A FAST, PERFORMANT, SECURE DISTRIBUTED TRAINING FRAMEWORK FOR LLM

Wei Huang, Yinggui Wang*† , Anda Cheng, Aihui Zhou, Chaofan Yu, Lei Wang*

Ant Group, China

ABSTRACT

The distributed (federated) LLM is an important method for co-training the domain-specific LLM using siloed data. However, maliciously stealing model parameters and data from the server or client side has become an urgent problem to be solved. In this paper, we propose a secure distributed LLM based on model slicing. In this case, we deploy the Trusted Execution Environment (TEE) on both the client and server side, and put the fine-tuned structure (LoRA or embedding of P-tuning v2) into the TEE. Then, secure communication is executed in the TEE and general environments through lightweight encryption. In order to further reduce the equipment cost as well as increase the model performance and accuracy, we propose a split fine-tuning scheme. In particular, we split the LLM by layers and place the latter layers in a server-side TEE (the client does not need a TEE). We then combine the proposed Sparsification Parameter Fine-tuning (SPF) with the LoRA part to improve the accuracy of the downstream task. Numerous experiments have shown that our method guarantees accuracy while maintaining security.

***Index Terms*—** Distributed LLM, Security, TEE, Lightweight encryption

1. INTRODUCTION

The representative Parameter Efficient Fine-Tuning (PEFT) [1] methods for the LLM include Low-Rank Adaptation (LoRA) [2], Prompt Tuning v2 (P-tuning v2) [3]. The limited amount of domain-specific data available at each institution limits the accuracy of the model. As a result, distributed (federated) LLM [4, 5] has become a direction of great interest. Given the efficient learning and memorization ability of the data by the LLM, it leads to some security issues in the training phase of the distributed LLM: 1) A malicious server will steal parameters and infer data from the model. 2) Malicious clients can also infer data from the rest of the clients through model parameters and intermediate embedding. Previous work considered server-side threats, but did not consider the leakage of parameters and data in the client at the same time [6, 7, 8]. Based on the above considerations, how the distributed LLM can avoid leakage of model parameters and data has become an urgent challenge.

To overcome the above problems, we propose a fast, high-accuracy and secure distributed LLM based on model slicing. It includes various implementations such as LoRA, P-Tuning v2, and the split fine-tuning proposed in this paper. And security is ensured through One Time Pad (OTP) [9] and the TEE [10, 11]. In particular, when training in a distributed manner, we only fine-tune the partial parameters. For different types of TEE, i.e., **SGX (small-memory and no GPU)** [10] and **Intel TDX/SGX (large memory and no GPU)** [11, 10], we propose different schemes in Section 3.1 and 3.2. The main contributions of this paper are as follows. We propose a secure distributed training framework for the LLM based on model slicing to solve the leakage problem of model parameters and data on the server and client side. And we effectively combine the TEE and lightweight encryption to ensure security. Then, we also propose a new split fine-tuning strategy. Experiments demonstrate that our method guarantees high efficiency and accuracy even with security.

2. RELATE WORK

2.1. Security Threats to Federated Learning

Federated learning has been found to face a variety of security threats [6]. One of the threats we consider is the malicious server attack. Wang et al [7], proposed a malicious server attack that utilizes an adversarial generative network of multi-task discriminators to restore typical data for each client. Phong et al [8]. proposed an aggregation method for privacy preservation using Additively Homomorphic Encryption (AHE) to defend against malicious server attacks. Agarwal et al [12], considering the scenario where the client does not trust the server, proposed a distributed Stochastic Gradient Descent (SGD) by utilizing a binary term mechanism to perturb the upload gradient, which is communication efficient and satisfies Differential Privacy (DP). However, DP can only protect the data but not the parameter leakage, and can cause significant performance degradation. Wagh et al [13], proposed a tripartite secure computing framework for neural network training. But MPC will greatly increase training time.

2.2. Trusted Execution Environment (TEE)

A Trusted Execution Environment (TEE) is an isolated hardware enclave that stores and processes sensitive data. Popu-

*These authors contributed equally to this work

† Corresponding anthor (wyinggui@gmail.com).

lar TEE implementations include Intel SGX [10], AMD SEV [14], Intel TDX [11], and TrustZone [15]. In this paper, we follow prior work and deem TEE as a secure area on a potential adversary host device (including GPUs) [16, 17]. It means the data, code, and the whole computation process inside TEEs are secure.

