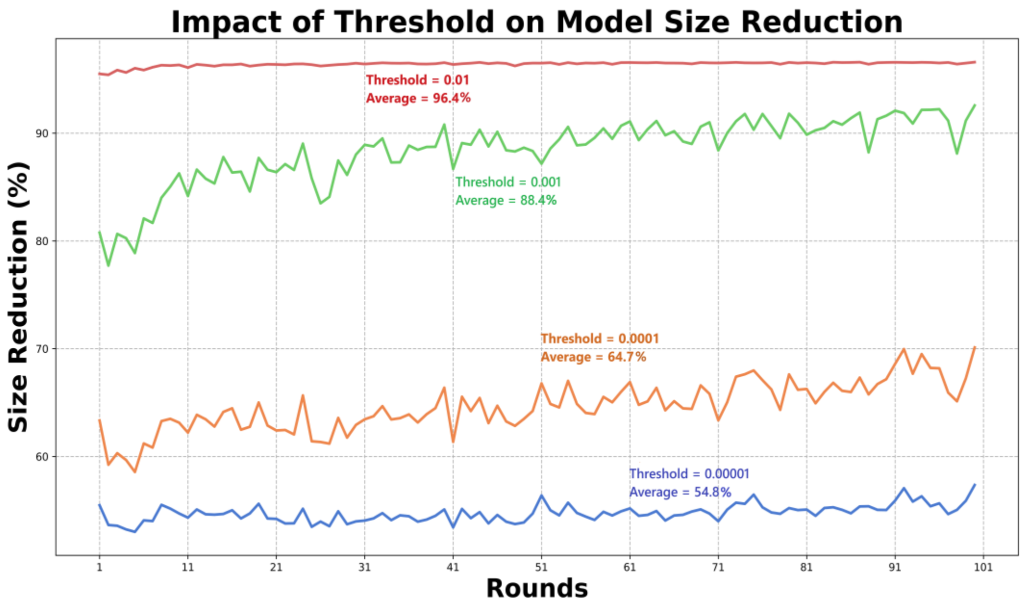

Figure 6: The percentage results of the size reduction of the proposed approach using different thresholds across FL rounds on CIFAR-10 dataset.

critical trade-offs between pruning thresholds, training rounds, and performance outcomes. The key findings based on Table 3 and Figures 5 and 6 are summarized below.

The same three-stage comparison was performed for CIFAR-10. The float32 → float16 downcasting reduced communication size by approximately 50 %, with an accuracy drop of about 2 % relative to the full-precision baseline.

Adding threshold pruning further reduced the transmitted size (as shown in Table 3) but was not used for aggregation, since this intermediate output only quantifies communication savings.

When the exponent-encoding step was introduced, reusing the exponent bits of float16 values to embed run-length information, the compression rate remained the same as that of stage (ii) but the aggregated model regained and slightly exceeded the float16 accuracy while maintaining overall communication savings of about 60–70 %.

This confirms that the proposed encoding is responsible for recovering learning performance without any additional transmission cost.

5.2.1 Optimal Threshold (τ = 0.001) efficiency and early convergence

The threshold τ = 0.001 provided the best balance between compression and accuracy. It achieved a 64% reduction in communication size while matching and even slightly surpassing the baseline model's accuracy. By round 29, the model at τ = 0.001 reached 54.8% accuracy, compared to the baseline's 54.7% at round 50, representing a 42% reduction in required rounds (29 vs. 50) with only 36% of the original transmission volume. This demonstrates that exponent encoding with τ = 0.001 accelerates convergence while maintaining fidelity, effectively filtering out parameters that do not contribute meaningfully to learning.

Interestingly, a milder threshold τ = 0.0001 also showed notable benefits. It yielded a 51.4% size reduction and achieved 55.4% accuracy at round 50, a +0.7% absolute improvement over the baseline. This suggests that pruning negligible weights below 0.0001 not only compresses the model but also acts as a noise filter, improving gradient clarity without disrupting essential features. These findings confirm that moderate pruning thresholds can simultaneously enhance accuracy and efficiency, demonstrating the adaptability of the proposed method.

