RESFED: COMMUNICATION EFFICIENT FEDERATED LEARNING BY TRANSMITTING DEEP COMPRESSED RESIDUALS

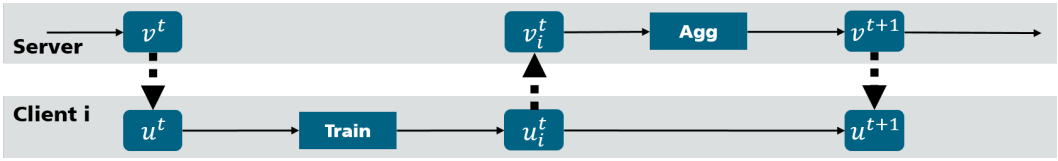
Anonymous authors
Paper under double-blind review

ABSTRACT

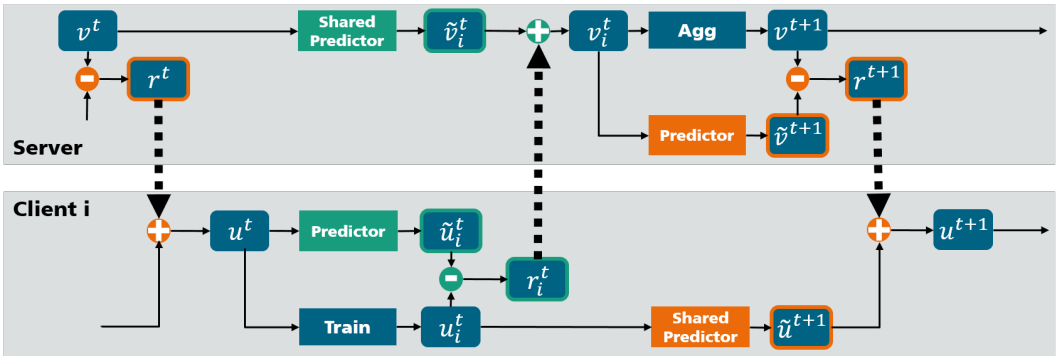
Federated learning enables cooperative training among massively distributed clients by sharing their learned local model parameters. However, with increasing model size, deploying federated learning requires a large communication bandwidth, which limits its deployment in wireless networks. To address this bottleneck, we introduce a residual-based federated learning framework (ResFed), where residuals rather than model parameters are transmitted in communication networks for training. In particular, we integrate two pairs of shared predictors for the model prediction in both server-to-client and client-to-server communication. By employing a common prediction rule, both locally and globally updated models are always fully recoverable in clients and the server. We highlight that the residuals only indicate the quasi-update of a model in a single inter-round, and hence contain more dense information and have a lower entropy than the model, comparing to model weights and gradients. Based on this property, we further conduct lossy compression of the residuals by sparsification and quantization and encode them for efficient communication. The experimental evaluation shows that our ResFed needs remarkably less communication costs and achieves better accuracy by leveraging less sensitive residuals, compared to standard federated learning. For instance, to train a 4.08 MB CNN model on CIFAR-10 with 10 clients under non-independent and identically distributed (Non-IID) setting, our approach achieves a compression ratio over 700$\times$ in each communication round with minimum impact on the accuracy. To reach an accuracy of 70%, it saves around 99% of the total communication volume from 587.61 Mb to 6.79 Mb in up-streaming and to 4.61 Mb in down-streaming on average for all clients.

1 INTRODUCTION

Federated learning has become an emerged machine learning paradigm, which enables distributed training on broad data sources without disclosing their original data McMahan et al. (2017). Instead of transmitting raw data, only parameters (mostly model weights or gradients) in federated learning are iteratively shared between clients and a server via heterogeneous networks. Federated learning has been successfully applied to various applications Lyu et al. (2020), such as mobile keyboard prediction, speech recognition, image object detection, etc, However, with the increasing size of machine learning models, the existing mobile communication infrastructure cannot always meet the requirement in terms of bandwidth and latency in federated learning, which constraints the wide deployment of federated learning. For instance, to train a transformer model with billions of parameters usually 32-bit float parameters), the size of a message in a single federated learning round can be several 10 or 100 Gigabytes, e.g. a CTRL model (Keskar et al. (2019)) with 1.6 billions parameters or a T5 model (Raffel et al. (2020)) with up to 11 billions parameters. That can cause an enormous and extremely costly data traffic, even in 5G NR networks, where the throughput can be from 5 Gbps to 18 Gbps. Another application scenario is to improve machine learning models for road traffic object recognition and detection in V2X (Vehicle-to-Everything) communication networks, where the bandwidth for V2X is also occupied for other traffic services at the same time, e.g. collective perception service, and obviously the safety-related services should have higher pri-



(a) Standard federated learning system



(b) Residual-based federated learning system

Figure 1: Paradigm shift from standard federated learning to residual-based federated learning system, with additional two pairs of predictors and corresponding operators (in green and orange). The model is updated in the client i by local training (Train) and in the server by aggregation (Agg).

ority. Therefore, communication efficiency is a pivotal component for deploying federated learning, especially in wireless networks.

In an attempt to tackle the communication bottleneck, the parameter compression is considered as one of the most effective approaches, which allows for updating the models by transmitting much smaller size of messages in networks, and thereby reduces the required time per communication round in federated learning. The approaches proposed by Xu et al. (2022); Reisizadeh et al. (2020); Hönig et al. (2022) can effectively reduce the communication volume in each round by various quantization techniques, however they only consider the communication efficiency for uploading (client-to-server) but not for downloading (server-to-client). Lin et al. (2018) compress the gradients instead of model parameters for distributed learning, which can not well fit federated learning, where clients can train multiple epochs in each round. Wu et al. (2022) use knowledge distillation (Hinton et al. (2015)) to learn and transmit a smaller model, where the original model structure is affected. Furthermore, all of those works attempt to compress the model parameters or gradients based on the model in a specific round, without consideration of inter-round model update similarity, which contains additional redundancy sequentially.

Inspired by residuals in video compression protocols from Li et al. (2021), we introduce a residual-based federated learning framework, termed as *ResFed*. It allows the server and clients to share and update models by sharing model parameter residuals rather than model parameters or gradients. Particularly, by observing training trajectory in each local client and the aggregation trajectory in the server, we believe model updates in both clients and the server can be predictable. Those predictive models – in analogy with predictive frames in video transmission – can foresee model updates in the federated learning. After each communication round, we use the deviations between the predictive and the actual updated model parameters, which we call the model residuals, for the communication in networks. Note that the actual updated models can be always recoverable by acquiring the residuals, as the predictors in the senders are shared to the receivers in *ResFed*. More details are provided in Sec. 2 and 3.

Unlike transmitting model weights, *ResFed* can wring out the potential redundancy by removing the predictive information from history updates and only keep the residuals for communication. Compared to transmitting model gradients after each training epoch, *ResFed* allows the models to be trained locally multiple times. Compared to transmitting residual accumulation for multiple epochs, *ResFed* further minimizes the information by predicting the model updates from history. As shown

in Fig.C.2, the values of residuals are overall smaller than weights and gradients during the entire training process. To further shrink the size of messages for communication, we then compress only residuals using sparisification and quantization, and encode the messages for information sharing in client-to-server and server-to-client.

Our main contributions are summarized as follows:

- We introduce and formulate the model residuals for the communication efficiency in federated learning and indicate the residuals contain more dense information than model weights and gradients.
- We propose a novel federated learning framework (*ResFed*) based on deep residual compression, which consists of the following steps: predictor sharing, model prediction, residual generation, residual compression, residual communicating, model recovering and model trajectory synchronization.
- We provide the experimental evaluation of our framework with various communication cost budgets in both up- and down-streaming, which gives an insight in deploying it in resource-constrained communication environments. The open source implementation of *ResFed* will be publicly available.

2 System Setup

We first introduce the related concepts and techniques that will be used in our framework. Given N clients and a server in a federated learning system, we only focus on the information sharing between one single client i and the server from communication round t to $t+1$, as shown in Fig 1. The information sharing for other clients is the same.

