Safely Learning with Private Data: A Federated Learning Framework for Large Language Model

Anonymous ACL submission

Abstract

Private data, being larger and quality-higher than public data, can greatly improve large language models (LLM). However, due to privacy concerns, this data is often dispersed in multiple silos, making its secure utilization for LLM training a challenge. Federated learning (FL) is an ideal solution for training models with distributed private data, but traditional frameworks like FedAvg are unsuitable for LLM due to their high computational demands on clients. An alternative, split learning, offloads most training parameters to the server while training embedding and output layers locally, making it more suitable for LLM. Nonetheless, it faces significant challenges in security and efficiency. Firstly, the gradients of embeddings are prone to attacks, leading to potential reverse engineering of private data. Furthermore, the server's limitation to handle only one client's training request at a time hinders parallel training, severely impacting training efficiency. In this paper, we propose a Federated Learning framework for LLM, named FL-GLM, which prevents data leakage caused by both server-side and peer-client attacks while improve the training efficiency. Specifically, we first place the input block and output block on local client to prevent embedding gradient attacks from server. Secondly, we employ key-encryption during client-server communication to prevent reverse engineering attacks from peer-clients. Lastly, we employ optimization methods like client-batching or server-hierarchical, adopting different acceleration methods based on the actual computational capabilities of the server. Experimental results on NLU and generation tasks demonstrate that FL-GLM achieves comparable metrics to centralized chatGLM model, validating the effectiveness of our federated learning framework.

1 Introduction

Existing large language models (LLM) have achieved astonishing results by utilizing vast amounts of public data and massive parameters. In comparison to public data, private data holds advantages in both quantity and quality, because private datasets typically encompass more comprehensive and detailed information about individuals or organizations, and the data production process is more rigorous. Therefore, private data can undoubtedly further enhance the performance of LLM.

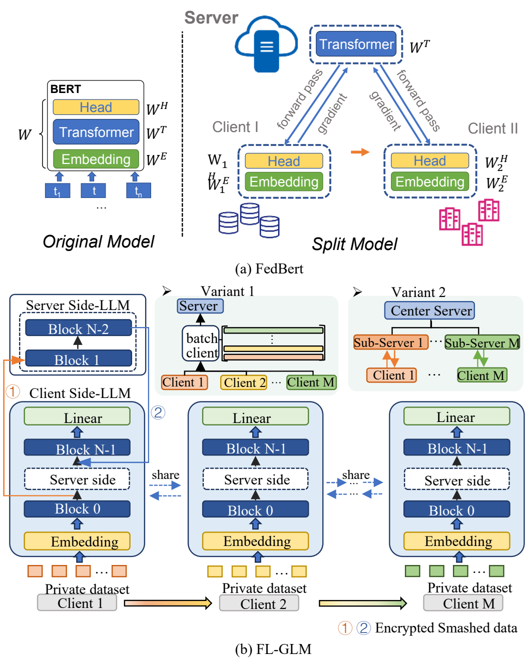

Figure 1: Model architecture of FedBert and FL-GLM.

However, private data is often stored in isolated data silos. For example, mobile users' data is kept locally, involving a significant amount of personal privacy. Considering privacy and security, LLM cannot store private data in a centralized manner for training. Hence, securely leveraging private data for language model training remains a challenging problem. An ideal solution is to utilize the Federated Learning (FL) (Li et al., 2020a) framework, which allows data to be retained on the user device for local training and only passes the model parameters to the server for model aggregation. This approach achieves the goal of keeping the data stationary while making the model updates. By using FL for LLM training, data privacy can be preserved, and the performance of LLM can be further improved.

Unfortunately, traditional FL frameworks, such as FedAvg (Stremmel and Singh, 2021) and FedProx(Li et al., 2020b), are not suitable for LLMs because they require each client to have sufficient computational resources to train the entire LLM. As an alternative method, transformer with split learning, represented by FedBERT (Tian et al., 2022) in Figure 1(a), focuses most of the parameters on the server while continuously training the embedding layer and output layer on the local client, making it more suitable for LLM. The process involves the client using the embedding layer for data input and forwarding it to the server, which then computes and returns the output states. The client calculates loss, sends gradients back for server updates, and receives updated gradients for the embedding layer. However, this method has security risks, as embedding gradients are prone to attacks (Yaldiz et al., 2023) and private data can be reconstructed through beam search(Gupta et al., 2022) or reverse-engineered (Asnani et al., 2023). Additionally, since the server processes one client at a time, it hinders parallel training and reduces efficiency.

In this paper, we propose a FL framework called FL-GLM for LLM, as shown in Figure 1(b). We partition the transformers of the chatGLM into three parts: input and output blocks are stored on the client (shared with peers), while the remaining large parameters are kept on the server. During the training process, the client first performs forward propagation on the input data to obtain hidden states. Then, these hidden states are encrypted with a secure key and sent to the server. Subsequently, the server, either in a client-batch or server-layered approach, receives more client hidden states at a training time and executes forward propagation to send output hidden states back to each client.

It is clear that our FL-GLM framework can effectively prevent data leakage attacks from both servers and peer-clients while enhancing training efficiency. Clients and servers jointly own and utilize the entire model, with certain input and output blocks placed on local clients to thwart embedding gradient attacks from the server. Although sharing input and output blocks between all clients can improve results, interception by peers poses risks, which can be resolved through key encryption during client-server communication. To overcome server capacity limitations, we propose various training acceleration methods. For clusters with multiple machines and GPUs, a hierarchical server architecture initializes sub-servers for parallel client training, with the central server aggregating and distributing models. With single machines and multiple GPUs, the client-batch method concatenates client information for training, enabling parallel execution and enhanced efficiency compared to traditional serial execution in split learning.

