FEDBFPT: An Efficient Federated Learning Framework for BERT Further Pre-training

Xin'ao Wang , **Huan Li** * , **Ke Chen** * and **Lidan Shou**

Key Lab of Intelligent Computing Based Big Data of Zhejiang Province
Zhejiang University, Hangzhou, China

{wangxin.ao, lihuan.cs, chenk, should}@zju.edu.cn

Abstract

This study proposes FEDBFPT (Federated BERT Further Pre-Training), a Federated Learning (FL) framework for further pre-training the BERT language model in specialized domains while addressing privacy concerns. FEDBFPT enables multiple clients to collaboratively train the shallower layers of BERT, which are crucial in the pre-training stage, without the need to share private data. To achieve this, FEDBFPT involves building a local model for each client, progressively training the shallower layers of local models while sampling deeper layers, and aggregating trained parameters on a server to create the final global model. This approach utilizes multiple smaller local models to further pre-train a global model targeted at specific tasks via fine-tuning, resulting in a reduction in resource usage while maintaining model accuracy. Theoretical analysis is conducted to support the efficiency of FEDBFPT, and experiments are conducted on corpora across domains such as medicine, biology, and computer science. Results indicate that FEDBFPT achieves performance levels comparable to traditional FL methods while reducing computation and communication costs by 46.70% and 7.04%, respectively, even approaching the performance of centralized training models. The Source code is released at https://github.com/Hanzhouu/FedBFPT.

1 Introduction

Federated Learning (FL) [McMahan *et al.*, 2017] is a technique that enables multiple clients to jointly train a global model without sharing private data. However, training methods in FL suffer from high communication and computational costs, as client devices often possess limited hardware resources and network bandwidth. Consequently, training complex models within the FL framework becomes challenging, especially when dealing with pre-trained language models that have demonstrated significant advancements in natural language processing tasks, such as ELMo [Peters *et al.*, 2018], GPT [Radford *et al.*, 2018], BERT [Devlin *et al.*, 2019], and RoBERTa [Liu *et al.*, 2019]. These models are typically trained on general corpora but can yield better performance on specialized domain tasks after undergoing *further pre-training [Beltagy* et al.*, 2019]* using specialized datasets. However, collecting such specialized datasets centrally poses privacy concerns, making it infeasible.

Previous research has shown that neural networks tend to stabilize from shallower to deeper layers during training [Raghu *et al.*, 2017]. This has been observed in the popular pre-trained language model BERT, where shallower layers have been shown to capture phrase-level information, which holds greater significance in model pre-training compared to the deeper layers [Jawahar *et al.*, 2019; Hao *et al.*, 2019; Manginas *et al.*, 2020; Wang *et al.*, 2022b]. The observation suggests that it might be feasible to train only a subset of layers on the client side initially and gradually increase the number of trained layers, thereby achieving efficient training of large models in an FL setting.

In this paper, we focus on the cost-effectiveness of BERT's pre-training in FL by asking the following questions: (1) Can we design a computational and communication efficient method to complete the training of large models in resource-constrained clients? (2) Can such a method achieve the accuracy of traditional FL or even centralized training?

Our answers to both questions are "Yes." Through investigation, we propose FEDBFPT (Federated BERT Further Pre-Training), an efficient framework that trains only part of the layers of BERT to reduce the computational and communication costs. FEDBFPT allows for training a large global model using FL by creating small local models for each client to train partial layers of the global model, incurring less computational resources and fewer weights to transmit. Within FEDBFPT, we also propose a method PL-SDL (Progressive Learning with Sampled Deeper Layers), which can train the crucial layers of the global model efficiently on local models with small capacity.

We perform downstream tasks via fine-tuning the resultant global model on specialized domains such as medicine, biology, and computer science, and show that FEDBFPT can achieve similar performance to traditional FL while using 46.70% and 7.04% of the original model's computational and communication costs, respectively, and can even approach the accuracy of centralized trained models.

*Ke Chen and Huan Li are the corresponding authors.

In summary, our main contributions are as follows.

- We propose FEDBFPT, an efficient FL framework for further pre-training the BERT language model on the client without sharing private corpora (cf. Section 3.2).
- We propose PL-SDL, a method for efficiently training the important layers of the global model on the client side with small-capacity local models (cf. Section 3.3).
- We conduct the analysis on the computational and communication costs (cf. Section 4); We also conduct experiments on specialized domains corpus such as medicine, biology, and computer science, validating the efficiency and effectiveness of FEDBFPT in comparison with many baseline methods (cf. Section 5).

2 Related Work

Federated Learning. Federated Learning (FL) [McMahan *et al.*, 2017] is a technique that enables multiple clients to work collaboratively to train a model without sharing their private data. The general FL process typically involves two types of updates, namely local updates and global updates. Particularly, local updates focus on minimizing a local loss function, whereas global updates consider aggregating local weights from clients and synchronizing the model changes across all clients at each communication iteration.