5.2.2 Pruning intensity trade-offs: accuracy vs. model size

Aggressive pruning thresholds (τ = 0.005–0.01) delivered 77–95.6% reduction in model size, with acceptable early training accuracy (51–53% by round 29). However, by round 50 the accuracy declined to 44–48%, approximately 6–10% lower than baseline. This degradation arises because pruning at such intensity removes weights that appear unimportant in early learning but become essential for modeling complex features later in training.

In contrast, more conservative pruning (τ = 0.0005–0.0001) offered a better trade-off. These thresholds achieved 51–57% size reduction while preserving or slightly improving accuracy (final accuracy 53.7–55.4% vs. baseline 54.7%). This confirms that retaining the majority of weights while discarding only the least significant parameters ensures stability and performance.

Overall, the results demonstrate that τ = 0.001 is optimal for maximizing efficiency with early convergence, while τ = 0.0001 ensures long-term accuracy improvement. This suggests that a dynamic threshold schedule—using τ = 0.001 in early rounds for rapid progress, then τ = 0.0001 in later rounds to stabilize accuracy—could provide the best overall outcome. Future research should explore adaptive thresholds across training stages and assess their robustness under non-IID FL settings to further enhance scalability.

5.3 Impact of dataset complexity on compression performance: MNIST vs. CIFAR-10

The performance of the proposed compression technique varies significantly between the MNIST and CIFAR-10 datasets, primarily due to differences in dataset complexity and the resulting parameter sparsity, as well as the implications of model architecture.

5.3.1 Dataset complexity and parameter sparsity

MNIST consists of grayscale images (1 channel, 28×28 pixels) that are inherently simpler and often contain large homogeneous regions, such as backgrounds, which contribute to many near-zero values in the input. Consequently, after convolutional processing and training, the resulting parameter updates tend to be sparse, with a substantial proportion of values falling below the compression threshold (e.g., 0.0001). This high level of sparsity enables the proposed method to achieve very high compression ratios—often resulting in a 70–90% reduction in model size. In contrast, CIFAR-10 comprises RGB images (3 channels, 32×32 pixels) that are more complex, with rich color and texture information. The increased complexity necessitates capturing intricate patterns, leading to fewer near-zero values in the parameter updates. As a result, the compression ratio is lower on CIFAR-10, typically yielding a reduction in the range of 20–50%.

5.3.2 Model architecture

Both datasets employ a similar network architecture featuring three convolutional layers followed by fully connected layers. However, the initial layer for MNIST processes a single channel, which results in inherently less data being passed through the network. This limitation in data complexity leads to sparser parameter updates in the early convolutional layers. Conversely, the CIFAR-10 model, with its three-channel input, processes a higher-dimensional feature space, resulting in denser and more substantial parameter updates across the layers. Although the shared structure of the network (e.g., filter sizes and fully connected layers) remains constant, the increased input dimensionality in CIFAR-10 propagates through the network, yielding less compressible updates. In summary, the superior compression performance observed on the MNIST dataset is attributed to its simpler, grayscale nature and the resulting sparsity in parameter updates, while the greater complexity of CIFAR-10's RGB images leads to denser parameters and a lower achievable compression ratio.

5.4 Comparative discussion with state-of-the-art compression techniques

To better position the effectiveness of the proposed exponent-encoded compression approach, we contrast its behavior against three widely adopted communication-efficient methods in federated learning: Top-K gradient sparsification with error feedback (DGC), QSGD quantization, and PowerSGD low-rank approximation. Each comparison considers the scenario of image classification tasks such as CIFAR-10 or MNIST, under non-IID data partitions and typical round settings (e.g., 100–200 communication rounds, 10–100 clients), which match the setup used in our experiments.