3. METHOD

3.1. Method1: Small memory (Consumer-grade) TEE-shielded LLM Partition

In a distributed scenario, suppose there are K clients. Let $D_k = \{(x^m, y^m)\}_{m=1}^{N}$ $k = 1, ..., K$ denote the training set of the kth client. We use W_k to refer to the parameters that the kth client needs to update. The federated training of Method1 can be seen in Figure 1(a), where only the parameters of LoRA and the embedding of P-Tuning v2 are communicated between the server and clients. We use OTP to encrypt the communicated gradient and decrypt it in TEE. Then at round t, the global model parameters $W^{'}$ can be expressed as $W^{'} = \sum_{k=1}^{K} \frac{n_k}{n} W_k$, where $n = |D| = \sum_{k=1}^{K} n_k$ is the total number of the global joint data. To address the leakage of model parameters and data on the server side, we deploy TEE(SGX) on the server side and do parameter aggregation in it. In training, we fix the parameters except W_k to learn the model parameters θ with the following objective function:

$$\arg\min_{\theta} \left[L(\theta) = \sum_{k=1}^{K} \frac{n_k}{n} L_k(\theta) \right] \tag{1}$$

where L is the loss function (Cross-Entropy).

Note that the current client can maliciously infer the data of the rest clients from the parameters and intermediate embedding of the local model. To avoid the above problems, we deployed a TEE (SGX) on each client and put the model's LoRA and P-Tuning v2 embedding into this device. Since the TEE (SGX) has small memory and is cheap, each client can afford the expense of the device. However, the TEE needs to interact with external computing environments (e.g., on a general GPU). For the output embedding of the TEE carrying a large amount of user information, it also poses a large privacy risk without protecting it. We address the above problem using OTP to encrypt the features transmitted between the GPU and the TEE. Since OTP only supports encrypted computing for linear operations, in addition to the structure that needs fine-tuning, the non-linear layers of the model and the operations are shielded by the TEE, including the layer norm, activation function and softmax. Specific divisions can be seen in Figs. 1(a) and 1(b).

Feature Encryption. For the LLM with a linear layer $h(.)$, we use E to be as the plaintext output of the TEE. Then we generate a random mask r and encrypt the features by the following formula: $E_{en} = E + r$, where E_{en} denotes the encrypted feature.

Feature Decryption. The GPU receives E_{en} , computes $h(E_{en})$, and returns the result to the TEE. The decryption formula in the TEE is $h(E) = h(E_{en}) - h(r)$. Since $Q \cdot K$ (Q and K represent specific linear layers in the attention module) involves matrix multiplication, we run this calculation on the GPU considering that it is more time-consuming for the TEE operations. The decryption formula for matrix multiplication is as follows:

$$\begin{aligned} Q_E \cdot K_E &= Q_E \cdot K_{En} - Q_E \cdot r_k \\ &= Q_{En} \cdot K_{En} - Q_{En} \cdot r_k - r_q \cdot K_{En} + r_q \cdot r_k \end{aligned}, \tag{2}$$

where Q_E and K_E refer to the plaintext matrices of Q and K, Q_{En} and K_{En} refer to the ciphertext matrices of Q and K, and r is the corresponding mask data.

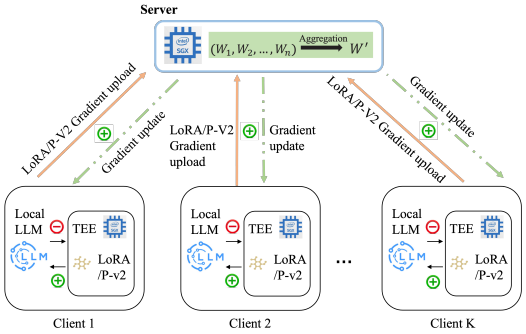
Method1 slices the LLM, deploys the sensitive part of the structure in the TEE, and protects the transmission between the TEE and GPU through OTP to achieve the protection effect of the model parameters and data.