Notation. Throughout this paper, we use w to denote model weights, r to denote model residuals. To distinguish the parameters, we use u and v to denote the model weights in the server and clients respectively. For residuals, we use $r^t_{i,ul}$ and $r^t_{i,dl}$ to denote the residuals of the client i for uploading and the residuals of the server for downloading respectively. More details on notation are shown in Tab. A1.

2.1 Model Update

Client. Given a client i with a local dataset $\mathscr{D}_i$, the initial local model in a new round t is u_i^{t-1}. Before the local training starts, the client initially receives the global model v^t from the server and updates the local model to $\hat{u}_i^t$. After that, local model $\hat{u}_i^t$ is trained on $\mathscr{D}_i$ and transited to u_i^t. We mark the first update as $u_i^{t-1} \rightarrow \hat{u}_i^t$ and the second one as $\hat{u}_i^t \rightarrow u_i^t$. Note that the first updated model is equal or similar to the global updated model v^t, i.e. $\hat{u}_i^t \simeq u^t$. If lossy compression is used for communication and the loss due to compression can not be repaired, then $\hat{u}_i^t \sim u^t$.

Server. Similarly, the global model v_{t-1} in the server is also updated twice after one round of communication t. The first update happens when it receives models from the clients, i.e. $v^{t-1} \rightarrow \{\hat{v}_i^t | i = 1,2,...,N\}$. Then the aggregation leads to the second update, $\{\hat{v}_i^t | i = 1,2,...,N\} \rightarrow v^t$.

2.2 Model Trajectory

Client. Given a client i at time point t, we cache the updated models with a sliding time window $[t-T,t]$ in two different queues, that distinguish by two model updates. We refer the time sequence of local model updates $\mathscr{L}_i^t = \{u_i^{t-T},...,u_i^t\}$ from $u_i^t \rightarrow \hat{u}_i^t$ as a local model trajectory, and $\hat{\mathscr{G}}_i^t = \{\hat{u}_i^{t-T},...,\hat{u}_i^t\}$ from $\hat{u}_i^t \rightarrow u_i^{t+1}$ as a global model trajectory.

Server. Correspondingly, we cache the local and global model updates in the local and global model trajectories for all client at the server, i.e. $\{\hat{\mathscr{L}}_i | i = 1,2,...,N\}$ and $\{\mathscr{G}_i | i = 1,2,...,N\}$. Note that if the server can always send the lossless global model update to all clients, the global trajectories at time t are the same for all clients.

2.3 MODEL PREDICTION

Client. Given a client i at time point t, we predict $\hat{u}_i^t \rightarrow u_i^t$ from the local and global training trajectories, $\mathscr{L}_i^{t-1}$ and $\hat{\mathscr{G}}_i^{t-1}$ as follows:

$$\tilde{u}_i^t = f_{predict,i}(\mathscr{L}_i^{t-1}, \hat{\mathscr{G}}_i^{t-1}, \hat{u}_i^t) = \arg\max_{u_i^t} p(u_i^t | \underbrace{u_i^{t-T}, ..., u_i^{t-1}}_{\mathscr{L}_i^{t-1}}, \underbrace{\hat{u}_i^{t-T}, ..., \hat{u}_i^{t-1}}_{\hat{\mathscr{G}}_i^{t-1}}, \hat{u}_i^t) \tag{1}$$

where $f_{predict,i}$ is the used predictor for model prediction in the client i.

Server. For the server, we predict model updates $\hat{v}_i^t \rightarrow v^t$ for each client i from local and global trajectories $\hat{\mathscr{L}}_i^{t-1}$ and $\mathscr{G}_i^{t-1}$ as follows:

$$\tilde{v}_i^t = h_{predict,i}(\hat{\mathscr{L}}_i^{t-1}, \mathscr{G}_i^{t-1}, \hat{v}_i^t) = \arg\max_{v_i^t} p(v_i^t | \underbrace{v_i^{t-T}, ..., v_i^{t-1}}_{\hat{\mathscr{L}}_i^{t-1}}, \underbrace{\hat{v}_i^{t-T}, ..., \hat{v}_i^{t-1}}_{\mathscr{G}_i^{t-1}}, \hat{v}_i^t), \forall i \in 1, ..., N \tag{2}$$

where $h_{predict,i}$ is the used predictor for model prediction in the server for each client i.

2.4 MODEL RESIDUAL

Given a model update $v^{t-1} \rightarrow \hat{v}_i^t$ for the client i at time t in the server or $\hat{u}_i^t \rightarrow u_i^t$ in the client i, if we can compute the model prediction $\tilde{v}_i^t$ or $\tilde{u}_i^t$ based on Eq. 1, we define the model residual as follows:

$$r_{i,ul}^t = u_i^t - \tilde{u}_i^t \quad or \quad r_{i,dl}^t = v_i^t - \tilde{v}_i^t \tag{3}$$

where $r_{i,ul}^t$ is the residuals from the client i for uploading, and $r_{i,dl}^t$ is the residuals from the server for downloading, respectively. More understanding of residuals is provided in Sec. C.

2.5 DEEP COMPRESSION

To reduce the model size for more efficient communication, we shrink the model size before sending it out. We define a compressed model in the client i:

$$\vec{u}_i^t = f_{compress}(u_i^t) \tag{4}$$

and in the server:

$$\vec{v}_i^t = h_{compress}(v_i^t), \forall i \in \{1, ..., N\} \tag{5}$$

where $f_{compress}$ and $h_{compress}$ is the used compressor for model compression in clients and server respectively. In our system, we consider to compress and communicate model residuals instead of model itself in the client i:

$$\vec{r}_{i,ul}^t = f_{compress}(r_{i,ul}^t) = f_{compress}(u_i^t - f_{predict,i}(u_i^{t-T}, ..., u_i^{t-1}, \hat{u}_i^{t-T}, ..., \hat{u}_i^{t-1}, \hat{u}_i^t)) \tag{6}$$

and in the server:

$$\vec{r}_{i,dl}^t = h_{compress}(r_{i,dl}^t) = h_{compress}(v_i^t - f_{predict,i}(v_i^{t-T}, ..., v_i^{t-1}, \hat{v}_i^{t-T}, ..., \hat{v}_i^{t-1}, \hat{v}_i^t)), \forall i \in \{1, ..., N\} \tag{7}$$

We provide details on our employed lossy compression scheme consisting of sparsification, quantization and encoding in Sec. B.

3 RESFED: RESIDUAL-BASED FEDERATED LEARNING FRAMEWORK

The overview of the *ResFed* is shown in Fig. 1 and the detailed steps with lossy compression in one communication round is given in Fig. 2. In particular, we introduce *(a)* predictor sharing, *(b)* model prediction, and *(c)* residual generation in Sec 3.1. In Sec. 2.5, *(d)* residual compression is formulated in Eq. 6 and Eq. 7. Then, after *(e)* communicating residual bits, we provide details on *(f)* model recovering in Sec. 3.2. Finally in Sec. 3.3, we describe *(g)* trajectory synchronization.