In FL, clients typically have limited resources, including less powerful hardware and limited network bandwidth, which can hinder training large models. To address these challenges, researchers have developed techniques from various perspectives [Wang *et al.*, 2022a], including:

1. *Message Compression* that represents gradients (or parameters) with fewer bits to achieve reduced communication cost. Prominent examples are quantization [Alistarh *et al.*, 2017; Lin *et al.*, 2018; Fu *et al.*, 2020] and sparsification [Stich *et al.*, 2018; Konečnỳ *et al.*, 2016].
2. *Model Pruning* that identifies (much smaller) subnetworks within the original model for less computational cost at the inference phase [Li and Wang, 2019; Lin *et al.*, 2020].
3. *Model Distillation* [Hinton *et al.*, 2015] that allows the server to extract knowledge from clients using hold-out datasets [Li and Wang, 2019; Lin *et al.*, 2020].

Message compression helps reduce communication costs, but it does not address the main bottleneck in FL, which is the cost of computerization on the client side. Techniques on model pruning and knowledge distillation can create small, accurate models to speed up inference, but they are ineffective in training large models. Our approach employs the local model to train only the important layers of the global model and use the resultant global model for fine-tuning. By using small models during training, we save computational resources and only transmit the important layer parameters to save on communication costs. This way balances the use of small models for computational efficiency and large models for effectiveness.

Progressive Learning. Progressive Learning, a paradigm initially proposed to stabilize the training process, has been widely used in computer vision tasks such as image synthesis [Karras *et al.*, 2018], image super-resolution [Wang *et al.*, 2018], facial attribute editing [Wu *et al.*, 2020], and representation learning [Li *et al.*, 2020b]. The core idea is to start training on easier tasks, such as low-resolution outputs or shallower models, and gradually progress to more difficult but desired tasks, such as high-resolution outputs or deeper models [Wang *et al.*, 2022a]. The cost-saving benefit of progressive learning, i.e., using shallower models during initial training stages, has not been explored in the context of FL.

Symbol	Meaning
C_k	The k-th client
D_k	The local dataset at C_k
P_k	The local parameter pool at C_k
ℓ	The layer index number
m^G	The number of T-layers in the global model
m_k^L	The number of T-layers in the k-th local model
i, I	The current FL iteration and iteration threshold

Table 1: Notation

Further Pre-Training. Pre-trained language models are usually trained on general text corpora, which makes them perform well in general domains tasks but poorly in specialized scientific domains tasks. Some studies, such as SciBERT [Beltagy *et al.*, 2019], BioBERT [Lee *et al.*, 2020], have demonstrated that further pre-training using specialized domain-specific corpus can improve performance in those domains tasks. However, in specialized scientific domains, data is often stored in different professional institutions, leading to privacy concerns during the collection process. Motivated by this, we propose to use FL to protect user privacy, as well as competent techniques to address the issue of limited resources of FL clients and their inability to train large models.

3 FEDBFPT

Below, we go through the concepts, present the overall framework, and detail the internal designs.

3.1 Definitions and Notation

The commonly used notation is listed in Table 1.

In Federated Learning (FL), a centralized server and multiple clients (labeled as C_k, where $1 \leq k \leq N$) are used. Initially, the server has a **global (BERT) model** that has already been *pre-trained* from large-scale general corpora. A BERT model consists of an embedding layer, a sequence of the intermediate transformer layers (**T-layers**), and an output layer. Typically, the size of the output layer is much smaller than those of the embedding layer and T-layers. Meanwhile, each client C_k has a non-shared, domain-specific corpus (dataset), labeled as D_k, that can be used to *further pre-train* the global model for increased accuracy at that domain-specific task. To achieve this, each client C_k runs a **local (BERT) model** and the Masked Language Modelling (MLM) [Devlin *et al.*, 2019] task is performed to update the weights of the local model using the domain-specific corpus. Later, the weights of all local models are uploaded to the server to generate the further pre-trained global model, which needs to be *fine-tuned*

subsequently to accomplish a specific task, e.g., classification [Collier and Kim, 2004] or Named Entity Recognition (NER) [Luan *et al.*, 2018].

In total, three training processes are involved, namely **pre-training** that generates the pre-trained BERT model from the general corpora, **further pre-training** that generates a further pre-trained BERT model from the domain-specific corpus, and **fine-tuning** that generates a fine-tuned model that targets a domain-specific task. In this study, we focus on the further pre-training process and aim to propose an FL-based framework for further pre-training the global model at the server using local datasets distributed at the clients.

Considering practical needs, the framework should have low computational and communication costs for clients, while preserving the performance of the global BERT model after being further pre-trained. To ensure fair and reproducible comparisons with previous work, we define

- **computational cost of a client** as the training time of the local model in a fixed setting of hyperparameters, training data size, and hardware specification;
- **communication cost of a client** as the size of the parameters that a client sends to the server;
- **final performance of the global model** as the effectiveness at the downstream task (e.g., accuracy in classification or F1 in NER) after a fixed round of fine-tuning.

We identify two variables that are related to the above performance indicators, namely the number of layers of the local model and the number of layers to be trained and transmitted at client C_k, denoted as m_k^L and m_k^T, respectively. Intuitively, a smaller m_k^L will reduce C_k's computational cost as well as the final performance of the global model. Likewise, a smaller m_k^T will reduce C_k's communication cost while reducing the final performance of the global model as well.

