LLM Optimization Without Data Sharing: A Split Learning Paradigm

Omar Basubeit
Lucidya
Riyadh, Saudi Arabia
ogumaan@lucidya.com

Abdulaziz Alenazi
Lucidya
Riyadh, Saudi Arabia

Boris Radovič
KAUST and *University of Ljubljana*
Thuwal, Saudi Arabia
Ljubljana, Slovenia

Ameera Milibari
Lucidya
Riyadh, Saudi Arabia

Marco Canini
KAUST
Thuwal, Saudi Arabia

Zuhair Khayyat
Lucidya
Riyadh, Saudi Arabia

***Abstract*—Resource-constrained organizations with vast datasets often face significant challenges in training and fine-tuning large language models (LLMs) due to privacy and compliance requirements. These organizations cannot share their sensitive data but still require external computational resources for model creation. In this paper, we investigate the effectiveness of split learning (SL) techniques for fine-tuning LLMs while maintaining data privacy. We compare three strategies: a centralized approach, where all fine-tuning occurs on a centralized server, and two split learning strategies (plain and U-shaped), where fine-tuning is distributed between a client and a cloud server, ensuring the client retains control over sensitive data. The U-shaped strategy enhances the plain split learning approach by ensuring labels are not shared with the server, offering stronger privacy protection. Our experiments empirically evaluate these methods, comparing model quality and training efficiency. The results show that all three strategies achieve comparable model performance, evaluated using the loss metric. Although the U-shaped SL strategy incurs higher network transmission costs, it provides enhanced privacy guarantees. This presents a trade-off between privacy and communication efficiency, making U-shaped SL a strong candidate for privacy-preserving LLM fine-tuning in resource-constrained environments.**

***Index Terms*—LLMs, Split Learning, Fine-tuning, Privacy.**

I. Introduction

Large Language Models (LLMs) are increasingly gaining traction, finding applications across diverse domains, including programming, biomedicine [1], and question answering [2]. With model sizes often surpassing several billion parameters, successful LLMs are typically trained in data centers equipped with specialized hardware, requiring significant investments in terms of finances, energy consumption, and time [3]. Given their scale, these models necessitate vast datasets for training; thus, most LLMs leverage extensive publicly available data sourced from heterogeneous domains, such as web-scraped content. This diversity enables them to function as versatile problem solvers across various tasks, including powering AI-driven interactions in digital ecosystems such as the Metaverse.

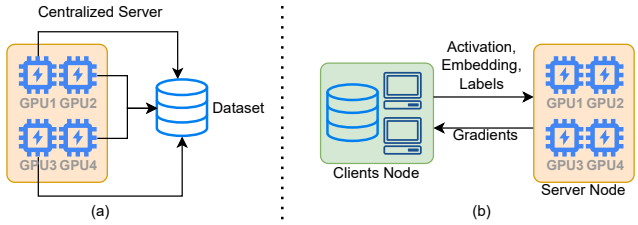

Fig. 1. High level architecture of (a) centralized training vs (b) SL training.

Tailoring an LLM using domain-specific private data—the focus of our work—promises to enhance performance for specialized applications. Many organizations, for instance, depend on models trained on proprietary customer data as a cornerstone of their business, requiring a model specifically adapted to their private datasets. This is particularly relevant in customer experience management and personalized virtual experiences, where organizations seek to fine-tune LLMs on sensitive user interactions to improve engagement, recommendation accuracy, and real-time responsiveness. In Metaverse environments, where AI plays a critical role in generating dynamic content, moderating interactions, and personalizing user experiences, the need for adaptive and privacy-preserving fine-tuning is even more pronounced. However, training such models in-house is often infeasible for Small and Medium-sized Enterprises (SMEs) due to prohibitively high computational costs. A company managing terabytes of data, for instance, would need significant on-premise resources to train a model on its dataset, but setting up such an infrastructure requires substantial upfront investment and may sometimes be infeasible. Fine-tuning pre-trained models, particularly with parameter-efficient strategies [4]–[6], reduces the training investment compared to training from scratch. However, the computational cost remains substantial, especially when models require frequent updates to adapt to shifting data distributions.