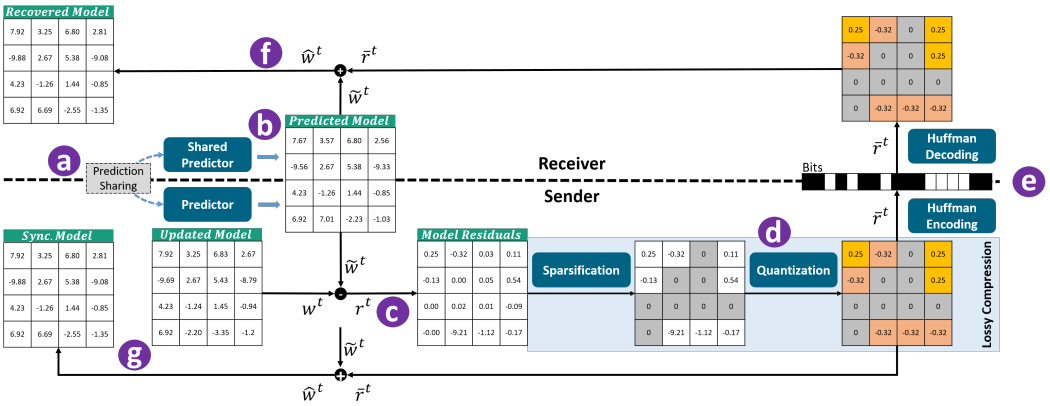


Figure 2: *ResFed* system model with a lossy deep compression pipeline for efficiently transmitting encoded model residuals. The following steps should be implemented for one communication: *(a)* Share the predictor for both sender and receiver before the federated learning starts; *(b)* Execute the same model prediction before communicating any model information; *(c)* Generate the model residuals; *(d)* Compress residuals using deep compression; *(e)* Communicate residual bits with encoding and decoding; *(f)* Recover the model from received residuals; *(g)* Synchronize the model trajectory by simulation, recovering the model locally with consideration of lossy compression.

3.1 Predictor Sharing

Then, we consider to deploy a pair of predictors in both clients and the server, which can execute the model predictions based on the local and global model trajectories in the time series. In our framework, the server caches the local model trajectories in all clients once the local model update is received. Given a client i, if the received models in the server is exactly the same as the local model, and the models in $\hat{\mathscr{G}}_i$ also the same, we say that both trajectories in client i are fully observable in the server.

Give a predictor $f_{predict,i}$ in the client i, we share it to make the predictor in the server $f'_{predict,i} = f_{predict,i}$ in *ResFed*. If the trajectories in client are fully observable in the server, we can get the same model predictions in both server and clients from Eq. 1. Then, by communicating the model residuals, the new model update at time $t+1$ can be recovered by the model residual and the model at last time point. Also, we share the set of predictors in the server $\{h_{predict,i} | i = 1,...,N\}$ to the corresponding client, i.e. $h'_{predict,i} = h_{predict,i}, \forall i \in \{1,...,N\}$. Then we can get the same model predictions based on Eq. 2.

For the predictor, various design choices exist. In this work, we employ the predictor with respect to the model transition dynamics in a sliding history time window. The predictor is formulated as follows:

$$\tilde{w}_i^t = f_{predict}(\mathscr{L}_i^{t-1}, \hat{\mathscr{G}}_i^{t-1}, \hat{w}_i^t) = \begin{cases} \hat{w}_i^t, & T = 0. \\ \hat{w}_i^t + \sum_{\tau=1}^{T} (-1)^{T-\tau} (T-\tau+1)(w_i^{t-\tau} - \hat{w}_i^{t-\tau}), & T > 0. \end{cases} \tag{8}$$

To reduce the used memory for caching trajectories in the client, we apply a short time window $[t-T, t]$ in the prediction process. We term *(i) stationary predictor* when $T = 0$; and *(ii) linear predictor* when $T = 1$. Note that we consider the model updates in Barnes et al. (2020); Mitchell et al. (2022); Isik et al. (2022) as special residuals, which calculated by stationary predictor. We compare the statistical features (sum and variance) of residual values from stationary and linear predictors in Fig. C.3. The residual values from linear predictor are lower and more concentrated than from stationary predictor at the beginning of the federated learning (before convergence), which can potentially achieve a higher accuracy with the same compression ratio, as exactly shown in Fig 3.

Specifically, the stationary predictor uses the current model for the prediction of the next model, $\tilde{w}_i^t = \hat{w}_i^t$, where the model residual is always $r_i^t = w_i^t - \hat{w}_i^t$. Note that when the number of local training

Algorithm 1 : *ResFed*: Residual-based federated learning framework

Server runs:
initialize the global model v
initialize the empty local model trajectories $\hat{\mathscr{L}}_1, ..., \hat{\mathscr{L}}_N$
initialize the global model trajectories $\mathscr{G}_1, ..., \mathscr{G}_N$
initialize the predictor $h_{predict}$
for $i \in \{1, 2, ..., N\}$ **do**
    initialize an empty local model trajectories $\mathscr{L}_i$ ▷ @client i
    initialize an empty global model trajectories $\hat{\mathscr{G}}_i$ ▷ @client i
    $h'_{predict,i} = h_{predict}$ ▷ sharing predictors to client i
    $f'_{predict,i} = f_{predict,i}$ ▷ get the shared predictors from client i
end for
for $t \in \{1, 2, ..., M\}$ **do**
    for $i \in \{1, 2, ..., N\}$ **in parallel do**
        if $t < T$ **then**
            $\hat{\mathscr{G}}_i \leftarrow cache(v)$ ▷ cache the global model in $\hat{\mathscr{G}}_i$ @client i
            server communicates w to the client i
            $\hat{v}_i \leftarrow$ **LocalTrain**(v) ▷ @client i
            client i communicates $\hat{w}_i$ to the server
            $\mathscr{L}_i \leftarrow cache(\hat{v}_i)$ ▷ cache the local model in $\mathscr{L}_i$ @client i
        else
            server communicates $\bar{r}_{i,dl}$ to the client i
            $\bar{r}_{i,ul} \leftarrow$ **ResFedClientUpdate** $(i, \bar{r}_{i,dl})$ ▷ @client i
            client i communicates $\bar{r}_{i,ul}$ to the server
            $\tilde{v}_i \leftarrow f'_{predict,i}(\mathscr{G}_i, \hat{\mathscr{L}}_i, \hat{v})$ ▷ predict updated model
            $\hat{v}_i \leftarrow \tilde{v}_i + \bar{r}_{i,ul}$ ▷ recover models
        end if
        $\hat{\mathscr{L}}_i \leftarrow cache(\hat{v}_i)$ ▷ update local trajectory
    end for
    $v \leftarrow$ **Aggregate**$(\hat{v}_1, ..., \hat{v}_N)$
    for $i \in \{1, 2, ..., N\}$ **do**
        $\tilde{v}_i \leftarrow h_{predict}(\mathscr{G}_i, \hat{\mathscr{L}}_i, v)$ ▷ predict updated model
        $r_{i,dl} \leftarrow v - \tilde{v}_i$ ▷ compute model residuals
        $\bar{r}_{i,dl} \leftarrow h_{compress}(r_{i,dl})$ ▷ compress model residuals
        $\mathscr{G}_i \leftarrow cache(\tilde{v}_i + \bar{r}_{i,dl})$ ▷ synchronize global trajectory
    end for
end for
return v

ResFedClientUpdate $(i, \bar{r})$
$\tilde{u} \leftarrow h'_{predict,i}(\hat{\mathscr{G}}_i, \mathscr{L}_i, u_i)$ ▷ predict updated model
$\hat{u} \leftarrow \tilde{u} + \bar{r}$ ▷ recover models
$\hat{\mathscr{G}}_i \leftarrow cache(\hat{u})$ ▷ update global trajectory
$u_i \leftarrow$ **LocalTrain**$(\hat{u})$
$\tilde{u}_i \leftarrow f_{predict,i}(\hat{\mathscr{G}}_i, \mathscr{L}_i, \hat{w})$ ▷ predict updated model
$r_i \leftarrow u_i - \tilde{u}_i$ ▷ compute model residuals
$\bar{r}_i \leftarrow f_{compress}(r_i)$ ▷ compress model residuals
$\mathscr{L}_i \leftarrow cache(\tilde{u}_i + \bar{r}_i)$ ▷ synchronize local trajectory
return $\bar{r}_i$

epochs is fixed to 1, the stationary residuals is proportional to gradients, i.e. $r = \eta g$, where η is the learning rate and g represents the gradients. In the linear predictor, $\tilde{w}_i^t = \hat{w}_i^t + w_i^{t-1} - \hat{w}_i^{t-1}$, the model transition in the last local training step is always considered, where $r_i^t = w_i^t - \hat{w}_i^t - w_i^{t-1} + \hat{w}_i^{t-1}$. The predictor for the client i in the server $h_{predict,i}$ is similar to $f_{predict,i}$.