Top-K Sparsification with Error Feedback. This technique retains only a small fraction (e.g., top 0.1–1%) of the most significant gradient updates in each round, transmitting a sparse vector alongside an accumulated residual buffer to compensate for dropped entries. While it achieves extremely high compression ratios—up to 600× on large models like ResNet-50—this method introduces additional memory and compute overhead on each client, stemming from the need to maintain and update the error feedback buffer and to compute the top-K selection. In practice, it also requires careful tuning of the sparsity level and warm-up strategies to avoid convergence instability. Compared to this, our method achieves moderate compression (4×–8×) but with no memory overhead and minimal extra computation—limited to simple thresholding and bit manipulation—while preserving accuracy without special heuristics. Under similar FL conditions (non-IID CIFAR-10 with 100 clients), both methods maintain high accuracy, but our method is notably simpler to integrate and incurs far less client-side burden.

QSGD Quantization. QSGD compresses model updates by stochastically quantizing each gradient coordinate into a small number of bits, often as low as 2–4 bits. It provides unbiased estimators and has been shown to preserve model accuracy across various datasets, including CIFAR-10 and ImageNet, with compression ratios between 4× and 6.8×. However, QSGD introduces slight extra variance in training and typically requires multiple SGD iterations to converge to the same accuracy as full-precision training. Moreover, random quantization and encoding procedures introduce mild computational overhead per communication round. In comparison, our exponent-encoded method achieves similar compression without introducing randomness or increasing the number of training rounds, and its decoding logic is deterministic and hardware-friendly. As such, our approach provides a more lightweight and deterministic alternative with comparable accuracy performance in practice.

PowerSGD. This method compresses updates—particularly large weight matrices—by approximating them as low-rank products using power iteration. It is particularly effective for dense layer gradients (e.g., in LSTMs or deep CNNs), often yielding compression factors over 100× with negligible or controlled accuracy loss. However, PowerSGD requires matrix sketching and orthogonalization steps on clients, increasing computational complexity significantly, especially when using larger rank settings to preserve accuracy. Moreover, it typically benefits from tuning the rank and applying momentum correction or warm-starting strategies. In contrast, our method offers fixed-cost encoding that does not depend on model architecture or gradient shape. For federated training with lightweight models (e.g., CNNs on MNIST or CIFAR-10), our approach provides a favorable trade-off by achieving solid compression (up to 8×) with minimal runtime overhead and without requiring any additional hyper-parameters.

In summary, while each of these methods offers strong compression capabilities, the proposed exponent-based encoding strikes a compelling balance between compression, accuracy, and computational simplicity. It avoids the memory overhead and tuning demands of Top-K and PowerSGD and circumvents the convergence variance of QSGD, all while achieving meaningful bandwidth savings and preserving accuracy in realistic FL settings.

6 Discussion

The evaluation results presented in Section 5 show that the proposed exponent-encoded compression achieves competitive or superior communication savings compared with state-of-the-art (SOTA) methods summarized in Table 1. For instance, ResFed [23] reported extreme per-round compression (>700× on CIFAR-10) but relied on residual coding and lossy transformations, whereas our method achieves a 64.7% reduction on CIFAR-10 without auxiliary structures and with accuracy slightly higher than baseline. Similarly, FLCP [24] obtained 87–89% ciphertext reduction by combining adaptive weight compression with homomorphic encryption and differential privacy, but introduced cryptographic overhead. In contrast, our approach delivers 88.4% reduction on MNIST with only a −0.2% accuracy change and 64.7% reduction on CIFAR-10 with a +0.66% accuracy gain, while requiring no additional metadata or encryption. These comparisons indicate that the performance gains stem primarily from the exponent encoding mechanism, which embeds run-length counts directly into float16 exponents, rather than from aggressive pruning. Pruning is applied only through a lightweight threshold to mark negligible updates, while the compression benefit arises from the efficient reuse of the exponent field.