3.2. Method2: Large memory TEE-shielded LLM Partition

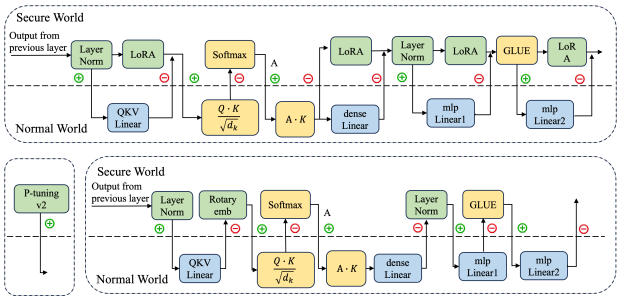
Method1 solves the security problem, but its client generates a higher number of transmissions between the GPU and TEE during training. Secondly, during the process of encryption and decryption, there is a loss of numerical accuracy due to truncation errors, resulting in a slight degradation of accuracy. To alleviate the above problems, we propose a split fine-tuning method in Method2 and apply it to the distributed LLM. The model structure and fine-tuning methodology can be seen in Figure 2.

Method2. The model is first sliced by layer into two parts. Let M_f refer to the first half of the model, M_l the second half of the model, and E_f the output features of the last layer of the first part. For each client, we deploy M_f. For the server, we deploy an TEE (Intel TDX) and put M_l into that device. Since M_l occupies a large amount of memory and the server side is used to provide external services with adequate resources, we use TEE (Intel TDX) as the device for this scheme. In distributed training, each client freezes the parameters of M_f and saves E_f generated for each piece of data. After each client collects the embedding of all the data, it is uploaded to the server's TEE using the OTP encryption. The server's TEE receives the data and does the decryption to obtain E_f, and then each E_f is used as an input for M_l to fine-tune the model.

The strategy for fine-tuning is shown in Fig. 2. In particular, we fine-tune the QKV Linear and dense Linear for each part in M_l, for which we propose the SPF strategy based on the consideration that parameters with larger modulus values play a greater role in the model than parameters with smaller values. Specifically, we use W_{all} to denote the weights of the linear layers and compute the L1 norm for each head on the second dimension of W_{all}. We select the heads by L1 norm value from large to small. After that, we recompose



(a) Distributed LLM training.



(b) Our TEE-shielded LLM Partition for Method1.

Fig. 1. Block diagram of the distributed LLM based on the model partition for Method1, where the green plus sign indicates encryption and the red minus sign indicates decryption.

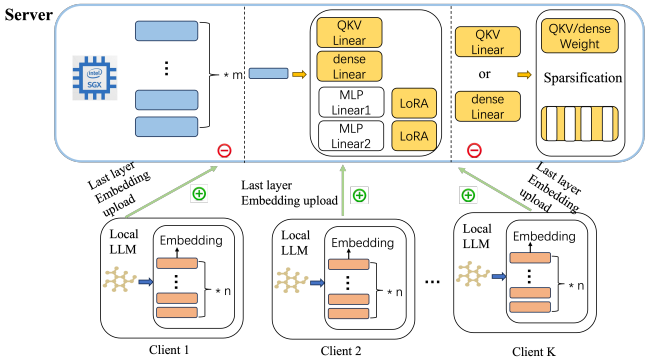


Fig. 2. The distributed LLM training for Method2.

a certain proportion of these heads into the weights denoted as W_{train} (W_{freeze} denotes the weights of the remaining heads). During the fine-tuning process, we freeze W_{freeze} to update W_{train} only. We use X_{train} and X_{freeze} to denote the inputs of the linear layers. The forward formula of the SPF can be expressed as follows:

$$O = (X_{train} \cdot W_{train}^{T} + b_{train}) + (X_{freeze} \cdot W_{freeze}^{T} + b_{freeze}). \quad (3)$$

The SPF method alleviates the problem of requiring a large number of fine-tuning parameters for fully fine-tuning the QKV linear and dense linear of M_l. To further reduce the number of fine-tuning parameters, we choose to freeze the MLP parameters. To ensure performance, we fine-tune LoRA spliced to the MLP layer.

There is no interaction between each client, so there is no need to deploy a TEE. This scheme not only prevents the embedding transmitted from the client to the server from being stolen, but also the gradient does not need to be transmitted back to the client to avoid further information leakage. Since the M_f parameter is frozen and does not have to be updated, each client can generate E_f offline and upload it once. Since the updated model parameters are not available to each client at the end of distributed training, the scheme is served outward by the server to receive queries from users.