Experimental results on NLU and generation tasks demonstrate that FL-GLM achieves performance comparable to centralized chatGLM-6B models, validating the effectiveness of our framework. Further analysis of training costs indicates that our client-batch and server-hierarchical mechanisms can save more than 48% of training time.

The innovations of this paper are as follows:

- To the best of our knowledge, we are the first to design a federated learning framework specifically tailored for LLMs. Starting from user privacy concerns and considering the computational demands of LLMs, we improve split learning to adapt to LLMs, and develop a reasonable, effective, and secure federated LLM framework.

- We propose client-batch and server-hierarchical acceleration optimization methods based on the server's computational capacity to address the issue of low training efficiency in split learning.

- Experimental results on SuperGLUE and abstractive summarization datasets demonstrate that the proposed FL-GLM model can obtain comparable performance to centralized chatGLM models, validating the effectiveness of our FL framework.

2 Related Work

2.1 Federated Learning in LM

Federated Learning (FL) has emerged as a promising approach to train language models (LM) in a decentralized manner while respecting user privacy and data safety. Federated Averaging (FedAvg) (McMahan et al., 2017) is a popular federated optimization algorithm used in language models (Hard et al., 2018; Chen et al., 2019; Stremmel and Singh, 2021). In FedAvg, each client trains its model on locally stored data and communicates updates to the server. The server then performs weighted aggregation of these updates to create a new global model. To reduce local training rounds and accelerate the learning process, Stremmel and Singh (2021) proposes to utilize the pre-trained global models on FedAvg. Ji et al. (2019) proposes Attentive Federated Aggregation (FedAtt) and applies a layer-wise soft attention mechanism to the trained parameters of the neural network model.Previous works (Jalalirad et al., 2019; Thakkar et al., 2020) have integrated DP mechanisms into FedAvg and FedAtt, respectively.

Split learning, represented by SplitFed (Thapa et al., 2022), has emerged as a distributed and collaborative training approach to enable efficient training on resource-constrained devices (Abedi and Khan, 2020; Abuadbba et al., 2020; Rahman et al., 2020; Matsubara and Levorato, 2020), such as mobile devices or small clients without GPU resources. To address sequential data training in language models, FedBERT (Tian et al., 2022) introduces a novel federated learning framework. It splits language model pre-training, easing limited computing resources on client devices. FedBERT segments the BERT model into Embedding, Transformer, and Output layers. It trains the Transformer layer on a powerful server, while less demanding layers (Embedding and Output) train on client devices. However, this setup incurs high communication costs and risks data leakage via embedding gradient attacks.

2.2 Attacks and Defenses

In federated learning, various eavesdroppers threaten client privacy, including servers attempting data recovery and peer-clients intercepting data sent to servers.In NLP, attacks from embedding gradients can easily recover users's private data. Gupta et al. (2022) proposes to infer which words the client used by observing the non-zero values in embedding gradients. They then use beam search and resort to arrange these words, thereby reconstructing private data. To defend against this attack, To counter this, they recommend freezing embedding layers during training. Zhu et al. (2019) briefly mention defending by adding differentially private noise or setting small gradients to zero (gradient clipping). Huang et al. (2020) propose MixUp data augmentation on the BERT model's [CLS] token. Yaldiz et al. (2023) suggest server-side cosine similarity checks on client-uploaded weights to filter out malicious clients. However, these defenses often reduce model accuracy (Yu et al., 2021; Li et al., 2021).

In order to retain the model structure and minimize the performance loss caused by model changes, we propose to move some head layers to the client and use a key-encryption mechanism to protect data privacy during client-server communication. This not only prevents gradient attacks from the server but also prevents information eavesdropping from peers.

3 Model

In this section, we provide the details of our FL-GLM model, as shown in Figure 1(b). FL-GLM consists of three parts: model split, encrypted transmission, and parallel acceleration. Firstly, we split LLM into three parts, saving the first block 0 and the last block N-1 on the local client and placing the remaining parameters on the server. Then, the smashed data is encrypted using keys during client-server transmission. Finally, the server employs either client-batch or hierarchical-server methods to achieve parallel acceleration.

3.1 Model Split

For protecting data privacy, the FL-GLM framework splits LLM into three parts for deployment. During forward operations, private data is processed by the client-side model to produce smashed data, which is then received by the server-side model for computation. Encrypted smashed data ensures data security. Given the input data $x = \{x_1, \ldots, x_L\}$ and the next output y, the smashed data h_0 of the client is defined as:

$$h_0 = Block_0\left(Embedding\left(x\right)\right),$$

where $Block_0$ is the 0^{th} block of LLM, and $Embedding$ is the embedding layer of LLM.

The server-side model contains the 1^{th} to the N-2^{th} blocks of LLM, denoted as $Block_{(1,\text{N-2})}$, which takes the received smashed data h_0 as input, and the hidden state $h_{\text{N-2}}$ as output:

$$h_{\text{N-2}} = Block_{(1,\text{N-2})}(h_0).$$

Then the server send the output $h_{\text{N-2}}$ back to client.