An important distinction between datasets is also observed. MNIST, being grayscale and structurally simpler, produces highly sparse updates with many near-zero parameters, allowing compression ratios above 80%. By contrast, CIFAR-10, with its RGB inputs and richer feature space, generates denser updates with fewer negligible values, limiting achievable compression to ~50–65%. Additionally, the deeper convolutional processing required for CIFAR-10 propagates denser activations through the network, reducing sparsity in later layers. These findings suggest that data complexity and model architecture strongly influence the effectiveness of exponent encoding, but the method adapts gracefully to both scenarios while preserving convergence. Overall, the discussion underscores that our technique closes a gap in FL compression research: it provides metadata-free, accurate, and scalable reduction, competitive with SOTA methods while avoiding their reliance on residual, lossy, or cryptographic components.

An additional consideration is the numerical stability of the proposed encoding scheme. Since run-length counts are embedded only into float16 exponents of negligible updates (i.e., parameters below τ), significant gradients remain unaffected. The representable dynamic range of float16 is preserved, and the added counts do not approach overflow limits in any of the evaluated settings. Across all experiments, no divergence or instability was observed. The accuracy and reduction curves presented in Figures 3–6 show smooth convergence behavior across all thresholds, further confirming that the encoding does not introduce overflow risks or instability in practice.

Furthermore, although this study primarily focuses on communication efficiency and model fidelity, it is also important to consider potential security and privacy implications. The proposed exponent-encoding mechanism operates entirely within the numerical domain of existing floating-point parameters and does not introduce auxiliary metadata or side channels external to model updates. All encoded information represents only run-length counts of negligible parameters and is embedded inside the exponent bits of locally computed updates before secure transmission. Consequently, the encoded values do not convey any client-specific data or gradient semantics beyond what is already present in conventional parameter magnitudes. When combined with standard secure aggregation or differential-privacy noise injection, the compression step remains orthogonal and compatible: aggregation operates on reconstructed float32 updates after decompression, while differential-privacy perturbations can be applied either before or after compression without affecting the statistical guarantees. Because exponent modification affects only negligible weights ($|w| < \tau$), the risk of adversarial reversibility or structural leakage is minimal. Nevertheless, future work may investigate formal privacy proofs under adversarial inference models to further ensure robustness in privacy-critical deployments.

In contrast to alternative compression paradigms such as entropy coding (e.g., Huffman or arithmetic encoding) and vector quantization, the proposed exponent-encoded scheme follows a distinct objective. Entropy-based approaches aim to reduce bit-level redundancy after quantization and often require maintaining codebooks or frequency tables, which can introduce additional computational overhead during encoding and decoding. Vector quantization methods, on the other hand, cluster similar parameter values into shared representatives, yielding high compression at the cost of additional reconstruction error and lookup complexity. In comparison, our approach leverages the existing floating-point exponent field to embed run-length information of negligible updates without any external dictionaries or probability models. This allows near-zero reconstruction loss, minimal computational cost, and compatibility with standard IEEE 754 operations. Thus, the proposed method can be viewed as orthogonal to entropy- or codebook-based methods—prioritizing communication simplicity and low processing latency over maximal compression ratio.

7 Limitations and future work

The proposed exponent-encoding approach achieves notable communication savings; however, its efficiency decreases when model updates are densely distributed or contain few negligible parameters. In particular, when a client's local update diverges significantly from the recent global model, the resulting gradients contain fewer near-zero values, leading to a lower compression ratio. These cases do not affect convergence or accuracy but reduce compression effectiveness. Future work will investigate adaptive thresholding and hybrid encoding strategies that can dynamically adjust to parameter sparsity and update magnitude.

8 Conclusion

This paper presents an innovative communication-efficient approach for FL that uniquely exploits the exponent field of float16 representations to dramatically reduce transmission overhead while preserving model accuracy. Our technique encodes sequences of negligible parameter updates directly into the exponent bits of significant values, achieving: (1) 88.4% size reduction on MNIST with only 0.2% accuracy degradation, demonstrating exceptional efficiency for sparse parameter distributions; and (2) 51.4-64% compression on CIFAR-10 while maintaining or even improving accuracy (up to +1.2%), proving effective for complex, dense update patterns. The exponent-based compression mechanism provides three key advantages over conventional methods: precise preservation of critical weights through threshold-based filtering, implicit positional encoding via exponent manipulation, and computational efficiency suitable for edge devices. Comparative evaluations confirm our method's superiority to existing sparsification and quantization techniques, particularly in non-IID settings, while maintaining full compatibility with standard FL frameworks. Future work will investigate dynamic threshold adaptation and hybrid compression schemes to further optimize the accuracy-efficiency trade-off across diverse application domains, from IoT to healthcare systems.