Next, we will present our overall FL-based framework (cf. Section 3.2) and the associated techniques (cf. Section 3.3), which can maintain the final performance of the global model with small m_k^L and m_k^T.

3.2 Framework Overview

The proposed FEDBFPT framework is illustrated in Figure 1.

Specifically, the server runs a global model while each client runs a local model that keeps the same embedding and output layers as the global model but a reduced sequence of T-layers in between. Let m^G and m_k^L be the number of T-layers in the global model and the k-th local model, respectively. We stipulate that $m^G > m_k^L$ is such that the local model can be deployed and trained at resource-constrained end devices. In the implementation, we set $m^G = 12$ and $m_1^L = \ldots = m_N^L = 6$. Nevertheless, our proposed framework supports using different m_k^L for the local models at different clients. Moreover, each local model is associated with a *parameter pool*, which is implemented as a dictionary that organizes model parameters as key-value pairs. It is used to help create the local model and will be updated in each iteration of FL. At the initial stage, each client stores a local dataset that will be used to further pre-train the global model at the server in a collaborative manner.

We proceed to go through the steps of the further pre-training of the global BERT model in Figure 1. The whole process is formalized in Algorithm 1.

Algorithm 1 FEDBFPT

Input: datasets $\{D_1, \ldots, D_N\}$, global model G with initial model parameters $W^{(0)}$, epoch threshold I, local MLM training epoch e_k and layer index number ℓ.
Output: final model parameters $W^{(I)}$ of G.
Initialize: parameter pool P_k of each client C_k; $\ell \leftarrow 0$.
`// FL training for I iterations`
for $i = 1$ **to** I **do**
  `// Parallel executions at clients`
  for each client C_k **do**
    if $i > 1$ **then**
      update P_k with $W^{(i-1)}$ fetched from the server
    end if
    `// Train ℓ-th T-layer and the output layer of the local model and obtain the weights`
    $W_k^{(i)} \leftarrow \text{MLM\_Train}(D_k, P_k, \ell, e_k)$
    upload $W_k^{(i)}$ to the server
  end for
  `// Merge clients' weights at server`
  $W^{(i)} \leftarrow \text{Merge}(W_1^{(i)}, \ldots, W_N^{(i)})$
  determine the next layer index number ℓ
  sent $W^{(i)}$ and ℓ to each client
end for

S1 Each client C_k fetches the parameters of the global model from the server and persists the fetched parameters in the local parameter pool. Simultaneously, an index number ℓ is received from the server, which indicates the specific ℓ-th T-layer to be trained and updated for the local model at C_k. The determination of the layer index number ℓ is to be detailed in Section 3.3.

S2 With the other layers including the embedding layer kept frozen, the ℓ-th T-layer and the output layer at each client C_k are trained using the local data D_k. Specifically, the MLM task [Devlin *et al.*, 2019] is performed for a number e_k of epochs (cf. $\text{MLM\_Train}(D_k, P_k, \ell, e_k)$ in Algorithm 1). How to train the ℓ-th T-layer in the small-capacity local model will be presented in Section 3.3.

S3 Each client C_k uploads the weights of the trained ℓ-th T-layer and the output layer to the server.

S4 The server merges the weights uploaded by all clients. There are many approaches for merging the weights in FL, such as FedAvg [McMahan *et al.*, 2017], FedProx [Li *et al.*, 2020a], and FedAdam [Reddi *et al.*, 2020]. In our implementation, we choose the typical approach FedAvg, which obtains the merged weights $W^{(i)}$ at the i-th FL training iteration as follows.

$$W^{(i)} = \frac{1}{N} \sum_{k=1}^{N} W_k^{(i)}, \tag{1}$$

where $W_k^{(i)}$ is the weights uploaded by client C_k at the i-th iteration.

S5 The server updates the global model using the merged weights and determines the next index number ℓ.

S6 Resume to S1 if the current FL training iteration number i does not exceed the user-specified threshold I. Otherwise, we terminate the further pre-training process,