4. EXPERIMENTAL EVALUATION

4.1. Datasets and Experimental Configurations

We select five medical datasets as the industry data for this experiment, namely, CHIP-CTC, KUAKE-IR, KUAKE-QIC, KUAKE-QQR, and KUAKE-QTR [18, 19]. We select ChatGLM-6B [20] as the base LLM. For the experimental setup, we set the number of clients to be 3, the number of communication rounds for distributed training to also be 10. And each dataset is divided into three equal parts, which are assigned to different clients. For LoRA, we set the rank to be 8 and lora_dropout to be 0.1. For P-Tuning V2, we set the sequence length to be 128. For Method2, we set the location of the split to be the 24th layer (more details in Section 4.3). For the QKV linear and dense linear, we choose the ratios (12.5%,25%) and (50%,62.5%), respectively. As far as we know, we are the first to propose to prevent both malicious theft of model parameters and data by the server and malicious theft by the client during distributed training of the LLM, and we will compare it with the plaintext (without any security measures) Federal LLM (FL-LLM) and Shielding-Whole-Model by the TEE (SWMT). Table 3 shows the comparison between our two proposed schemes and the remaining schemes.

4.2. Experimental Results of Method1 and Method2

Table 1 shows the accuracy of Method1 and Method2 for the five datasets and the number of parameters that need to be fine-tuned. From the table, we can see that the average accuracy of Method1 on the five datasets is slightly lower than that of FL-LLM, mainly due to the fact that the encryption and decryption of FP16 introduce some truncation errors. Second, the accuracy of Method2 is significantly higher than that of other methods for the five datasets, especially on the

		CHIP-CTC	KUAKE-IR	KUAKE-QIC	KUAKE-QQR	KUAKE-QTR	Number of parameters (10 million)
Method1	LoRA	0.822	0.702	0.815	0.630	0.483	**1.5**
	P-Tuning v2	0.816	0.687	0.811	0.630	0.457	2.9
Method2	4 layers in TEE (25%,50%)	**0.858**	**0.750**	**0.845**	0.633	**0.497**	8.5
FL-LLM	LoRA	0.827	0.730	0.812	0.633	0.473	**1.5**
	P-Tuning v2	0.818	0.720	0.816	0.620	0.453	2.9
SWMT	LoRA	0.825	0.730	0.811	**0.635**	0.474	**1.5**
	P-Tuning v2	0.820	0.724	0.813	0.622	0.451	2.9

Table 1. Comparison of the accuracy of different methods on datasets and the amount of parameters that need to be fine-tuned.

		The ratio	CHIP-CTC	KUAKE-IR	KUAKE-QIC	KUAKE-QQR	KUAKE-QTR	Number of parameters (10 million)
Method2	2 layers in TEE	100% , 100%	0.858	0.741	0.839	0.623	0.503	13.5
	4 layers in TEE	100% , 100%	0.867	0.733	**0.858**	0.657	0.540	27.0
	8 layers in TEE	100% , 100%	**0.884**	**0.770**	0.852	**0.677**	**0.587**	54.0
Method2	4 layers in TEE	12.5% , 100%	0.849	0.723	0.827	0.633	**0.543**	9.4
	4 layers in TEE	25% , 100%	0.862	**0.753**	0.842	**0.650**	0.503	11.8
	4 layers in TEE	12.5% , 50%	0.836	0.723	0.824	0.620	0.490	**6.0**
	4 layers in TEE	12.5% , 62.5%	0.847	0.727	0.821	0.627	0.511	6.9
	4 layers in TEE	25% , 50%	0.858	0.750	0.845	0.633	0.497	8.5
	4 layers in TEE	25% , 62.5%	**0.865**	0.743	**0.848**	0.637	0.510	9.3

Table 2. Comparison of model accuracy and parameter quantities after LLM is divided into different layers using Method2.

three datasets of CHIP-CTC, KUAKE-IR, and KUAKE-QIC, which indicates that the split fine-tuning strategy of Method2 can effectively improve the accuracy of the downstream tasks. However, the number of fine-tuned parameters in Method2 is about 5 times as many as that of Method1. Table 4 shows the training and inference time for different methods, where the training time is the average forward and backward computation time for one piece of data from the client. We can conclude that SWMT has the longest training and inference time mainly since the TEE has only CPU processors. For Method1, it shows that the encryption-state operations and data transmission between the TEE and GPU incur a large time overhead. Since only the last m layers of operations in the TEE, Method2 has a shorter training time. Though FL-LLM has a small latency, it cannot guarantee the security of the model parameters and the data.