Acknowledgements

We sincerely thank the University of Human Development, Kurdistan Region, Iraq, for providing access to a high-performance computer server, which facilitated the implementation and testing of the proposed approaches.

Author contributions

Goran contributed to the conceptualization, literature review, implementation, validation, formal analysis, and writing of the original draft. Dr. Hozan provided supervision, and contributed to the writing through review and editing.

References

[1] J. Ayeelyan, S. Utomo, A. Rouniyar, H.-C. Hsu, and P.-A. Hsiung, "Federated learning design and functional models: survey," *Artificial Intelligence Review,* vol. 58, no. 1, p. 21, 2024/11/16 2024, doi: 10.1007/s10462-024-10969-y.

[2] E. M. El Houby, "A Review of Machine Learning Techniques in the Medical Domain," *Informatica,* vol. 49, no. 16, 2025.

[3] G. S. Nariman and H. K. Hamarashid, "Hierarchical federated learning for health trend prediction and anomaly detection using pharmacy data: from zone to national scale," *International Journal of Data Science and Analytics,* 2025/03/24 2025, doi: 10.1007/s41060-025-00756-5.

[4] B. Mohammed, "A Comprehensive Overview of Federated Learning for NextGeneration Smart Agriculture: Current Trends, Challenges, and Future Directions," *Informatica,* vol. 49, no. 1, 2025.

[5] N. Azeri, O. Hioual, and O. Hioual, "Efficient Vanilla Split Learning for Privacy-Preserving Collaboration in Resource-Constrained Cyber-Physical Systems," *Informatica,* vol. 48, no. 11, 2024.

[6] L. Xiao, H. Shan, J. Zhu, R. Mao, and S. Pan, "FD3QN: A Federated Deep Reinforcement Learning Approach for Cross-Domain Resource Cooperative Scheduling in Hybrid Cloud Architecture," *Informatica,* vol. 49, no. 10, 2025.

[7] M. Harasic, F.-S. Keese, D. Mattern, and A. Paschke, "Recent advances and future challenges in federated recommender systems," *International Journal of Data Science and Analytics,* vol. 17, no. 4, pp. 337-357, 2024/05/01 2024, doi: 10.1007/s41060-023-00442-4.

[8] I. Varlamis *et al.*, "Using big data and federated learning for generating energy efficiency recommendations," *International Journal of Data Science and Analytics,* vol. 16, no. 3, pp. 353-369, 2023/09/01 2023, doi: 10.1007/s41060-022-00331-2.

[9] I. Ullah, X. Deng, X. Pei, H. Mushtaq, and M. Uzair, "IoV-SFL: A blockchain-based federated learning framework for secure and efficient data sharing in the internet of vehicles," *Peer-to-Peer Networking and Applications,* vol. 18, no. 1, p. 34, 2024/12/06 2024, doi: 10.1007/s12083-024-01821-9.

[10] M. K. Pulligilla and C. Vanmathi, "A traffic flow prediction framework based on integrated federated learning and Recurrent Long short-term networks," *Peer-to-Peer Networking and Applications,* vol. 17, no. 6, pp. 4131-4155, 2024/11/01 2024, doi: 10.1007/s12083-024-01792-x.

[11] D. S. B. Naik and V. Dondeti, "Trust-based secure federated learning framework to mitigate internal attacks for intelligent vehicular networks," *Peer-to-Peer Networking and Applications,* vol. 18, no. 2, p. 10, 2025/01/03 2025, doi: 10.1007/s12083-024-01835-3.