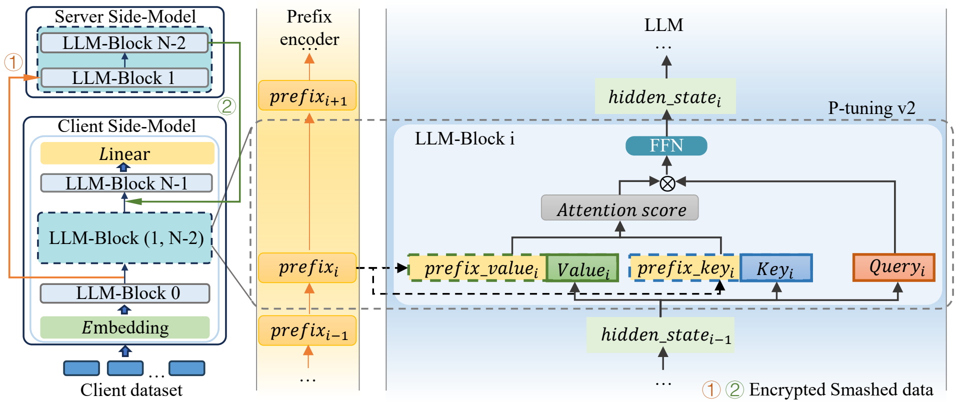

Figure 2: Model Split with p-tuning v2 fine-tuning by training a prefix encoder to adjust LLM-Block outputs.

After the last block N-1 and the linear layer operation on client, the prediction result $y^{'}$ is output and the cross-entropy loss $\mathcal{L}$ is calculated:

$$y^{'} = Linear\left(Block_{\text{N-1}}\left(h_{\text{N-2}}\right)\right),$$

$$\mathcal{L} = Cross\_Entropy\left(y^{'}, y\right),$$

where $Block_{\text{N-1}}$ is the N-1th block of LLM and $Linear$ is the linear layer of LLM. During the whole computation process, the data and data labels are kept in the client to avoid data privacy leakage.

It's important to note that the LLM-Block is constructed from a transformer layer comprising multi-head self-attention mechanisms and a forward network (FFN). With the stacking of LLM-Blocks, large pre-trained models have an extremely high number of parameters, making fine-tuning computationally intensive. To fine-tune large models with limited computational resources, efficient techniques such as p-tuning v2(Liu et al., 2021b) can be employed, as depicted in Figure 2. The FL-GLM framework supports the p-tuning v2 method, wherein all original model parameters are frozen, and the prefix encoder is trained to splice the prefix_key and prefix_value with the key and value of the original model, adjusting the output of each LLM-Block. Further details are available in Appendix A.

3.2 Encrypted Transmission

Since the data features need to flow between the client and the server after the model split, the FL-GLM framework uses a key encryption strategy to complete the encrypted transmission of data. The RSA algorithm generates a pair of public and private keys by factorizing a very large integer. The message is encrypted with the public key and can only be decrypted by the receiver who has the corresponding private key. The RSA key generation process is as follows:

1) Select two large prime numbers, usually denoted as p and q.

2) Calculate their product $n = pq$. n will be used as the common modulus.

3) Compute the Euler function $\phi(n) = (p-1)(q-1)$.

4) Choose an integer e, called the public key exponent, satisfying $1 < e < \phi(n)$, and e and$\phi(n)$ are mutually prime.

5) Compute the private key index d satisfying $d * e \equiv 1 \, (Mod\, \phi(n))$. d is the multiplicative inverse of e to $\phi(n)$.

After the key computation is complete, n and e are disclosed as the public key, where n is the modulus and e is the public key index. Convert the plaintext message M to an integer m with $0 < m < n$. Calculate the ciphertext$C = m^e \, (mod\, n)$. C is the encrypted message. After receiving the ciphertext C, decrypt it using the private key exponent d. Compute the plaintext message $M = C^d \, (Mod\, n)$. m is the original plaintext message.

3.3 Parallel Acceleration

After deploying the large model separately from the client and the server, the server node will bear most of the training cost, and according to the difference in the computing power of the server node, the FL-GLM framework supports two training strategies: serial training and parallel training. If the server node has limited computing resources and can hardly afford a large batch size, serial training is a more suitable choice. As shown in Figure 1(b), during serial training, the server interacts with

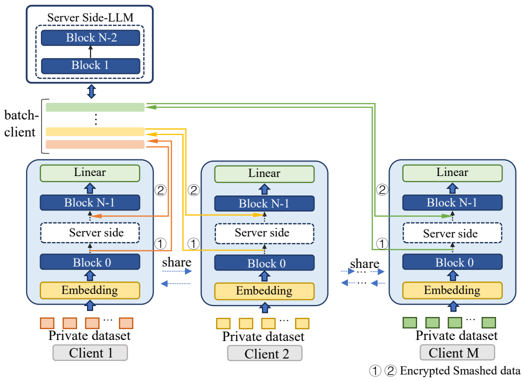

Figure 3: FL-GLM with client-batch parallel training.