On the other hand, companies may decide to rent cloud computing resources to train on their data. However, transferring private data to third-party cloud providers in order to use traditional *centralized learning* approaches raises concerns, as private datasets such as customer experience data often contains personally identifiable information (PII) and, in some cases, cannot be outsourced to external entities due to privacy, legal, and compliance restrictions [7]. This challenge is particularly evident under regulations like the Saudi Personal Data Protection Law (PDPL), which imposes strict data localization and processing limitations, often exceeding the restrictions found in GDPR, thereby placing additional constraints on AI companies operating in the region.

To sump up, training[1] very large models remains impractical for SMEs due to the high memory and compute requirements. Consequently, there is a pressing need for methods that can leverage both local infrastructure and cloud computing resources *without* exposing raw data to the cloud. In this paper, we address this gap by empirically analyzing split learning (SL) – a state-of-the-art approach that enables model training on private data while leveraging the high compute resources of a centralized server – as a privacy-preserving alternative to traditional centralized learning, specifically for training LLMs. Fig. 1 contrasts these approaches.

SL divides the burden of model training between a data-owning and resource-constrained client – in our case, the company possessing vast datasets – and a compute-powerful server. Thus, SL is expected to accelerate model training by leveraging cloud resources while preserving data privacy, effectively addressing both restrictions mentioned earlier. However, its practical performance for training LLMs remains unclear, especially given the lack of comprehensive studies on LLM training with SL in the current literature, motivating our work. In our analysis, we consider both plain SL, where the client must disclose target labels to the server, and U-shaped SL, which eliminates this requirement, thus further enhancing client data privacy.

Our main contributions therefore are:

- We conduct comprehensive experiments using centralized, plain SL, and U-shaped SL approaches, offering valuable insights and analysis to guide practitioners in applying these methods to real-world scenarios.
- We explore optimizations to enhance the performance of training LLM with SL, in terms of resource efficiency and privacy enhancement.

II. Related Work

SL was initially proposed to address the growing computational demands of deep neural networks by enabling data-owning clients to offload computation to a powerful server, with the topmost layers placed on the server [8]. However, this formulation required clients to send target labels to the server. The introduction of the U-shaped architecture resolved this issue by placing both the lowermost and topmost layers on the client [9]–[12]. In this approach, the client computes the loss locally, eliminating the need to share target labels with the server. Regardless of the specific architecture – U-shaped or Plain SL – training proceeds via standard backpropagation, with devices exchanging activation maps during the forward pass and gradients during the backward pass. Early SL formulations involved only one client training the model at a time. However, this approach proved inefficient because the server remains idle during the client's local computation and data communication. This limitation led to the development of parallel SL (PSL) approaches, where multiple clients train a model simultaneously using the same server [13]–[16]. While PSL mitigates server idle time, it is beneficial only when *multiple* clients collaborate in training.

Other work in the PSL domain has tackled other challenges, such as high network overhead [17]–[19], server-client update imbalance [20], [21], heterogeneous model partitioning [22], [23], and security aspects of the training process [24].

In the domain of LLMs, SplitLoRA [25] introduces the first SL LLM fine-tuning framework, combining the advantages of Federated Learning's (FL) parallel training with SL's model partitioning, addressing the computational and communication challenges of fine-tuning LLMs on distributed private data. However, authors in [25] do not provide an analysis of how varying layer placements affect validation loss, training behavior, and GPU utilization.

From an application perspective, SL has been successfully applied across various domains, including healthcare [10] and assisting IoT devices during training [18].

III. Methodology

We start by contrasting the centralized approach with two SL variants, namely Plain SL and U-shaped SL.