3.2 MODEL RECOVERY

We cache the trajectories in both server and clients. Each client has two model trajectories for local and global model updates in the history. In the server, it caches the global trajectory and the local model trajectories of all connecting clients in the history. In this case, the two trajectories in each client are fully observable in the server. Through sharing predictors, given a client i at round t, the server is able to get the same model prediction $\tilde{w}_i^t$ as the client i.

If uncompressed model residuals ($\hat{w}_i^t = w^t$) are received from client i , the model update after local training w_i^t can be recovered in the server as follows:

$$w_i^t = r_i^t + \hat{w}_i^t = r_i^t + w^t \tag{9}$$

where $\hat{w}_i^t$ is the global model in the last round. Similarly, if we predict the global model update in the client, through sharing predictors and uncompressed residuals, the aggregated model can also be recovered in the client.

3.3 MODEL TRAJECTORY SYNCHRONIZATION

Since for the model residuals lossy compression is applied, i.e. $\bar{r} \neq r$, the updated model of a sender w can be recovered in receivers as $\hat{w} = \bar{r} + \tilde{w}$. Therefore, we say that the w cannot be fully recovered in receivers, as $\hat{w} \neq w$. If we cache the original models w in the sender and $\hat{w}$ in receivers, the trajectories in the sender and receiver are different, which leads to drift in results of shared predictors.

To avoid the drift effect, we synchronize the model trajectories in the sender by simulating the recovering process: The originally updated models are not cached in the trajectories; instead, we recover the model from compressed model residuals locally, in order to enforce the trajectories in the sender in the same way as the trajectories in the receiver. The *ResFed* pseudocode is given in Algorithm 1 and the simplified procedures in Algorithm 2.

4 EXPERIMENT

4.1 EXPERIMENTAL SETTINGS

Guided by the previous work by Caldas et al. (2018)Li et al. (2020a), we process and distribute the datasets MNIST (LeCun & Cortes (2010)), Fashion-MNIST (Xiao et al. (2017)), SVHN (Netzer et al. (2011)), CIFAR-10 (Krizhevsky (2009)) and CIFAR-100 (Netzer et al. (2011)) on a set of clients, and train LeNet-5, CNN[1], ResNet-18 on those federated dataset distributively, as shown in Tab. B1. We provide details on the experimental settings in Sec. B.

4.2 RESIDUALS VS GRADIENTS AND WEIGHTS

On top of the basic evaluation in Fig. C.2, we believe deep residual compression can save more communication volume in federated learning with minimum impact on the accuracy. Thus, we demonstrate the federated learning integrating compressing weights, gradients and two different residuals, i.e. stationary and linear residuals in Eq. 8. As shown in Fig. 3, the testing accuracy on both IID and Non-IID datasets from deep residual compression always outperforms weight and gradient compression. Also, the linear residuals can achieve a higher accuracy and faster convergence than stationary residuals. The results indicate communicating residuals in federated learning can enable larger compression ratio per communication round, compared to communicating other parameters.

4.3 COMMUNICATION EFFICIENCY IMPROVEMENT

Next, we evaluate the required communication volume for training three sizes of models on different datasets in both IID and Non-IID settings. Tab. 1 shows that to reach a promising target accuracy, *ResFed* with lossy compression (compression ratio is set on $350\times - 375\times$) can save on average

[1]It consists of 5 convolutional and 3 fully connected layers.

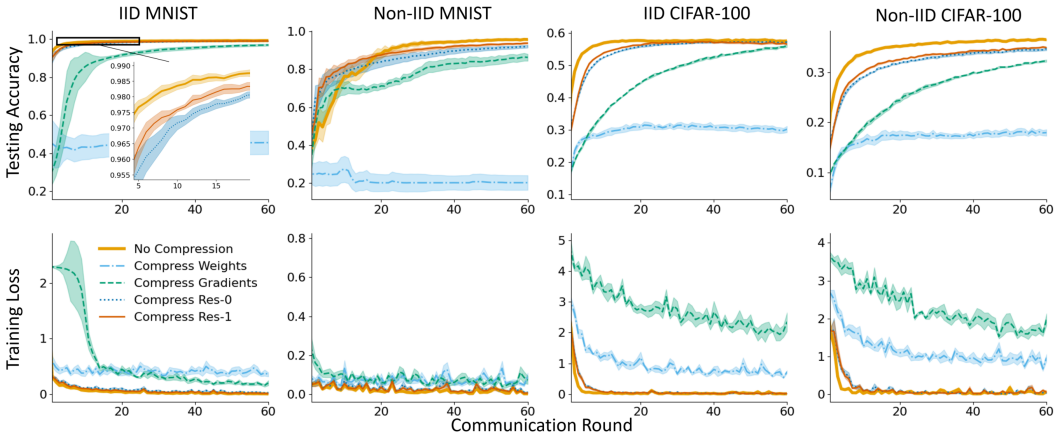


Figure 3: Comparison of compressing weights, gradients, residuals with stationary (Res-0) and linear (Res-1) predictors. The sparsities are 0.2 and 0.01 for training on MNIST and CIFAR-100 distributed in 10 clients, respectively. We quantize the non-zero parameters and use 1 bit to represent each of them. For Non-IID setting, each client owns only the data with half classes.

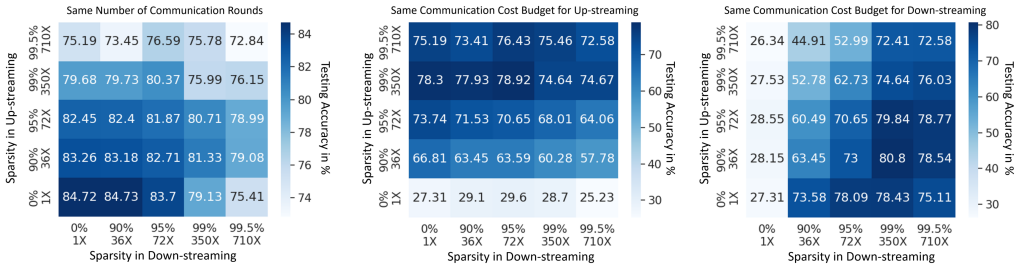
Table 1: The communication volume (CV) and the bitsaving rate (BR) to reach the target accuracy (ACC) for only using *ResFed* in uploading (UL) and downloading (DL). We use FedAvg for both baseline (without any compression) and ResFed (Res-1). Note that the compression ratio per communication round is set from 350× to 375×. More details on experiment setup are shown in Sec. B.

Dataset		Fashion-MNIST		CIFAR-10		SHVN	
		IID	Non-IID	IID	Non-IID	IID	Non-IID
Target ACC		85%		70%		88%	
Baseline	CV	17.73 Mb	29.55 Mb	261.16 Mb	587.61 Mb	7.15 Gb	10.73 Gb
ResFed UL	CV	0.16 Mb	0.28 Mb	4.09 Mb	6.79 Mb	0.08 Gb	0.17 Gb
	BR	**99.10%**	**99.10%**	**98.43%**	**98.84%**	**98.89%**	**98.42%**
ResFed DL	CV	0.10 Mb	0.21 Mb	1.48 Mb	4.61 Mb	0.07 Gb	0.11 Gb
	BR	**99.43%**	**99.30%**	**99.43%**	**99.22%**	**99.02%**	**98.97%**

around 99% of the total communication volume for all clients in only up- or down-streaming. Furthermore, the bitsaving ratios of *ResFed* on IID and Non-IID settings are similar, which indicates the compression performance of *ResFed* is robust to data heterogeneity in federated learning. We show testing accuracy and training loss change with increasing required communication volume in *ResFed* in Fig. B.1. The results indicate communicating residuals in federated learning can remarkably save overall communication volume.