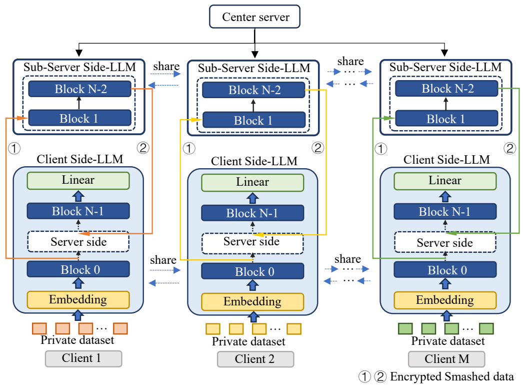

Figure 4: FL-GLM with server-hierarchical parallel.

only one of the clients, and when one client completes the training, the training process for the next client is started. After completing the training, the parameters of multiple client models need to be averaged. Serial training is time-consuming, but one-to-one communication requires less communication, thread processing, and server processing power and is suitable for training scenarios with limited server capacity.

Since the special structure of split learning does not allow smashed data from multiple clients to be averaged, which will result in features and labels not being aligned and a substantial decrease in model performance, two parallel training strategies are designed in the FL-GLM framework. As shown in Figure 3, the first strategy is to stack the smashed data from different clients during parallel training as a set of data to expand the batch for collaborative training. Take clients' batch size=1 as an example; the number of clients is M, and in each round of training, every client sends smashed data of size seqlength, batchsize=1, hiddensize to the server, and the data received by the server will be integrated into a tensor with batch size M for subsequent training. The second parallel strategy is shown in Figure 4. Each client model will correspond to a server-side model, and the server node will run multiple models simultaneously, which can alleviate the threading problem in one-to-many communication to a certain extent. The server-side model parameters and client-side parameters are averaged at the end of the training period.

4 Experiments

In order to demonstrate the performance of chatGLM model within the federated learning framework(FL-GLM), we conduct experiments using the same benchmarks as those used for the GLM model (Du et al., 2022).

4.1 Experimental Settings

We first introduce some empirical settings, including datasets, evaluation metrics, baselines and parameter settings for FL-GLM.

4.1.1 Dataset

For a fair comparison with centralized chatGLM-6B, we test our model on the SuperGLUE (Wang et al., 2019) benchmark for NLU tasks, and on CNN/DailyMail and XSum datasets for abstractive summarization tasks.

The SuperGLUE benchmark is a collection of challenging NLU tasks designed to evaluate the performance and capabilities of state-of-the-art language models. It consists of eight diverse tasks, i.e., ReCoRD, COPA, WSC, RTE, BoolQ, WiC, CB, and MultiRC, each representing a different aspect of language understanding. The details of the SuperGLUE benchmark can be seen in Appendix B. Following GLM (Du et al., 2022), we formulate these tasks as blank infilling tasks. Specifically, given a labeled example (x, y), we rewrite the input x as a closed question $q(x)$ through a mask token [M] and rewrite the output y as an answer $a(y)$.

For abstractive summarization tasks, we append a mask token [M] at the end of the given context as input and treat the summary as output. Then the model generates the summary autoregressively.

4.1.2 Metrics

Since the NLU tasks are reformulated as blank infilling tasks, the model performance can be evaluated using the generated probability of the ground-truth answer $a(y)$. For the RTE, BoolQ, WiC, CB, and MultiRC datasets, the generated answer may contain a single word. Therefore, we compute the

Model	ReCoRD F1/Acc.	COPA Acc.	WSC Acc.	RTE Acc.	BoolQ Acc.	WiC Acc.	CB F1/Acc.	MultiRC F1a/EM	Avg
$T5_{large}$ (Du et al., 2022)	85.7/85.0	78.0	84.6	84.8	84.3	71.6	96.4/98.2	80.9/46.6	81.2
$BART_{Large}$ (Du et al., 2022)	88.3/87.8	60.0	65.4	84.5	84.3	69.0	90.5/92.9	81.8/48.0	76.0
$RoBERTa_{Large}$ (Du et al., 2022)	89.0/88.4	90.0	63.5	87.0	86.1	72.6	96.1/94.6	84.4/52.9	81.5
$GLM_{RoBERTa}$ (Du et al., 2022)	89.6/89.0	82.0	83.7	87.7	84.7	71.2	98.7/98.2	82.4/50.1	82.9
ChatGLM-6B (Zeng et al., 2022)	80.2/78.7	85.0	71.2	81.6	83.4	71.0	85.7/83.9	78.2/45.6	79.6
FL-GLM	79.8/78.4	85.0	71.2	80.1	81.9	69.6	85.7/83.9	79.3/46.1	79.1

Table 1: Results on the SuperGLUE dev set.

Model	CNN/DailyMail			XSum		
	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-1	ROUGE-2	ROUGE-L
BERTSumAbs (Liu and Lapata, 2019)	41.7	19.4	38.8	38.8	16.3	31.2
$UniLMv2_{Base}$ (Bao et al., 2020)	43.2	20.4	40.1	44.0	21.1	36.1
$T5_{Large}$ (Raffel et al., 2020)	42.5	20.7	39.8	40.9	17.3	33.0
$BART_{Large}$ (Lewis et al., 2020)	44.2	21.3	40.9	45.1	22.3	37.3
$GLM_{RoBERTa}$ (Du et al., 2022)	43.8	21.0	40.5	45.5	23.5	37.3
ChatGLM-6B (Zeng et al., 2022)	40.4	17.0	28.0	37.6	12.5	30.1
FL-GLM	39.6	16.9	28.0	37.0	11.9	29.4

Table 2: Results of abstractive summarization on the CNN/DailyMail and XSum test sets.

logit of the corresponding answer token as the evaluation score, defined as:

$$p(y|x) = \frac{p(a(y)|q(x)}{\sum_{y' \in Y} p(a(y')|q(x))},$$

where Y is the ground-truth label set.