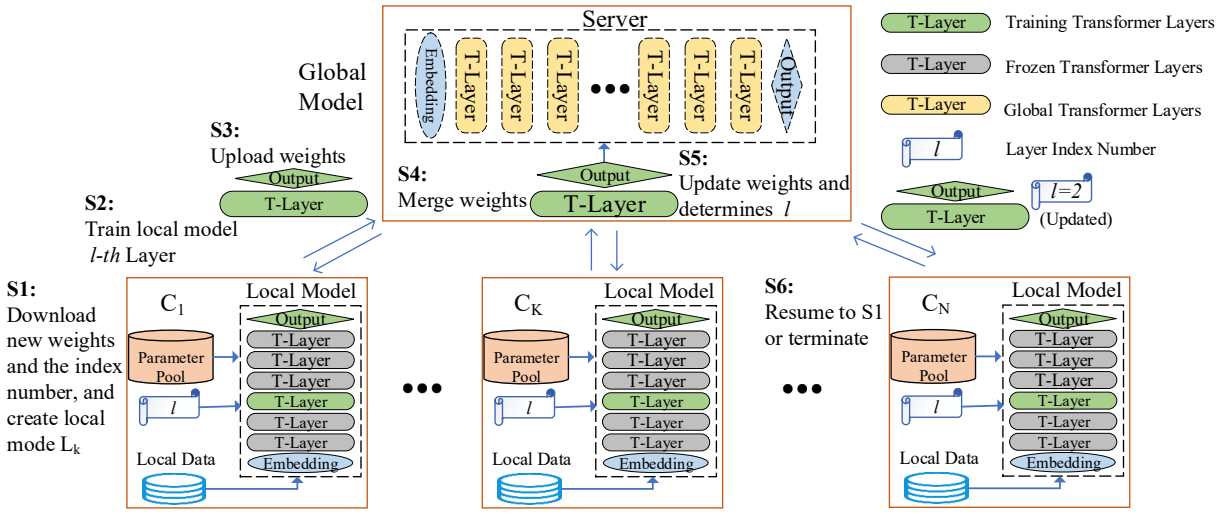


Figure 1: FEDBFPT framework where the FL-based further pre-training process with the current layer index number $\ell = 2$ is depicted. For all local models, the 0-th and 1-st T-layers have been trained and synchronized with the global model at the server.

and consequently, the further pre-trained global model is ready to be fine-tuned for downstream tasks.

To minimize computational expenses for clients with limited resources, we opt to train only a single T-layer and output layer while keeping the remaining parts of the model frozen. However, choosing which T-layer to train (i.e., determining ℓ) and how to efficiently train on smaller capacity local models (cf. step S2) are critical procedures. To resolve these, we propose the Progressive Learning with Sampled Deeper Layers (PL-SDL) method in Section 3.3.

3.3 Progressive Learning with Sampled Deeper Layers (PL-SDL)

As introduced in Section 3.2, a local model uses fewer T-layers than the global model due to the resource constraints of the clients. In this sense, we must train the model parameters more efficiently. This requests us to find and train *partial* T-layers that have a higher potential to boost the global model's final performance. Previous studies [Jawahar *et al.*, 2019; Hao *et al.*, 2019; Manginas *et al.*, 2020; Wang *et al.*, 2022b] have pointed out shallower T-layers of the BERT model excel at capturing phrase-level information, which plays a more important role in the pre-training, compared to those deeper T-layers. Inspired by this, we decide to focus on training the shallower T-layers of the local models and only upload the corresponding weights to the server. More specifically, in each FL iteration, we choose only one specific T-layer [1] and synchronize the training results of the clients with the global model. As outlined in the cost analysis to be discussed in Section 4, this approach effectively reduces both computational and communication costs for individual clients.

[1]It is feasible to train more than one T-layer at an iteration. However, our preliminary experiments show that this does not bring much performance gain but increase the client's computational and communication cost significantly.

In line with Progressive Learning [Wang *et al.*, 2022a], we train the T-layers one by one, starting from the most shallow T-layer and working our way to the deeper T-layers. This approach raises two technical questions: (1) how to determine the index number of the T-layer to be trained in each iteration and (2) how to handle the rest of the T-layers when pre-training the local model with limited capacity. We discuss the questions below with a running example in Figure 2.

Determination of the Layer Index Number. Our preliminary experiments indicate that shallower T-layers are more cost-effective to train. Therefore, for a given number I of FL iterations to train the model, we propose assigning more iterations to shallower T-layers. Specifically, we assign the first $\lceil I/2 \rceil$ FL iterations to train the 0-th T-layer of the local model, the next half of the remaining $(I - \lceil I/2 \rceil)$ FL iterations to train the 1-th T-layer, and so on. Once the FL iterations assigned to the ℓ-th T-layer are completed, the server increments ℓ by 1, with ℓ always being smaller than m_k^L, the number of T-layers in client C_k. This mechanism prioritizes training the most important T-layers in the overall FL process.

Handling the Rest T-layers. Before training the ℓ-th T-layer in the client C_k, the rest $(m_k^L - 1)$ T-layers must be selected from the local parameter pool. In simpler terms, the 0-th to the $(\ell - 1)$-th T-layers are mapped from the corresponding T-layers in the parameter pool. On the other hand, the deeper T-layers, i.e., the $(\ell + 1)$-th to the $(m_k^L - 1)$-th T-layers, are randomly *sampled* from the $(\ell + 1)$-th to the $(m^G - 1)$-th T-layers in the parameter pool; it should be noted that the mapped layer number of a shallower T-layer must be no larger than that of a deeper T-layer. Take the local model training in Figure 2 as an example, suppose $\ell = 2$, $m_k^L = 6$, and $m^G = 12$; the shallower layers, i.e., the 0-th to 2-nd T-layers are directly mapped from the local parameter pool; for the sampled deeper layers from the 3-rd to 5-th T-layers, a possible mappling list could be $[5, 5, 9]$, while $[5, 9, 5]$ is not.