4.4 Scalability for Resource-constrained Communication Environments

Finally, we explore the scalability of *ResFed* by tuning compression ratios for client-to-server and server-to-client, as in real application scenarios. The available network resources for up- and down-streaming can be heterogeneous. Fig 4 shows the test accuracy effected for different values of sparsity, which leads to various compression ratios for each communication round for up- and down-streaming. From Fig. 4a, we can observe the testing accuracy reduces with higher compression ratio per communication round, when the number of communication rounds is always the same, i.e. set to 300. However, when we consider the dedicated budget for communication costs in up- or down-streaming, a large compression ratio in *ResFed* can achieve better accuracy, as shown in Fig. 4b. By



(a) Number of communication rounds (300) is the same.

(b) The communication cost budget (14 Mb) is the same in up-streaming (left) and down-streaming (right).

Figure 4: Testing accuracy on various values of sparsity and compression ratio per communication round in deep residual compression for both up- and down- streaming in *ResFed* (Res-1). We train a CNN model on a federated CIFAR-10 dataset with 10 clients. The testing accuracy decreases with a higher sparsity at the same number of communication rounds, while when the communication resource is constrained, the deep residual compression with higher sparsity can achieve more compromising testing accuracy.

adapting the compression ratio in resource-constrained communication environments, *ResFed* can effectively enhance the federated learning using deep residual compression.

5 Related Work

Deep compression. Deep compression was originally proposed by Han et al. (2016) and aims at compressing deep learning models by a pipeline including sparsification (pruning), quantization and encoding for more efficient deployment. Based on deep compression, Lin et al. (2018) have proposed deep gradient compression to reduce the communication costs in distributed learning by compressing gradients rather than model weights, which can also be used in federated learning.

However, models are usually trained more than one epoch locally in federated learning (McMahan et al. (2017); Li et al. (2020b); Wang et al. (2020); Karimireddy et al. (2020)), which results in gradient accumulation instead of gradients in other distributed learning scenarios (Sattler et al. (2019)). Our conducted experiments also show compressing residuals can achieve better communication efficiency comparing to compress gradients due to the additional prediction step. In *ResFed*, we especially consider residuals, which eliminate the model similarity in a single inter-round of federated learning communication and achieve a better compression performance by leveraging the deep residual compression.

Federated learning and communication efficiency. Communication efficiency is the key for deploying federated learning in real application scenarios, especially to train a large model. Previous research by Yuan & Ma (2020); Karimireddy et al. (2020); Hamer et al. (2020) attempted to reduce the number of needed communication rounds for a better communication efficiency. Meanwhile, the proposed approaches by Xu et al. (2022); Reisizadeh et al. (2020); Hönig et al. (2022) are built upon deep compression and focus on improving communication efficiency by decreasing the communication volume. However, unlike compressing residuals in *ResFed*, they compressed model weights without consideration of any potential redundancy in sequential updating of federated learning. The recent work by Yue et al. (2022) has also mentioned the predictive model update in federated learning, which is concurrent to our work, but the information in the history of model updating is not considered for reducing the parameter redundancy there.

Additionally, all those algorithms above can only be used to improve the communication efficiency for up-streaming, while *ResFed* can handle with up- and/or down-streaming for heterogeneous resource-constrained environments.

Residuals in video encoding. The residuals have been widely and successfully utilized in video encoding since H.261 (Girod et al. (1995); Li et al. (2021)). By considering inter-frame correlations, the pixel values in the current frame are predicted from history frames and then only residuals, i.e. the deviations between predicted and the actual pixel values in the current frame, are encoded and

streamed to the receivers. Inspired by the residuals in video encoding, we integrate the model residuals into federated learning in *ResFed*, where the inter-round similarity of a model update is analogous to inter-frame correlation in video encoding.

6 CONCLUSION

In this work, we introduce a residual-based federated learning framework, which allows clients and the server to share residuals instead of weights or gradients. It achieves more efficient communication for both up- and down-streaming in federated learning by leveraging deep residual compression, and hence can be flexibly deployed in heterogeneous network environments. Our conducted experimental evaluation shows that the framework remarkably reduces overall communication volume to reach the same prediction accuracy in standard federated learning. Compared to compressing model weights or gradients, *ResFed* achieves higher accuracy and faster convergence speed.

Limitations. We cache the recovered models as local and global trajectories for continual model prediction and residual computing in all clients and server. Assuming that we perform *ResFed* with N clients for training a model with V 32-bit float parameters and we set the trajectory length on T, each client should use $2*32*V*T$ bits memory to cache the 2 trajectories. Thus, the additional required memory size in each client is proportional to $T*V$. For the server, it needs $2*32*V*T*N$ bits memory to cache the local and global trajectories for all clients. In order to reduce the required memory, a potential solution is to cache the compressed models in the trajectories for both sender and receiver symmetrically, after model recovery. However, the accuracy of model prediction based on compressed trajectories is reduced, then the memory-accuracy trade-off needs to be investigated in future work.

7 REPRODUCIBILITY STATEMENT

We provide the source-code for the implementation and evaluation of our proposed framework *ResFed* in the code appendix. The user guidelines for installation and execution is given in the file *README.md*. The details on our conducted experiments are provided in Sec. B. The source-code will be made publicly available after double-blind review.

REFERENCES

Leighton Pate Barnes, Huseyin A Inan, Berivan Isik, and Ayfer Özgür. rtop-k: A statistical estimation approach to distributed sgd. *IEEE Journal on Selected Areas in Information Theory*, 1(3): 897–907, 2020.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Bernd Girod, Eckehard G Steinbach, and Niko Faerber. Comparison of the H.263 and H.261 video compression standards. In *Standards and Common Interfaces for Video Information Systems: A Critical Review*, volume 10282, pp. 230–248. SPIE, 1995.

Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. FedBoost: A communication-efficient algorithm for federated learning. In *International Conference on Machine Learning*, pp. 3973–3983. PMLR, 2020. URL `https://proceedings.mlr.press/v119/hamer20a.html`.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, May 2016.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Robert Hönig, Yiren Zhao, and Robert Mullins. DAdaQuant: Doubly-adaptive quantization for communication-efficient federated learning. In *International Conference on Machine Learning*, pp. 8852–8866. PMLR, 2022.

Berivan Isik, Tsachy Weissman, and Albert No. An information-theoretic justification for model pruning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3821–3846. PMLR, 2022.

Sai Praneeth Karimireddy et al. SCAFFOLD: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pp. 5132–5143. PMLR, 2020.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Computer Science Department, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. pp. 58.

Yann LeCun and Corinna Cortes. Mnist handwritten digit database, 2010. URL https://arxiv.org/abs/1812.01097. URL: http://yann.lecun.com/exdb/mnist/.

Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=evqzNxmXsl3.

Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=ByexElSYDr.

Tian Li et al. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020b.

Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkhQHMW0W.

Lingjuan Lyu, Han Yu, Xingjun Ma, Chen Chen, Lichao Sun, Jun Zhao, Qiang Yang, and Philip S Yu. Privacy and robustness in federated learning: Attacks and defenses. *arXiv preprint arXiv:2012.06337*, 2020.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54, pp. 1273–1282, 2017.

Nicole Mitchell, Johannes Ballé, Zachary Charles, and Jakub Konečnỳ. Optimizing the communication-accuracy trade-off in federated learning with rate-distortion theory. *arXiv preprint arXiv:2201.02664*, 2022.

Yuval Netzer et al. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Colin Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Sparse binary compression: Towards distributed deep learning with minimal communication. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Jan Van Leeuwen. On the construction of Huffman Trees. In *Third International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 382–410, July 1976.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in Neural Information Processing Systems*, 33:7611–7623, 2020.

Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1):1–8, 2022.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms, August 2017. arXiv:cs.LG/1708.07747.

Jinjin Xu, Wenli Du, Yaochu Jin, Wangli He, and Ran Cheng. Ternary compression for communication-efficient federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(3):1162–1176, 2022. doi: 10.1109/TNNLS.2020.3041185.