For the ReCoRD, COPA, and WSC datasets, the answers may contain multiple words, therefore we compute the sum of the log-probabilities of the answer tokens as the evaluation metrics, which is defined as

$$s(y|x) = \sum_{t=1}^{|L_y|} \text{logP}(y_t|y_1...y_{t-1}, x; \theta).$$

For the summarization task, we use ROUGE-1, ROUGE-2, and ROUGE-L as quantitative metrics, which are widely used in NLP and summary tasks (Liu et al., 2021a; Chen and Yang, 2020; Fang et al., 2022).

4.1.3 Baselines

We apply FL-GLM to ChatGLM-6B model[1], who is an open-source pre-trained language model with 6 billion parameters and building upon the General Language Model(GLM-130B) (Zeng et al., 2022; Du et al., 2022). The reason why we utilize ChatGLM-6B as the basement model is that the ChatGLM-6B model offers a breakthrough scaling property that enables efficient inference on a single RTX 3060 (12G) GPU through INT4 quantization. This property is especially valuable in resource-constrained scenarios, allowing for cost-effective computation on affordable GPUs. Additionally, ChatGLM-6B can leverage low-rank adaptation (LoRA) for fine-tuning on an A100 80G GPU, which results in faster inference times, making it convenient for researchers and developers to use LLMs for various applications.

For a fair comparison with ChatGLM-6B, following GLM, we use 7 baselines, including $T5_{large}$ (Raffel et al., 2020), $BART_{Large}$ (Lewis et al., 2020), $RoBERTa_{Large}$ (Liu et al., 2019), $GLM_{RoBERTa}$ (Du et al., 2022), BERTSumAbs (Liu and Lapata, 2019), $UniLMv2_{Base}$ (Bao et al., 2020) and ChatGLM-6B (Zeng et al., 2022).

4.1.4 Parameter Settings

We utilize the open-source ChatGLM-6B model as the basement model for the FL-GLM model. It has 28-layer transformer blocks, 4096 hidden-size, and 32 self-attention heads. We utilize P-tuning v2 for more efficient fine-tuning on downstream tasks. Experiments are conducted on 2, 3, 5, and 10 clients with NVIDIA A100 GPUs, 40GB RAM per client, and one server with one NVIDIA A100 GPU and 40GB RAM. The keys used in client-server communication are static. In order to make a fair comparison between our FL-GLM model and ChatGLM-6B, the training hyperparameters include a batch size of one, a learning rate of 2e-2 with the Adam optimizer, training epochs, and

[1] https://github.com/THUDM/ChatGLM-6B

Strategy	Centralized	Sequential	client-batch parallel			server-hierarchical			
num. of clients	2	None	2	4	8	2	3	5	10
time(s)	166.4±9.2	175.2±10.1	85.3±4.1	43.0±2.5	22.5±1.7	87.3±4.9	65.5±3.2	34.5±1.9	17.3±0.9

Table 3: Comparison of training time between different training strategies

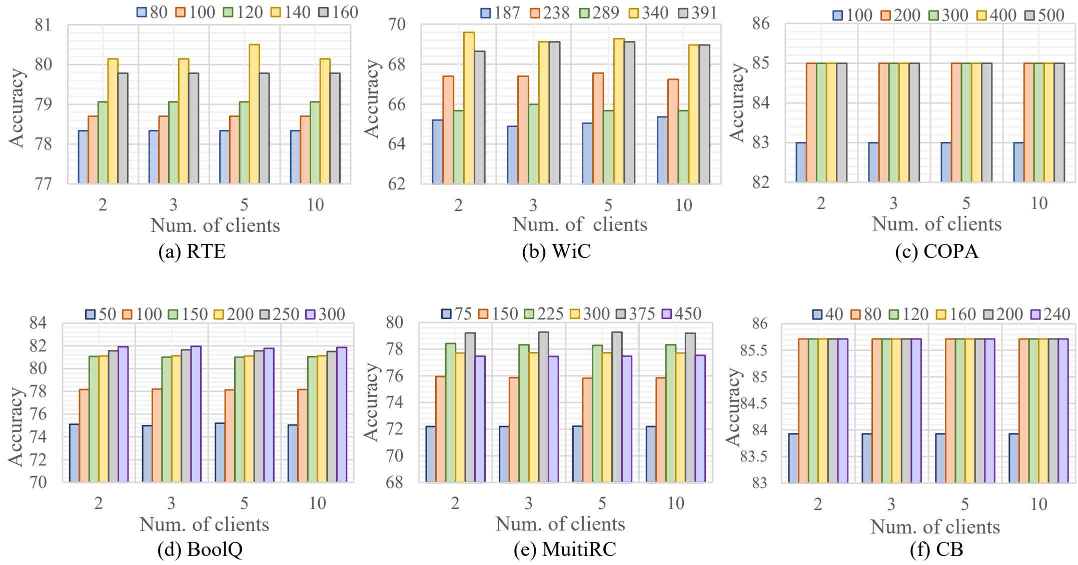

Figure 5: Comparison of model performance under serial training, where colors denote distinct training steps.

maximum sequence length, which are different for different datasets with no warmup and weight decay. The code will be released when this paper is accepted.

4.2 Experimental Results

In this section, we demonstrate our experiment results on SuperGLUE benchmark, CNN/DialyMail and XSum datasets.