A. Centralized Approach

As shown in Figure 1, in centralized training, all participating nodes – e.g., compute nodes in a data center – have full access to the entire training dataset [26]. Training can occur on a single machine or make use of nodes using techniques such as data parallelism (where different nodes process separate batches of the same dataset), tensor parallelism (where model parameters are split across nodes), or pipeline parallelism (where different model layers are assigned to different nodes and micro-batches of training data are pipelined to parallelize resource utilization). As shown in Figure 1, in centralized training, all participating nodes – e.g., compute nodes in a data center – have full access to the entire training dataset [26]. Training can occur on a single machine or make use of nodes using techniques such as data parallelism (where different nodes process separate batches of the same dataset), tensor parallelism (where model parameters are split across nodes), or pipeline parallelism (where different model layers are assigned to different nodes and micro-batches of training data are pipelined to parallelize computation).

[1]We broadly refer to LLM training as the process of updating model parameters using an organization's private data, whether this involves fine-tuning a pre-trained LLM or (pre-)training an LLM from scratch.

Regardless of the number of nodes involved, the key characteristic of centralized fine-tuning is that all computations happen within a controlled environment where data remains fully accessible to all training nodes. This approach simplifies debugging and management but completely exposes training data, thus potentially introducing privacy and data governance issues.

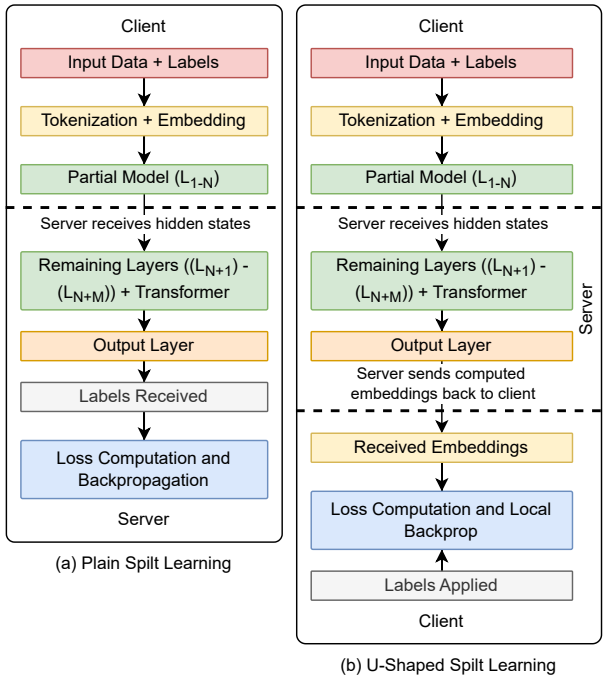

Fig. 2. Comparison of Plain SL and U-Shaped SL. L_N represents the last layer processed on the client side, while L_{N+1} to L_{N+M} indicate the layers handled by the server.

B. Plain SL Approach

SL is a decentralized training method in which a model is partitioned into separate parts between a resource-constrained client and a compute-powerful server [9]. In the plain configuration, the model is divided so that the initial layers – e.g., in the case of LLMs, the embedding table and the initial transformer layers – are trained on the client, while the rest on the server. Training then proceeds by partitioning the computational graph between the two entities. More in detail, in the case of LLMs, the workflow of the Plain SL approach is the following:

1) The client samples a batch of data and performs a forward pass through its assigned model segment.
2) The client sends the necessary data to the server to execute its portion of the computational graph. Typically, this includes the activation maps from the forward pass and the target labels, which the server uses to compute the loss. In the case of LLM training, where fixed-sized token sequences are used but some tokens serve as padding, the client must also send an attention mask to indicate which tokens should be attended.
3) The server receives the data from the client and completes the forward pass on its model segment. It then computes the loss and backpropagates it until obtaining the partial derivatives with respect to the embeddings received from the client. Finally, the server sends these gradient updates back to the client.
4) The client receives the server's gradients and backpropagates them through its model, updating its weights accordingly.