Honglin Yuan and Tengyu Ma. Federated accelerated stochastic gradient descent. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5332–5344. Curran Associates, Inc., 2020.

Kai Yue, Richeng Jin, Chau-Wai Wong, and Huaiyu Dai. Communication-efficient federated learning via predictive coding. *IEEE Journal of Selected Topics in Signal Processing*, 16(3):369–380, 2022. doi: 10.1109/JSTSP.2022.3142678.

A TABLES OF NOTATIONS

We provide an overview of the most relevant notations in Tab. A1.

Table A1: Summary of mainly used notations in this paper.

Notation	Meaning	Navigation
i	Index of clients	Sec. 2.1
t	Index of communication round	Sec. 2.1
η	Learning rate	Sec. 3.1
N	Number of connected clients	Sec. 2
T	Number of total communication rounds	Sec. 2.2
V	Number of parameters in a machine learning model	Sec. 6
w	Model weights	Sec. 2.1
u	w in clients	Sec. 2.1
v	w in server	Sec. 2.1
w^t	Model weights in communication round t, also for u^t, v^t	Sec. 2.1
w_i	Model weights for client i, also for u_i, v_i	Sec. 2.1
$\bar{w}$	Compressed model weights, also for $\bar{u}, \bar{v}$	Sec. 2.5
$\tilde{w}$	Predicted model weights, also for $\tilde{u}, \tilde{v}$	Sec. 2.3
$\hat{w}$	Received model weights for update, also for $\hat{u}, \hat{v}$	Sec. 2.1
g	Model gradients	Sec. 3.1
r	Model residuals	Sec. 2.4
r^t	Model residuals in communication round t	Sec. 2.4
r_i	Model residuals in client i	Sec. 2.4
$r_{i,dl}$	Model residuals of client i for uploading in client i	Sec. 2.4
$r_{i,ul}$	Model residuals of server for downloading in client i	Sec. 2.4
$\bar{r}$	Compressed model residuals	Sec. 2.5
$\hat{r}$	Received model residuals	Sec. 2.1
$\mathscr{D}$	Data set	Sec. 2.1
$\mathscr{G}$	Global trajectory queue	Sec. 2.2
$\mathscr{L}$	Local trajectory queue	Sec. 2.2

B EXPERIMENTAL DETAILS AND FURTHER RESULTS

In this section, we provide the details on our conducted experiment in 4. We run on a computer cluster with 4× NVIDIA-A100-PCIE-40GB GPUs and 4× 32-Core-AMD-EPYC-7513 CPUs. The environment is a Linux system with Pytorch 1.8.1 and Cuda 11.1.

We demonstrate the learning task on 5 different datasets:

- MNIST LeCun & Cortes (2010): 60000 data points in the training set and 10000 data points in the test set. Each data point is a 28x28 gray-scale digit image, associated with a label from 10 classes.
- CIFAR-10 Krizhevsky (2009): 50000 data points in the training set and 10000 data points in the test set. Each data point is a 32x32 RGB image, associated with a label from 10 classes.
- Fashion-MNIST Xiao et al. (2017): 60000 data points in the training set and 10000 data points in the test set. Each data point is a 28x28 gray-scale image, associated with a label from 10 classes.
- SVHN Netzer et al. (2011): 73257 data points in the training set and 26032 data points in the test set. Each data point is a 32x32 RGB digit image, associated with a label from 10 classes.
- CIFAR-100 Netzer et al. (2011): 50000 data points in the training set and 10000 data points in the test set. Each data point is a 32x32 RGB image, associated with a label from 100 classes.

The models trained on those dataset are shown in Tab. B1.

Table B1: Dataset and models in experiments

Dataset	MNIST	CIFAR-10	Fashion-MNIST	SVHN	CIFAR-100
Model	LeNet-5	CNN	LeNet-5	ResNet-18	CNN
# of Param	61706	1020160	61706	11175370	1020160
Size in MB	0.25	4.08	0.25	44.70	4.08

B.1 Experiments for Sec. 4.2

We divide the dataset, i.e. MNIST and CIFAR-100, into 10 clients and run the FedAvg with local optimizer of stochastic gradient descent (SGD) (momentum is 0.9) to train LeNet-5 and CNN (with 5 convolutional and 3 fully connected layers), respectively. The learning rate is fixed on 0.01 and the batch size is 64 for all tests. The number of local epoch is 5. In Non-IID data setting, each client owns only 2 out of 10 classes in MNIST and 50 out of 100 classes in CIFAR-100.

In clients, we consider 5 different approaches as follows:

- No Compression: The standard federated learning without any compression methods is used as the baseline, which provides the best results among all of the approaches, when number of communication rounds is the same.
- Compress Weights: Before communication in standard federated learning, the model weights are first compressed.
- Compress Gradients: The gradients in each epoch are compressed and communicated to the server.
- Compress Res-0: The residuals are computed by stationary predictor, i.e. Eq. 8 with $T = 0$.
- Compress Res-1: The residuals are computed by linear predictor, i.e. Eq. 8 with $T = 1$.

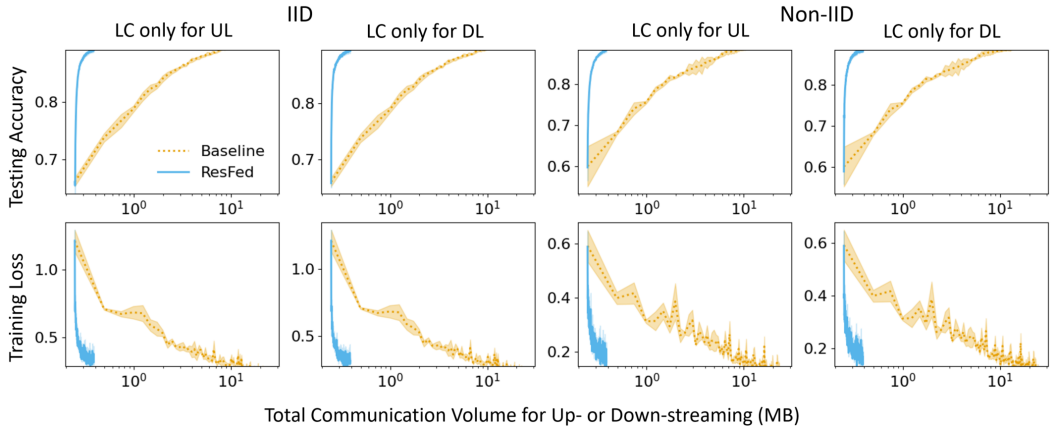
For each approach on each dataset, we run 10 tests with different seeds and show the mean value and standard variance in Fig. 3.

Lossy Compression. We compress the those model parameters using deep compression pipeline (Han et al. (2016)) only for client-to-server. In particular, we set sparsity on 80% and 99% for residuals in LetNet and CNN, respectively. We use SGD optimizer momentum of 0.9. Those sparsified parameters are zero-parameters and the number of the continually appearing zero-parameters are encoded as a 16-bit float parameters (Lin et al. (2018)). After that, we quantize the non-zero parameters in 1 bit with median value of positive and negative parameters (Xu et al. (2022)). Finally, Huffman encoding (Van Leeuwen (1976)) is used.