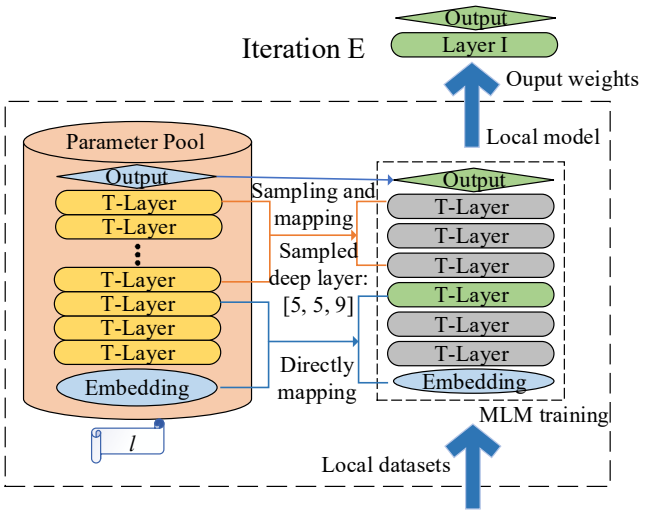


Figure 2: Local model training with the layer index number $\ell = 2$, number of local T-layers $m_k^L = 6$, and number of global T-layers $m^G = 12$. The 2-nd T-layers and those beneath are mapped directly from the T-layers stored in the parameter pool. The deeper T-layer (the 3-rd to 5-th T-layers) are mapped from the sampled T-layers (the 3-rd to 11-th T-layers). The embedding and output layers are also directly mapped from the counterparts stored in the parameter pool.

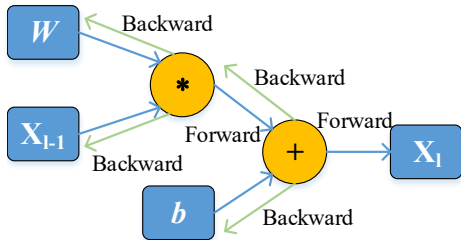


Figure 3: Example of neural network calculations. The yellow circle represents a forward propagation calculation whereas the green line represents a backward calculation to update weights and bias.

4 Cost Analysis

We perform a theoretical analysis of the computational cost and communication cost of FEDBFPT in Section 4.1 and Section 4.2, respectively. To ease the theoretical analysis, the following setups are established:

- To accurately assess the impact of the local model on the global model during FL iterations, we assume that all clients use the same learning rate, batch size, sentence length, and other hyperparameters and train the local models for the same number of epochs e_k locally.
- To accurately assess the effect of adjustments to the local model on the costs at the client, we analyze a specific client. As a result, the size of the local dataset D_k and any effects from external factors such as the hardware specification are excluded.

4.1 Computational Cost

Given a BERT model, let V be the vocabulary size, S the sentence length, H the word vector dimension size, C the number of classifications, and m the number of T-layers. A local BERT model differs from a global BERT model only in the number of T-layers m. We use m^G and m_k^L to differentiate the number of T-layers in the global model and the k-th local model. A previous study [Vaswani *et al.*, 2017] breaks down the computation cost of a BERT model into the following several parts:

- **Embedding** in a time complexity of $\mathcal{O}((V+S)\cdot H)$;
- **Self-attention** in a time complexity of $\mathcal{O}(m\cdot S^2\cdot H^2)$;
- **Feed Forward** in a time complexity of $\mathcal{O}(m\cdot S^2\cdot H^2)$;
- **Add&Norm** in a time complexity of $\mathcal{O}(m\cdot S\cdot H)$.

In training a BERT model, two calculation processes are involved, namely the forward propagation to obtain results, and the backward propagation to update model parameters. An example is illustrated in Figure 3.

Based on the above information, we have the overall time complexity of the model as follows. First, the embedding time cost is in the complexity of $\mathcal{O}((V+S)\cdot H)$. Then, the forward propagation time cost, including Self-attention, Feed Forward, and Add&Norm, is in $\mathcal{O}(mf\cdot(S^2\cdot H^2+S^2\cdot H^2+S\cdot H))\approx\mathcal{O}(mf\cdot S^2\cdot H^2)$ where mf denotes the number of T-layers participating in forward propagation. Third, the backward propagation time cost, similar to the forward counterpart, is in $\mathcal{O}(mb\cdot S^2\cdot H^2)$ where mb refers to the number of T-layers participating in backward propagation. Finally, the overall time cost is in $\mathcal{O}((V+S)\cdot H+mf\cdot S^2\cdot H^2+mb\cdot S^2\cdot H^2)$ and we have $(V+S)\cdot H\ll(mf+mb)\cdot S^2\cdot H^2$.

In FEDBFPT, we have $mf=mb=m^G$ (typically 12) for the global model and $mf=m_k^L$ (typically 6) and $mb=1$ for the local model. Comparing the global model and the local model, it is clear that FEDBFPT saves a lot of computations.

4.2 Communication Cost

Following studies [Vaswani *et al.*, 2017; Devlin *et al.*, 2019], we list the space complexity of the parameter size of different components in a BERT model as follows.

- **Embedding**: $\mathcal{O}((V+S)\cdot H)$;
- **T-layers**: $\mathcal{O}(m\cdot(3\cdot H^2+4H\cdot(2H+1))$;
- **Output**: $\mathcal{O}(H\cdot C)$.

If transmitting the model parameters fully in each iteration, the communication cost will be in the complexity of $\mathcal{O}((V+S)\cdot H+mb\cdot(3\cdot H^2+4H\cdot(2H+1))+H\cdot C)$. Here, we use mb, the number of T-layers in backward propagation, because only these T-layers will be uploaded to the global model in FL. Instead, our proposed FEDBFPT only transmits as few as one T-layer and the output layer to the server for updating. In this sense, a large fraction of parameters to be transmitted are saved for each client given that mb is usually 12 or more for the global model.