4.2.1 Metric-based Evaluation

The quantitative evaluation results on SuperGLUE are shown in Table 1. From the results, we can see that the recent large language models, such as ChatGLM-6B outperform the traditional pre-training models, showing the effectiveness of human-aligned language models for NLU tasks. As a distributed learning pattern, our FL-GLM model performs a little worse than the basement model, ChatGLB-6B. Take the accuracy of the ReCoRD, RTE, BoolQ, and Wic datasets. For example, our FL-GLM model obtains 78.4, 81.6, 81.9, and 69.6, respectively, which is lower than the centralized ChatGLB-6B model in the acceptable range, i.e., 0.3, 1.5, 1.5, and 1.4.From the results on CNN/DialyMail and XSum datasets in Table 2, FL-GLM can obtain 39.6 ROUGE-1, 16.9 ROUGE-2, and 28.0 ROUGE-L on the CNN/DailyMail dataset, 37.0 ROUGE-1, 11.9 ROUGE-2, and 29.4 ROUGE-L on the XSum dataset. Not more than 1.0 lower than the results of the centralized ChatGLM-6B model. In conclusion, our FL-GLM model has the comparable ability to understand language and generate relevant summary with centralized models.

4.3 Analysis

An analysis is conducted including training efficiency and impact of participants.

4.3.1 Training Efficiency

To further investigate the impact of our speedup optimization mechanism on the training cost, we tested the average training duration of the FL-GLM model under three training strategies: serial, client-batch, and server-hierarchical. We randomly selected 1000 data points from the ReCoRD dataset for communication cost analysis experiments. We tested 10 times and took the mean and standard deviation of the total communication time, as shown in Table 3. From the results, we can see that the time consumed in serial training mode with 1000 data points is close to that of centralized training, while parallel training can significantly improve

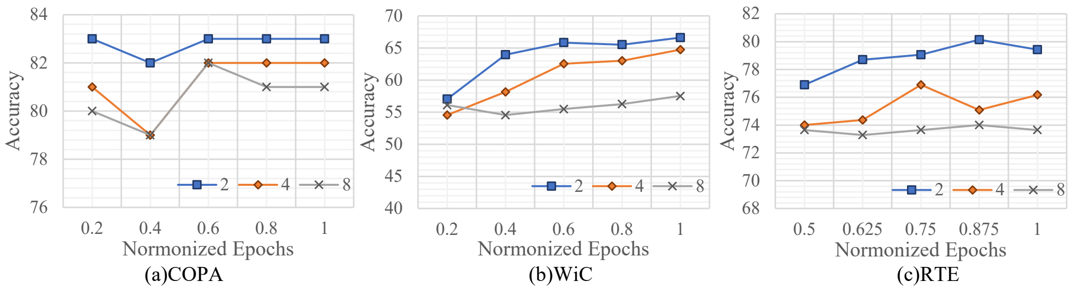

Figure 6: Comparison of the accuracy curves under varying numbers of clients using a client-batch parallel training.

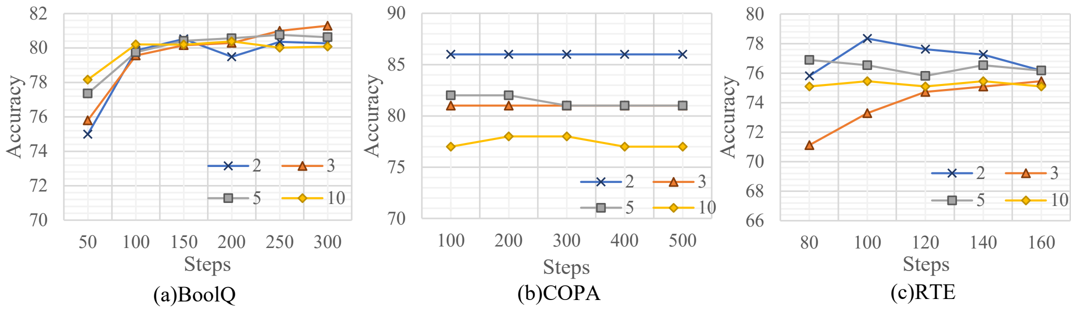

Figure 7: Comparison of the accuracy curves under varying numbers of clients using a server-hierarchical training.

the training time, which is directly proportional to the number of clients.

4.3.2 Impact of Participants

In this section, we test the three training strategies with different numbers of clients by calculating the accuracy scores of FL-GLM on different datasets. [2]

When training in serial, the impact of increasing the number of clients is minimal, as shown in Figure 5. This is because the majority of parameters are trained on the server, making the number of clients insignificant in server-side parameter training.

When training in parallel, the accuracy score of FL-GLM decreases slightly as the number of clients increases, which is more obvious on datasets with smaller data volumes. For client-batch parallel training, as shown in Figure 6, the accuracy score decreases with the increase in the number of clients due to the increase in the batch size, the frequency of model parameter updating decreases, and the server-side model is easy to converge to the saddle point. For hierarchical-server parallel, as shown in Figure 7, the increase in the number of clients makes the amount of data for a single client smaller, so the more the number of clients, the more obvious the overfitting phenomenon is.

[2] In the client-batch parallel test, in order to mitigate the effect of overfitting, the datasets are trained with the same number of training epochs for different numbers of clients, and normalization is used to enhance the visibility of the results.