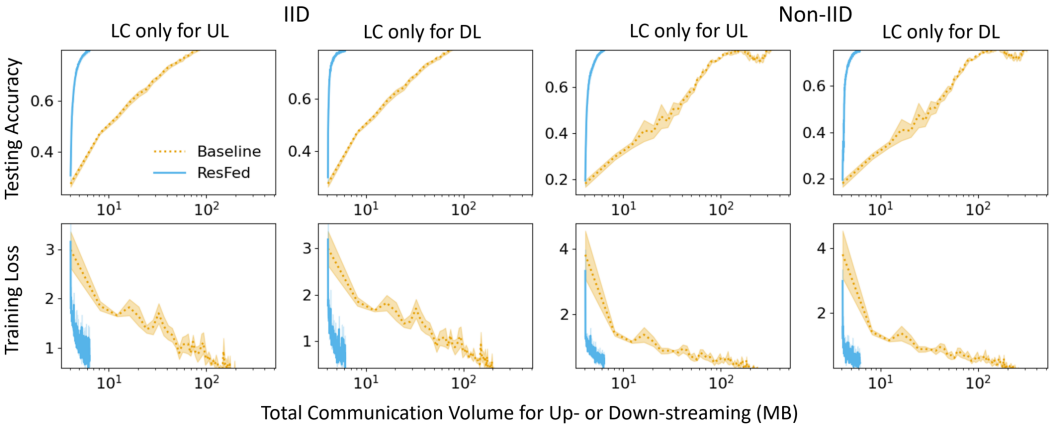
B.2 Experiments for Sec. 4.3

We train LeNet-5, CNN and ResNet18 (size from small to large) on Fashion-MNIST, CIFAR-10 and SVHN with 10 clients in both IID and Non-IID settings. We demonstrate the ResFed and lossy compress the residuals either only for uploading (UL) or downloading (DL) to study the effects on each direction separately. The learning rate is fixed on 0.01 and the batch size is 64 for all tests. In Non-IID data setting, each client owns 50% classes (5 out of 10). We use mean values from 5 tests in each experiment for the evaluation in Tab. 1 and show the results with standard variance in Fig. B.1, which indicate the overall communication volume can be remarkably reduced in *ResFed*.

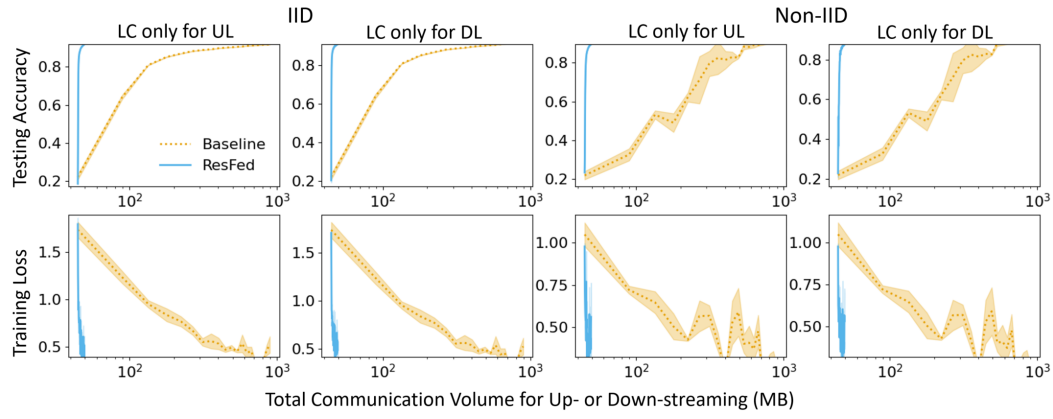
For all experiments, we set the sparsity on 99% and use 1 bit for each non-zero parameters. Consequently, the compression ratio per communication round for LetNet-5 is about 350×, CNN is about 375× and ResNet-18 is about 356×.



(a) Training LeNet-5 on Fashion-MNIST.



(b) Training CNN on CIFAR-10.



(c) Training ResNet-18 on SVHN.

Figure B.1: Overall communication efficiency enhanced by *ResFed* with Lossy Compression (LC) for only Uploading (UL) or Downloading (DL)

B.3 Experiments for Sec. 4.4

To show the correlation between deep residual compression in up- und down-streaming, we train the CNN model on IID CIFAR-10 with 10 clients and tune the sparsity for realizing different compression ratios per communication round in *ResFed*. The learning rate is fixed on 0.01 and the batch size is 64. We use SGD optimizer momentum of 0.9. The number of local epoch is 1. Specifically,

the value of sparsity is $\{0\%, 90\%, 95\%, 99\%, 99.5\%\}$ for both up- and down-streaming and then set 1 bit for all non-zero parameters in quantization.

C UNDERSTANDING RESIDUALS

We provide an illustration of model transitions during federated learning in Fig. C.1. Given a sender and a receiver (both can be a client or a server), the communication and operation result in model transition. Note that for a client, the operation is local training; for a server, the operation is aggregation. We consider the model transition caused by an operation as a internal model transition, and by communication as a external model transition. Then, as explained in Sec. 2.1, the model is updated twice between two communication rounds, which is shown in Fig. C.1 as dual model transition. Consequently, we can have an internal and an external model transition trajectory in both sender and receiver. Note that for a client, the internal model transition trajectory is a local model trajectory; for a server, the internal model transition trajectory is a global model trajectory, as described in Sec. 2.2.

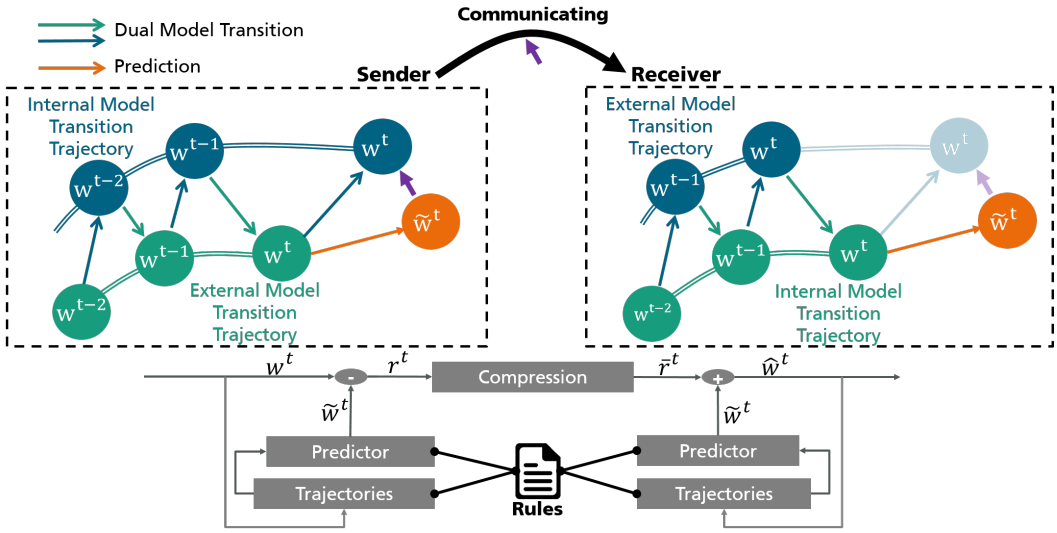


Figure C.1: Residual generation in *ResFed* during model transitions. For a better overview, we simplify the system by disregarding the trajectory synchronization step in Sec. 3.3.

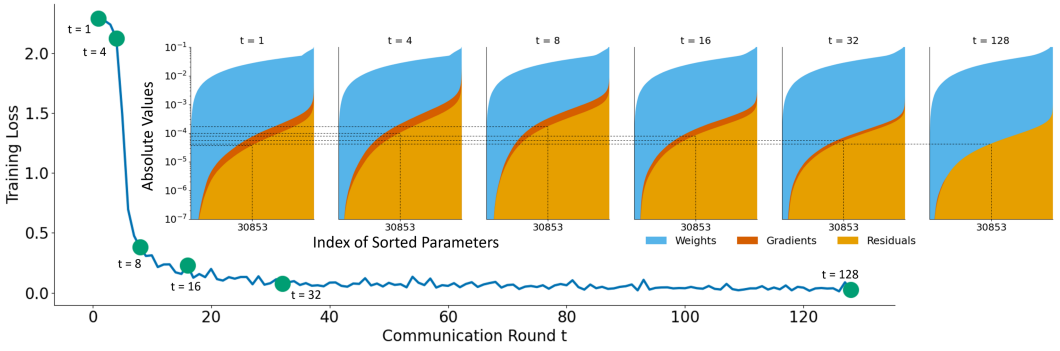


Figure C.2: Value comparison of model parameters, gradients and residuals in federated learning. We train a LeNet with 61706 weights of 32-bits float on MNIST distributed among 10 clients, with fixed learning rate 0.001 and batch size 64. For fairly comparing gradients and residuals, the number of local epoch in each client is set as 1. We set 6 checkpoints when the number of communication rounds is $\{1, 4, 8, 16, 32, 128\}$. The results show that most values of residuals are smaller than weights and gradients during the training. It indicates that lossly compressing residuals naturally lose less information and have a smaller affect on the accuracy.