5 Experiments

5.1 Setup

We outline the main implementation details of our approach. We use Pytorch 1.11 [Paszke *et al.*, 2019] and an NVIDIA RTX A6000 with 49140 MB Video Memory for training.

The base model is selected as the BERT-base-uncased [Devlin *et al.*, 2019]. Each client performs further pre-training on the base model through the proposed FEDBFPT framework, resulting in a final global model for downstream tasks. To simulate the FL setting, we generate 6 clients and build a local BERT model for each client. The dataset for pre-training is evenly partitioned and then stored in these clients. The uniform and skewed distributions of local datasets are compared in Section 5.4. For comparison use, we also gather the local datasets at the server for centralized further pre-training. The supplementary materials, including the codebase, datasets, and more experiments results are made available online [Wang *et al.*, 2023]. We use the S2ORC dataset [Lo *et al.*, 2020], which contains many general-purpose corpora for NLP and text mining research over scientific papers, for MLM further pre-training, and then fine-tune the global BERT model on classification (CLS) and Named Entity Recognition (NER) tasks of corresponding scientific domains. In particular, we use the JNLPBA dataset [Collier and Kim, 2004] in the Biology domain for NER and the SciERC [Luan *et al.*, 2018] dataset in the Computer Science domain for NER, and the Rct-20k [Beltagy *et al.*, 2019] [2] dataset in the Medicine domain for CLS for fine-tuning. For MLM training, the corpus in the corresponding domain is selected from S2ORC. For a direct comparison with previous studies, **accuracy** and **F1** are considered as the effectiveness metric for CLS and NER, respectively. Corresponding to the costs defined in Section 3.1, we measure the **computational cost** as the time to perform one epoch of MLM training on the S2ORC domain subset with batch size 256 and learning rate 5×10^{-5}, and the **communication cost** as the storage space occupied by the trained parameters per client.

5.2 Overall Comparisons

We include the following baseline methods for comparing with the proposed FEDBFPT:

- **BERT** is the original bert-based-uncased pre-trained model without any further pre-training.
- **BERT-C** is a global model resulting from further pre-training on the gathered dataset on the server side.
- **BERT-FL** is a model resulting from the FL-based further pre-training where all local models are the same as the global model.
- **DistilBERT** [Sanh *et al.*, 2019] is a lightweight model using prediction-based distillation on BERT.
- **TinyBERT** [Jiao *et al.*, 2020] is another lightweight model using layer-wise distillation on BERT. We test both the 6- and 4-layer checkpoints, which are denoted as TinyBERT-6 and TinyBERT-4, respectively.

Table 2 reports the computational and communication cost of a client when using different approaches for BERT further pre-training. In this section, the best and the second best measures in tables are **bold** and underlined, respectively.

After further pre-training, we fine-tune the resultant models on downstream tasks for specific domains. The F1 or accuracy measures are reported in Table 3, while detailed counterparts with increased training epochs are shown in Figure 4.

[2]Rct-20k is very large, and we used 2% of it to fine-tune.

Models	Computational Cost (unit)	Communication Cost (unit: MB)
BERT	1237.37 (-)	N/A
BERT-C	1237.37 (-)	N/A
BERT-FL	1237.37 (-)	417.83 (-)
DistilBERT	1241.18 (100.31%)	255.55 (61.16%)
TinyBERT-6	784.45 (63.40%)	255.57 (61.17%)
TinyBERT-4	**373.59 (30.19%)**	54.88 (13.13%)
FEDBFPT (ours)	577.85 (46.70%)	**29.42 (7.04%)**

Table 2: Costs in Further Pre-Training (The cost ratios compared to BERT-FL are indicated in parentheses)

Models	JNLPBA (F1)	SciERC (F1)	Rct-20k (Accuracy)
BERT	0.7181	0.6223	0.8248
BERT-C	**0.7244**	**0.6499**	0.8121
BERT-FL	0.7224	0.6330	0.8185
DistillBERT	0.6556	0.4891	0.7787
TinyBERT-6	0.7085	0.6089	0.8073
TinyBERT-4	0.6732	0.5664	0.7994
FEDBFPT (ours)	0.7198	0.6421	**0.8312**

Table 3: F1 or Accuracy Measures on Downstream Tasks

According to the experimental results, FEDBFPT achieves comparable effectiveness to BERT-FL [3] while only using **46.70%** of the computational cost and **7.04%** of the communication cost of BERT-FL, an FL-based approach. Our method outperforms other lightweight BERT models and requires fewer resources. We accomplish this by using a smaller model on clients and updating and uploading only one T-layer and the output layer. Additionally, we utilize the PL-SDL method (cf. Section 3.3), which prioritizes training the more important T-layers (i.e., the shallower T-layers) of the global model and helps these shallower T-layers gain more knowledge from the deeper T-layers through sampling and mapping. All these contribute to the decent model performance of FEDBFPT with limited resources.