4.3. Ablation Experiment for Method2

Table 2 demonstrates the model accuracy and the number of parameters for Method2, segmented by different layers and the selected ratios. As we can see, the more layers deployed in the server or the higher percentage selected, the larger the number of parameters that need to be fine-tuned, which in turn increases the training and inference latency, but brings some performance improvement. In order to balance the accuracy, time cost, and number of parameters, we choose to deploy 4 layers on the server side and the (25%, 50%) ratio of parameters to be fine-tuned for comparative experiments in Tab. 1.

	Method1	Method2	FL-LLM	SWMT
TEE in the client side	✓	✗	✗	✓
TEE in the server side	✓	✓	✗	✓
The type of TEE	SGX	TDX/SGX	-	TDX/SGX
Secure training	✓	✓	✗	✓

Table 3. Comparison of different schemes, where the memory of SGX is small and that of Intel TDX/SGX is large.

		Training time (s)	Inference time (s)
Method1	LoRA	17.06	2.43
	P-Tuning v2	11.76	1.96
Method2	4 layers in TEE (25%,50%)	4.33	0.73
FL-LLM	LoRA	0.48	0.11
	P-Tuning v2	0.36	0.10
SWMT	LoRA	17.92	3.14
	P-Tuning v2	17.60	3.03

Table 4. Comparison of average training and inference time for a piece of data in different methods.

5. CONCLUSION

In this paper, we propose a secure distributed training framework for LLM based on model slicing to prevent model parameters and data from being maliciously stolen by the server and client. We use model slicing and combine it with the TEE and lightweight encryption strategies to achieve both security and almost lossless model accuracy. We also propose a split fine-tuning scheme to achieve the goals of fast, safety, and high precision. Users can choose between consumer-grade small memory TEE (Method1) and industrial-grade large memory TEE solutions (Method2) in combination with cost consideration.

6. REFERENCES

[1] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul, "Peft: State-of-the-art parameter-efficient fine-tuning methods," https://github.com/huggingface/peft, 2022.

[2] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

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

[4] Weirui Kuang, Bingchen Qian, Zitao Li, Daoyuan Chen, Dawei Gao, Xuchen Pan, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou, "Federatedscope-llm: A comprehensive package for fine-tuning large language models in federated learning," 2023.

[5] Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Guoyin Wang, and Yiran Chen, "Towards building the federated gpt: Federated instruction tuning," *arXiv preprint arXiv:2305.05644*, 2023.

[6] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[7] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 2512–2520.

[8] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018.

[9] Florian Tramer and Dan Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[10] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas, "Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, pp. 1–9. 2016.

[11] "Intel. intel trust domain extensions," https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html.

[12] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan, "cpsgd: Communication-efficient and differentially-private distributed sgd," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[13] Sameer Wagh, Divya Gupta, and Nishanth Chandran, "Securenn: 3-party secure computation for neural network training.," *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[14] David Kaplan, Jeremy Powell, and Tom Woller, "Amd memory encryption," *White paper*, p. 13, 2016.

[15] Tiago Alves, "Trustzone: Integrated hardware and software security," *Information Quarterly*, vol. 3, pp. 18–24, 2004.

[16] Jiahui Hou, Huiqi Liu, Yunxin Liu, Yu Wang, Peng-Jun Wan, and Xiang-Yang Li, "Model protection: Real-time privacy-preserving inference service for model privacy at the edge," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4270–4284, 2021.

[17] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang, "Model stealing attacks against inductive graph neural networks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1175–1192.

[18] Ningyu Zhang, Mosha Chen, Zhen Bi, Xiaozhuan Liang, Lei Li, Xin Shang, Kangping Yin, Chuanqi Tan, Jian Xu, Fei Huang, et al., "Cblue: A chinese biomedical language understanding evaluation benchmark," *arXiv preprint arXiv:2106.08087*, 2021.

[19] "Promptcblue," https://github.com/michael-wzhu/PromptCBLUE, 2023.

[20] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang, "GLM-130b: An open bilingual pre-trained model," in *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.