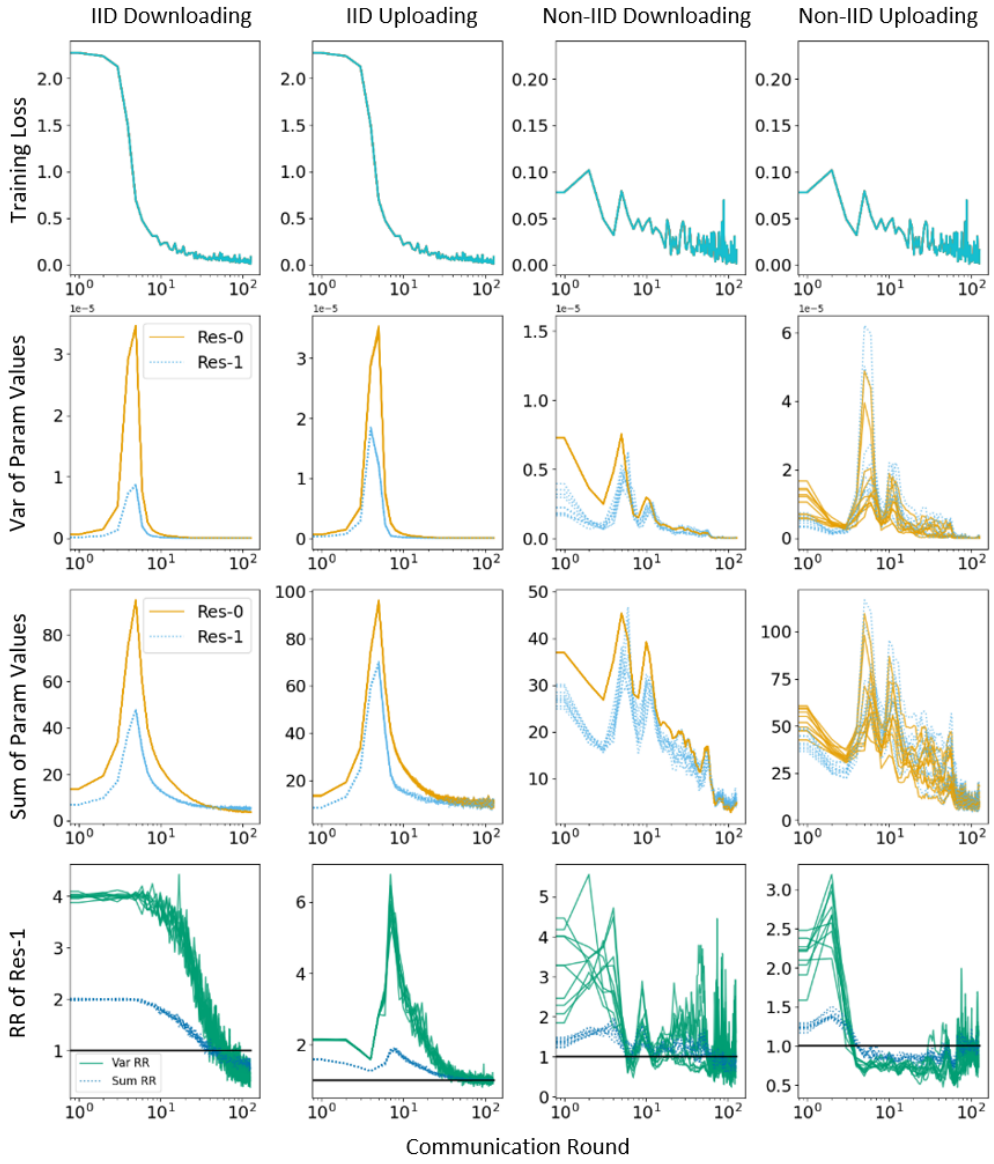


Figure C.3: Comparison of linear residuals (Res-1) and stationary residuals (Res-0) while communicating in down- and uploading (server-to-client and client-to-server) for federated learning on IID and Non-IID MNIST for each of 10 clients, with fixed learning rate 0.001 and batch size 64. The number of local epoch in each client is set as 3. We show the training loss, variance (Var) and sum of parameter (Param) values in the first 3 rows. In the fourth row, we evaluate the residuals using reduction ratio (RR) per communication round, where the *Var RR* denote the reduction ratio ($\times$) of parameter value variance from Res-1 w.r.t. Res-0, i.e. Var_{res-0}/Var_{res-1}, and similarly, the *Sum RR* denote the reduction ratio ($\times$) of parameter value sum from Res-1 w.r.t. Res-0, i.e. Sum_{res-0}/Sum_{res-1}. The comparison results show that the parameter values in Res-1 are lower and more concentrated than in Res-0 at the beginning of the federated learning (before convergence), which allows us to compress it with less information loss and higher accuracy using the same sparsification and quantization. The conclusion is consistent with the results shown in Fig. 3

Then, *ResFed* allows the sender to predict the model for the next internal model transition, which is shown in orange. Meanwhile the sender does the operation to execute the internal model transition and residuals (in purple) are deduced from the difference of both model transition results. We believe

the predicted model can be closer to the updated model than the previous model, which leads to smaller values of residuals. To evaluate it, we set 6 checkpoints when the number of communication rounds is $\{1, 4, 8, 16, 32, 128\}$, and show the values of model weights, gradients and residuals in Fig. C.2. Based on this, the residuals can be compressed smaller than weights and gradients. For a clear big picture of *ResFed* formulated in Algorithm 1, we provide a simplified formulation in Algorithm 2.

Finally, We conduct the experimental study on model residuals from stationary (Res-0) and linear (Res-1) predictors. The comparison results show that the parameter values in Res-1 are lower and more concentrated than in Res-0 at the beginning of the federated learning (before convergence), which allows us to compress it with less information loss and higher accuracy using the same sparsification and quantization. The conclusion is consistent with the results shown in Fig. 3.

Algorithm 2 : Simplified residual-based federated learning framework

server initializes the global model, empty local and global model trajectories, and the predictor at the server
for $i \in \{1, 2, ..., N\}$ **do**
    client i initializes empty local and global model trajectories, and the predictor
    client i and server share the predictors to each other
end for
for each communication round t **do**
    for each client i **in parallel do**
        if $t < T$ **then**
            do normal federated learning and update trajectories
        else
            server communicates $\bar{r}_{i,dl}$ to the client i
            $\bar{r}_{i,ul} \leftarrow$ **ResFedClientUpdate** $(i, \bar{r}_{i,dl})$ ▷ @client i
            client i communicates $\bar{r}_{i,ul}$ to the server
            server recovers models $\hat{v}_i$ based on predicted models $\tilde{v}_i$
        end if
        server updates local trajectory
    end for
    $v \leftarrow$ **Aggregate**$(\hat{v}_1, ..., \hat{v}_N)$
    for each client i **do** ▷ @server
        server computes residuals $r_{i,dl}$ based on predicted model $\tilde{v}_i$
        server compresses residuals $\bar{r}_{i,dl}$ and synchronizes global trajectory
    end for
end for
return v

ResFedClientUpdate $(i, \bar{r})$
client i recovers models $\hat{u}$ based on predicted models
client i updates global trajectory
$u_i \leftarrow$ **LocalTrain**$(\hat{u})$
client i computes residuals r_i based on predicted models $\tilde{u}$
client i compresses residuals $\bar{r}_i$ and synchronizes global trajectory
return $\bar{r}_i$