5.3 Ablation Study

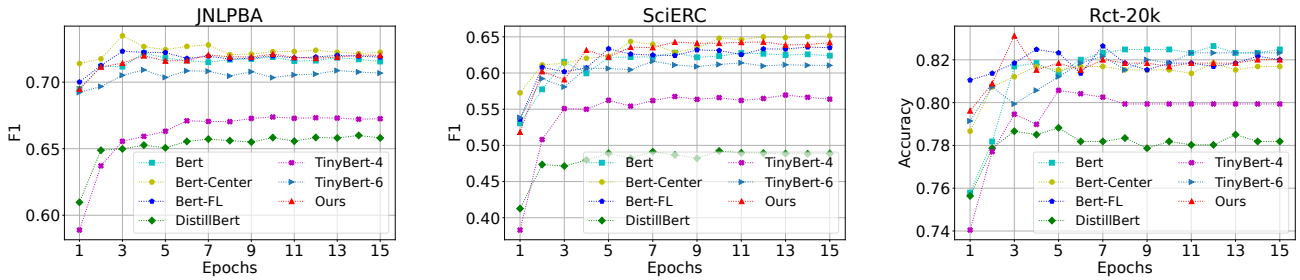
We include the following variants for the ablation study:

- **FEDBFPT-ALL** that always trains all layers of the local model;
- **FEDBFPT\PL** that only trains the first T-layer of the local model without Progressive Learning;
- **FEDBFPT\M** that always uses the first m_k^L T-layers in the parameter pool to build the local model without a mapping process.
- **FEDBFPT\S** that conducts a new deeper layer sampling only when ℓ increments. In contrast, FEDBFPT samples deeper layers in every FL iteration.
- **FEDBFPT\SP** that assigns the i-th iteration to the ℓ-th T-layer ($i \leq m_k^L$) for training.

Using the SciERC dataset in the computer science domain as our case study, we report the costs and F1 in Table 4.

The results disclose that our proposed framework, which utilizes PL-SDL, can efficiently complete further pre-training with a minimal amount of resources. The PL-SDL method

[3]We conducted a t-test via fine-tuning FEDBFPT and BERT-FL for downstream tasks over 10 random seeds. The obtained p-values, i.e., 0.2922 on JNLPBA, 0.4173 on SciERC, and 0.4511 on Rct-20k all pass the t-test, showing that the two methods are comparable.



(a) JNLPBA (Biology) (b) SciERC (CS) (c) Rct-20k (Medicine)

Figure 4: The F1 (on JNLPBA and SciERC) and accuracy (on Ret-20k) measures vs training epochs.

Models	Computational Cost (unit: second)	Communication Cost (unit: MB)	F1
FEDBFPT-ALL	786.86 (63.59%)	255.57 (61.17%)	0.6393
FEDBFPT\PL	**577.85 (46.70%)**	**29.42 (7.04%)**	0.6407
FEDBFPT\M	**577.85 (46.70%)**	**29.42 (7.04%)**	0.6239
FEDBFPT\S	**577.85 (46.70%)**	**29.42 (7.04%)**	0.6038
FEDBFPT\SP	**577.85 (46.70%)**	**29.42 (7.04%)**	0.6418
FEDBFPT (ours)	**577.85 (46.70%)**	**29.42 (7.04%)**	**0.6421**

Table 4: Ablation Study

Models	Computational Cost (unit: second)	Communication Cost (unit: MB)	F1
FEDBFPT-4	**463.87 (37.49%)**	**29.42 (7.04%)**	0.6201
FEDBFPT-8	693.62 (56.05%)	**29.42 (7.04%)**	0.6267
FEDBFPT-6 (ours)	577.85 (46.70%)	**29.42 (7.04%)**	**0.6421**

Table 5: Effect of Varying the Number of Trained T-layers

specifically prioritizes training the shallower T-layers of the local model, allowing them to gain more knowledge from deeper T-layers through sampling and mapping. This is particularly important when working with limited resources, as it ensures that the shallower T-layers receive the necessary information to improve their performance. Additionally, we found that sampling and mapping deeper T-layers on each iteration are crucial for achieving good performance.

5.4 Parameter Study

Effect of the Number of Trained T-layers

We test the effect of varying the number of trained T-layers in the local models and report the cost and F1 measures of the SciERC in Table 5. FEDBFPT-4 and FEDBFPT-8 use local models with 4 and 8 T-layers in the clients, respectively.

The results show that the value of 6 we chose effectively balances the trade-off between overhead and performance. Besides, our approach of focusing more on the shallow layers during the further pre-training phase and utilizing medium numbers of T-layers on the local model is effective in allowing the local model to gain more knowledge.

Effect of Skewed Local Datasets

To create a more realistic simulation, we introduce a normal distribution to determine the sizes of datasets assigned to clients. Each client receives a subset of the centralized dataset according to the proportions specified by the normal distribution. The server then merges parameters by assigning weights that are proportional to the sizes of the clients' data. Importantly, the total sum of the partitioned datasets remains unchanged.