5 Conclusions

To address the challenge of distributed training of LLMs with limited client computational resources, we propose to utilize the split learning method to segment the generative model. We place the input and output blocks locally on client devices, while the remaining primary model parameters are centralized on a server with ample computational resources. We secure client-server information transfers with encryption methods. To enhance training efficiency, we suggest selecting the client-batch and server-hierarchical acceleration optimization methods based on the server's actual computational capacity, thereby enabling parallel training. This distributed architecture not only ensures that user private data remains on local devices but also effectively reduces the training time, making it more suitable for the scale and complexity of LLMs. In the future, we contemplate employing more advanced privacy-preserving techniques, such as differential privacy, to safeguard the data transmitted from clients, enabling the application of large language models in privacy-sensitive scenarios.

Limitations

FL-GLM underwent evaluation on the SuperGLUE benchmark, CNN/DailyMail, and XSum datasets. While achieving results comparable to those of centralized models, our model falls short in terms of privacy protection. Notably, our encryption method may not fully secure user data in the event of server collusion with malicious clients. Advanced encryption techniques are imperative to bolster user privacy and security. Furthermore, our framework is currently limited to ChatGLM-6B. Future efforts will extend FL-GLM to diverse LLMs, demonstrating its adaptability and broader applicability.

Ethical Considerations

We propose a federated learning framework named FL-GLM, which aims to use private data to train LLM with considerations of prevent data privacy leakage. Our data originates from open-source NLU and NLG projects, adhering to their license limitations and public benchmarks. Moreover, we emulate a distributed data storage environment using open-source datasets, ensuring the exclusion of private data. We affirm our societal contribution without causing harm.

References

Ali Abedi and Shehroz S Khan. 2020. Fedsl: Federated split learning on distributed sequential data in recurrent neural networks. *arXiv preprint arXiv:2011.03180*.

Sharif Abuadbba, Kyuyeon Kim, Minki Kim, Chandra Thapa, Seyit A Camtepe, Yansong Gao, Hyoungshick Kim, and Surya Nepal. 2020. Can we use split learning on 1d cnn models for privacy preserving training? In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 305–318.

Vishal Asnani, Xi Yin, Tal Hassner, and Xiaoming Liu. 2023. Reverse engineering of generative models: Inferring model hyperparameters from generated images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Songhao Piao, Jianfeng Gao, Ming Zhou, et al. 2020. Unilmv2: pseudo-masked language models for unified language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, pages 642–652.

Jiaao Chen and Diyi Yang. 2020. Multi-view sequence-to-sequence models with conversational structure for abstractive dialogue summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4106–4118.

Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Yue Fang, Hainan Zhang, Hongshen Chen, Zhuoye Ding, Bo Long, Yanyan Lan, and Yanquan Zhou. 2022. From spoken dialogue to formal summary: An utterance rewriting for dialogue summarization. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3859–3869.

Samyak Gupta, Yangsibo Huang, Zexuan Zhong, Tianyu Gao, Kai Li, and Danqi Chen. 2022. Recovering private text in federated learning of language models. *Advances in Neural Information Processing Systems*, 35:8130–8143.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

Yangsibo Huang, Zhao Song, Danqi Chen, Kai Li, and Sanjeev Arora. 2020. Texthide: Tackling data privacy in language understanding tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1368–1382.

Amir Jalalirad, Marco Scavuzzo, Catalin Capota, and Michael Sprague. 2019. A simple and efficient federated recommender system. In *Proceedings of the 6th IEEE/ACM international conference on big data computing, applications and technologies*, pages 53–58.

Shaoxiong Ji, Shirui Pan, Guodong Long, Xue Li, Jing Jiang, and Zi Huang. 2019. Learning private neural language modeling with attentive aggregation. In *2019 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. 2020a. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020b. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450.

Xuechen Li, Florian Tramer, Percy Liang, and Tatsunori Hashimoto. 2021. Large language models can be strong differentially private learners. In *International Conference on Learning Representations*.

Junpeng Liu, Yanyan Zou, Hainan Zhang, Hongshen Chen, Zhuoye Ding, Caixia Yuan, and Xiaojie Wang. 2021a. Topic-aware contrastive learning for abstractive dialogue summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1229–1243.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021b. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021c. Gpt understands, too. *arXiv e-prints*, pages arXiv–2103.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740.

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

Y Matsubara and M Levorato. 2020. Neural compression and filtering for edge-assisted real-time object detection in challenged networks. In *IEEE International Conference on Pattern Recognition (IEEE ICPR)*.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. 2020. Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Network*, 34(6):310–317.

Joel Stremmel and Arjun Singh. 2021. Pretraining federated text models for next word prediction. In *Advances in Information and Communication: Proceedings of the 2021 Future of Information and Communication Conference (FICC), Volume 2*, pages 477–488. Springer.

Om Thakkar, Swaroop Ramaswamy, Rajiv Mathews, and Françoise Beaufays. 2020. Understanding unintended memorization in federated learning. *arXiv preprint arXiv:2006.07490*.

Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. 2022. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493.

Yuanyishu Tian, Yao Wan, Lingjuan Lyu, Dezhong Yao, Hai Jin, and Lichao Sun. 2022. Fedbert: When federated learning meets pre-training. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4):1–26.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. Superglue: a stickier benchmark for general-purpose language understanding systems. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 3266–3280.

Duygu Nur Yaldiz, Tuo Zhang, and Salman Avestimehr. 2023. Secure federated learning against model poisoning attacks via client filtering. In *ICLR 2023 Workshop on Backdoor Attacks and Defenses in Machine Learning*.

Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. 2021. Differentially private fine-tuning of language models. In *International Conference on Learning Representations*.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations*.

Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 14774–14784.

Dataset	Task	Cloze Question	Answers
ReCoRD	Question answering	[passage p] [cloze question q]	Answer candidates
COPA	Causal reasoning	“[choice c1]” or“[choice c2]”? [premise p], so [M].	c1/c2
WSC	Coreference resolution	[sentence s] The pronoun ‘*p*’ refers to [M].	Noun n
RTE	Textual entailment	“[hypothesis h]”? [M] “[premise p]”	“yes”/“no”
BoolQ	Question answering	[passage p]. Question: q? Answer:[M].	“yes” / “no”
WiC	Word sense disambiguation	“[sentence s1]”/“[sentence s2]”Similar sense of [word w]? [M].	“yes”/“no”
CB	Textual entailment	“[hypothesis h]”? [M], “[premise p]”	“yes”/“no”/“maybe”
MultiRC	Question answering	[passage p]. Question: q? Is it [answer a]? [M].	“yes”/“no”

Table 4: Cloze questions and answers for the 8 SuperGLUE tasks

A P-tuning v2

P-tuning v2 is proposed based on the p-tuning(Liu et al., 2021c) algorithm, and its basic principle is to add a prompt of length L_p as a learnable embedding, denoted as a prefix, to each LLM-Block's attention operation. Fine-tuning is done by freezing the model parameters and training only the prefix. In each LLM-Block, the corresponding prefix contains two parts: $prefix\_key \in R^{L \times B \times N_h \times d_h}$ and $prefix\_value \in R^{L \times B \times N_h \times d_h}$. Where L is the data length, B denotes batch size, N_h denotes the number of attention heads, and d_h is the dimension of each head.

In the process of forward operation, when the data passes through each LLM-Block, the prefix is spliced with the frozen key and value in the model to form a new key' and value', which are denoted as $K^{'}$ and $V^{'}$, respectively, with the original query parameter (Q) of the model to compute the attention score of the current data as well as the hidden state. Taking the i-th layer LLM-Block as an example, the computation process of p-tuning v2 is shown below:

$$key_i^{'} : K_i^{'} = [prefix\_key_i : key_i]$$

$$value_i^{'} : V_i^{'} = [prefix\_value_i : value_i]$$

$$Attention\ score : S_i^{\prime} = softmax(\frac{Q_i {K_i^{\prime}}^T}{\sqrt{d_h}})$$

$$hidden\_state_i = FFN(S_i^{\prime} V_i^{\prime})$$

B Dataset

Table 4 shows the cloze questions and answers for SuperGLUE tasks, and the detailed corresponding description of SuperGLUE benchmark are as below:

- ReCoRD(Reading Comprehension with Commonsense Reasoning and Disambiguation): In this task, models are required to answer questions by extracting information from a given passage, while also employing commonsense reasoning and resolving ambiguous pronouns.
- COPA(Choice of Plausible Alternatives): This task assesses causal reasoning abilities by providing a premise and two alternative hypotheses, where the model must choose the correct causal relationship.
- WSC(Winograd Schema Challenge): This task evaluates pronoun resolution and coreference resolution abilities, where the model must identify the correct referent for a pronoun in a given sentence.
- RTE(Recognizing Textual Entailment): The task requires determining if one sentence entails, contradicts, or remains neutral with respect to another sentence.
- BoolQ(Boolean Questions): Models must answer boolean questions, i.e., questions that require a yes or no answer, based on a given context.
- WiC(Word-in-Context): In this task, models must determine if a word has the same sense in two different contexts, requiring fine-grained lexical semantics understanding.
- CB(CommitmentBank): It is a famous corpus of short texts for textual entailment task, in which at least one sentence contains an embedded clause.
- MultiRC(Multiple-Choice Reading Comprehension): This task involves answering multiple-choice questions based on multiple passages, which tests the ability to comprehend complex documents.

Datasets	Average Period	Sequential	client-batch parallel	server-hierarchical
COPA	50	85	85	85
	100	85	85	85
WiC	50	69.1	66.6	68.2
	100	69.0	65.5	67.2
RTE	50	80.1	80.1	78.3
	100	79.8	79.4	77.6
BoolQ	50	81.6	79.9	81.0
	100	81.9	80.5	81.3
MultiRC	50	79.3	76.2	77.5
	100	77.5	76.6	77.1
CB	50	85.7	85.7	85.7
	100	85.7	85.7	85.7
WSC	50	71.2	63.5	63.5
	100	66.3	65.4	63.5

Table 5: Impact of different average period

C Impact of Average Period

For analyzing the effect of different averaging periods on the model performance, we tested the performance of FL-GLM with different averaging periods (50 step and 100 step).

The results are shown in Table 5, where the model with an average period of 100 steps slightly outperforms the model with an average period of 50 steps in the BoolQ task. However, in the WiC, RTE, and MultiRC tasks, better results are achieved with an average period of 50 steps. In the COPA and CB tasks, the averaging period has no effect on performance. The most noticeable difference occurs in the WSC task, with scores of 71.2 and 66.3 for an average period of 50 steps and 100 steps, respectively, for serial training, 63.5 and 65.4 for client-batch parallel, and flat accuracy scores for server-hierarchical. Among all the evaluation tasks, the WSC task has the highest sensitivity to the average period, but the average training period has little effect on the overall performance of the FL-GLM model with the same training strategy.