[12] B. Li, Y. Jiang, Q. Pei, T. Li, L. Liu, and R. Lu, "FEEL: Federated End-to-End Learning With Non-IID Data for Vehicular Ad Hoc Networks," *IEEE Transactions on Intelligent Transportation Systems,* vol. 23, no. 9, pp. 16728-16740, 2022, doi: 10.1109/TITS.2022.3190294.

[13] H. Zhang, Z. Li, S. Xi, X. Zhao, J. Liu, and P. Zhang, "Heterogeneity-aware device selection for clustered federated learning in IoT," *Peer-to-Peer Networking and Applications,* vol. 18, no. 1, p. 29, 2024/12/04 2024, doi: 10.1007/s12083-024-01869-7.

[14] G. S. Nariman and H. K. Hamarashid, "Communication overhead reduction in federated learning: a review," *International Journal of Data Science and Analytics,* 2024/12/04 2024, doi: 10.1007/s41060-024-00691-x.

[15] B. Li, Y. Shi, Q. Kong, Q. Du, and R. Lu, "Incentive-Based Federated Learning for Digital-Twin-Driven Industrial Mobile Crowdsensing," *IEEE Internet of Things Journal,* vol. 10, no. 20, pp. 17851-17864, 2023, doi: 10.1109/JIOT.2023.3279657.

[16] J. Hao, L. Zhang, and Y. Zhao, "Grouped federated learning for time-sensitive tasks in industrial IoTs," *Peer-to-Peer Networking and Applications,* vol. 17, no. 2, pp. 819-833, 2024/03/01 2024, doi: 10.1007/s12083-023-01616-4.

[17] Z. Wang, M. Wen, Y. Xu, Y. Zhou, J. H. Wang, and L. Zhang, "Communication compression techniques in distributed deep learning: A survey," *Journal of Systems Architecture,* p. 102927, 2023.

[18] G. S. Nariman and H. K. Hamarashid, "Analyzing and Classifying Data Format Strategies for Efficient Communication in Federated Learning," *Passer Journal of Basic and Applied Sciences,* vol. 7, no. 1, pp. 385-398, 2025.

[19] W. Dai, J. Fan, Y. Miao, and K. Hwang, "Deep Learning Model Compression With Rank Reduction in Tensor Decomposition," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 1315-1328, 2025, doi: 10.1109/TNNLS.2023.3330542.

[20] F. M. A. Khan, H. Abou-Zeid, and S. A. Hassan, "Model Pruning for Efficient Over-the-Air Federated Learning in Tactical Networks," in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, 28 May-1 June 2023 2023, pp. 1806-1811, doi: 10.1109/ICCWorkshops57953.2023.10283773.

[21] W. Dai, J. Fan, Y. Miao, and K. Hwang, "Deep Learning Model Compression With Rank Reduction in Tensor Decomposition," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[22] Y. Jiang *et al.*, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[23] Y. Mao *et al.*, "Communication-efficient federated learning with adaptive quantization," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 4, pp. 1-26, 2022.

[24] C. Chen *et al.*, "Communication-efficient federated learning with adaptive parameter freezing," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021: IEEE, pp. 1-11.

[25] R. Song, L. Zhou, L. Lyu, A. Festag, and A. Knoll, "Resfed: Communication-efficient federated learning with deep compressed residuals," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 9458-9472, 2023.

[26] W. Yang, Y. Yang, Y. Xi, H. Zhang, and W. Xiang, "FLCP: federated learning framework with communication-efficient and privacy-preserving," *Applied Intelligence*, vol. 54, no. 9, pp. 6816-6835, 2024.

[27] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," in *2022 IEEE 38th international conference on data engineering (ICDE)*, 2022: IEEE, pp. 965-978.

[28] P. Riedel, L. Schick, R. von Schwerin, M. Reichert, D. Schaudt, and A. Hafner, "Comparative analysis of open-source federated learning frameworks - a literature-based survey and review," *International Journal of Machine Learning and Cybernetics*, vol. 15, no. 11, pp. 5257-5278, 2024/11/01 2024, doi: 10.1007/s13042-024-02234-z.