Thus, in Plain SL, as illustrated in Figure 2, the server assists the client during training by handling a compute-intensive segment of the computational graph. This approach offers several benefits, namely that shifting computation does not alter the computation itself, and it significantly reduces the client's computational burden. Importantly, in SL, the server receives only intermediate activations rather than raw data, enabling privacy-preserving collaboration across distributed data sources while ensuring that raw data remains on the client.

In this work, we extend SL to a real-world fine-tuning scenario by integrating Low-Rank Adaptation (LoRA) [5] for efficient adaptation of LLMs. We implement LoRA within SL by applying rank decomposition to reduce the number of trainable parameters, thereby further reducing memory requirements on the client side. This integration allows fine-tuning LLMs under practical hardware constraints while maintaining computational feasibility. Furthermore, we evaluate different SL configurations, varying the number of layers placed on the client side, to analyze their impact on memory efficiency and performance.

C. U-Shaped SL Approach

One limitation of Plain SL as discussed above is that it requires the client to share the target labels with the server, possibly exposing sensitive information. This approach is thus appropriate only when labels are not highly sensitive. In contrast, U-Shaped SL offers a more secure approach by wrapping the network around at the server's end layers. In this configuration, the client handles the final output and generates gradients from the server's end layers. Crucially, no label sharing is required between the server and clients, and raw data is never exchanged [27]. This structure significantly reduces the exposure of sensitive information during both forward and backward propagation, making U-Shaped SL a more privacy-preserving alternative.

To enhance adaptability and safeguard sensitive data, we introduce client-side hidden layers both after the embedding layer and before the LLM head. The embedding layer itself, residing on the client side, generates the first textual representations (embeddings), while adding more hidden layers after the embeddings increases the complexity of exchanged features, thus enhancing training security. This structured placement ensures that raw data processing remains confined to the client, minimizing exposure to the server and reducing the risk of data leakage. During training, both activations and gradients

are exchanged between the client and server in a two-way process. The U-shaped SL architecture leverages this design by mirroring the number of client-side layers on both ends of the model. For instance, when the client retains two layers, one set is positioned after the embedding layer, and another set is placed before the LLM head, while the server handles the remaining layers. This approach maintains the integrity of contextual transformations while protecting sensitive input and output representations. By keeping terminal attention layers at the client side, we ensure controlled and secure processing without compromising the model's generative capabilities. In our configuration, the LLM head lacks trainable parameters as a result of the LoRA setup, which eliminates the risk of gradient leakage at the output stage.

IV. Experimental Design

To conduct our experiments, we utilize a leading cloud hosting provider to run the central node in the centralized setup, while in the SL setup, the server node is also hosted in the cloud, and the client nodes operate on local infrastructure. The network latency between client and server nodes in SL is approximately 0.394 ms, with a bandwidth of 4.97 Gbps, ensuring stable communication for model training.

A. Model

Throughout our experiments we use the 13B version of AceGPT [28], a state-of-the-art language model specifically designed to handle Arabic text, focusing on understanding and generating language that aligns with the cultural and linguistic nuances of the Arabic-speaking world.

AceGPT is configured using Parameter-Efficient Fine-Tuning (PEFT) techniques, specifically through the LoRA method [5]. This approach involves adding lightweight adapters to the original model layers, thereby reducing computational load and training time. The final trainable parameters constitute approximately 0.397% of the total model parameters.

The fine-tuning process is conducted with a batch size of 8, a sequence length of 200, and a hidden size of 5,210, optimizing the model's performance while balancing computational efficiency. The experimental setup utilizes the PyTorch framework with the AdamW optimizer, a learning rate of 5×10^{-4}, a weight decay of 0.01, and computations performed in half precision.