Datasets	JNLPBA (F1)	SciERC (F1)	Rct-20k (Accuracy)
Uniform Datasets	0.7198	0.6421	0.8312
Skewed Datasets	0.7188	0.6309	0.8232

Table 6: Skewed Datasets

The results of FEDBFPT, shown in Table 6, demonstrate that while the uneven distribution may result in some loss of model performance, our method still performs well. We could further improve performance by using different numbers of trained T-layers for different clients in the future. Nevertheless, our method has lower computational and communication costs compared to other models, regardless of the data distribution, as shown in the aforementioned experiments.

6 Conclusion and Future Work

In this study, we have built upon previous research to investigate the role of the shallow layer in federated learning (FL) based BERT further pre-training. To combat the limited computational and communication resources on the client side in FL, we proposed a novel framework, referred to as FEDBFPT, which allows for training a single transformer layer of a global BERT model on clients. Moreover, we proposed the Progressive Learning with Sampled Deeper Layers (PL-SDL) method as a means of effectively and efficiently training the local BERT model with a focus on the shallower layers. Through experiments on a variety of corpora across domains, including biology, computer science, and medicine, we have demonstrated that our proposed FEDBFPT in combination with PL-SDL, is capable of achieving accuracy levels comparable to traditional FL methods while significantly reducing computational and communication costs.

In future work, we plan to implement various-sized local models across clients to handle Non-IID (non-independent and identically distributed) data. We also aim to adapt FEDBFPT for downstream tasks by focusing on specific layers that play a crucial role in fine-tuning specific tasks.

Acknowledgements

This work is supported by the National Key R&D Program of China (No.2022YFB3304100) and by the Zhejiang University-China Zheshang Bank Co., Ltd. Joint Research Center.

References

[Alistarh *et al.*, 2017] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. *NeurIPS*, 30, 2017.

[Beltagy *et al.*, 2019] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *EMNLP-IJCNLP*, pages 3615–3620, 2019.

[Collier and Kim, 2004] Nigel Collier and Jin-Dong Kim. Introduction to the bio-entity recognition task at JNLPBA. In *NLPBA/BioNLP*, pages 73–78, 2004.

[Devlin *et al.*, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[Fu *et al.*, 2020] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript. In *ICML*, pages 3304–3314, 2020.

[Hao *et al.*, 2019] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Visualizing and understanding the effectiveness of bert. In *EMNLP-IJCNLP*, pages 4143–4152, 2019.

[Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[Jawahar *et al.*, 2019] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does bert learn about the structure of language? In *ACL*, 2019.

[Jiao *et al.*, 2020] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. In *EMNLP*, 2020.

[Karras *et al.*, 2018] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

[Konečnỳ *et al.*, 2016] Jakub Konečnỳ, H Brendan McMahan, X Yu Felix, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[Lee *et al.*, 2020] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[Li and Wang, 2019] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

[Li *et al.*, 2020a] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *MLSys*, 2:429–450, 2020.

[Li *et al.*, 2020b] Zhiyuan Li, Jaideep Vitthal Murkute, Prashnna Kumar Gyawali, and Linwei Wang. Progressive learning and disentanglement of hierarchical representations. In *ICLR*, 2020.

[Lin *et al.*, 2018] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. In *ICLR*, 2018.

[Lin *et al.*, 2020] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *NeurIPS*, 33:2351–2363, 2020.

[Liu *et al.*, 2019] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[Lo *et al.*, 2020] Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. S2ORC: The semantic scholar open research corpus. In *ACL*, pages 4969–4983, 2020.

[Luan *et al.*, 2018] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *EMNLP*, pages 3219–3232, 2018.

[Manginas *et al.*, 2020] Nikolaos Manginas, Ilias Chalkidis, and Prodromos Malakasiotis. Layer-wise guided training for BERT: Learning incrementally refined document representations. *arXiv preprint arXiv:2010.05763*, 2020.

[McMahan *et al.*, 2017] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282, 2017.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.

[Peters *et al.*, 2018] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *ACL*, 2018.

[Radford *et al.*, 2018] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[Raghu *et al.*, 2017] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *NeurIPS*, 30, 2017.

[Reddi *et al.*, 2020] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[Sanh *et al.*, 2019] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[Stich *et al.*, 2018] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. *NeurIPS*, 31, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.

[Wang *et al.*, 2018] Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, and Christopher Schroers. A fully progressive approach to single-image super-resolution. In *CVPR Workshops*, pages 864–873, 2018.

[Wang *et al.*, 2022a] Hui-Po Wang, Sebastian Stich, Yang He, and Mario Fritz. ProgFed: Effective, communication, and computation efficient federated learning by progressive training. In *ICML*, pages 23034–23054, 2022.

[Wang *et al.*, 2022b] Jue Wang, Ke Chen, Gang Chen, Lidan Shou, and Julian McAuley. SkipBERT: Efficient inference with shallow layer skipping. In *ACL*, pages 7287–7301, 2022.

[Wang *et al.*, 2023] Xin'ao Wang, Huan Li, Ke Chen, and Lidan Shou. Code, datasets, and supplementary materials. https://github.com/Hanzhouu/FedBFPT, 2023. Accessed: 2023-05-23.

[Wu *et al.*, 2020] Rongliang Wu, Gongjie Zhang, Shijian Lu, and Tao Chen. Cascade EF-GAN: Progressive facial expression editing with local focuses. In *CVPR*, pages 5021–5030, 2020.