B. Data

The dataset used in this study consists of Arabic tweet-reply pairs from X (formerly, Twitter) for fine-tuning an LLM focused on customer care interactions. This real-world conversational data helps the model learn interaction nuances but presents challenges such as diverse dialects, code-switching, informal language, spelling variations, emojis, abbreviations, short-text limitations, and limited labeled data. We extracted 10 million Arabic tweet-reply pairs and applied extensive cleaning by removing non-Arabic content, English characters, irrelevant symbols (except question marks), short numeric values (2 to 7 digits), excessive character repetitions, duplicates, and records under 17 characters, reducing the dataset to 3.8 million pairs. A logistic regression classifier, trained on 1,000 labeled customer care examples and 1,000 noisy tweets with an 86% accuracy, was applied to this dataset, identifying 2.5 million tweet-reply pairs relevant to customer care.

C. Implementation

We implement all SL algorithms using the SplitBud framework [29]. The server machine is equipped with 4 NVIDIA A10G GPUs (24 GB RAM each), a 48 vCPU AMD EPYC processor, and 192 GB DDR4 RAM, while the client machine features an NVIDIA Tesla T4 GPU (16 GB RAM), 4 vCPUs Intel Xeon processor, and 16 GB GDDR6 RAM and they communicate via bidirectional gRPC streams. The client establishes a gRPC connection with the server and uses it throughout training to exchange – depending on the SL configuration – activation maps, gradients, or target labels.

In our implementation, the process starts by initializing the server, which holds the majority of the pre-trained LLM model. The client is then initialized with only a few layers of the model (specified below). The training process involves splitting the dataset into batches, with the client sending activation maps after computing the forward pass. The server computes gradients and sends them back to the client, which updates its model using the received gradients.

V. Results and Discussion

The evaluation focuses on training loss behavior, validation loss, training time, GPU utilization, FLOPs, and computing and communication costs.

In the following graphs, each SL strategy is labeled with a number to indicate the number of transformer layers processed on the client before offloading to the server. In "Plain - 0" or "U-shaped - 0", the client handles only the embedding layer, while in "Plain - 1" or "U-shaped - 1", it processes one transformer layer before offloading, and in "Plain - 2" or "U-shaped - 2", it processes two transformer layers before offloading computation to the server. The centralized strategy serves as a baseline, where all computations occur on the server without any client-side processing.

A. Training Loss Behavior

Fig. 3 shows the training behavior of the central, plain, and U-shaped SL strategies, demonstrating that all three approaches exhibit nearly identical training dynamics. Despite differences in model partitioning, the validation loss curves follow the same trend, converging at similar rates and reaching comparable final values. This empirically demonstrates that SL – whether in a plain or U-shaped configuration – does not introduce significant divergence from centralized fine-tuning in terms of training behavior, reaffirming that model updates remain consistent across all strategies. The minor differences between strategies stem from the lack of synchronization between the client and server's random number generators,

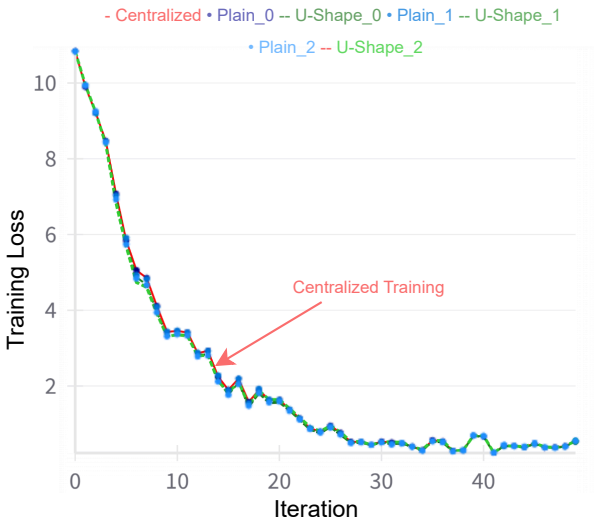

Fig. 3. Training loss behavior for different strategies.

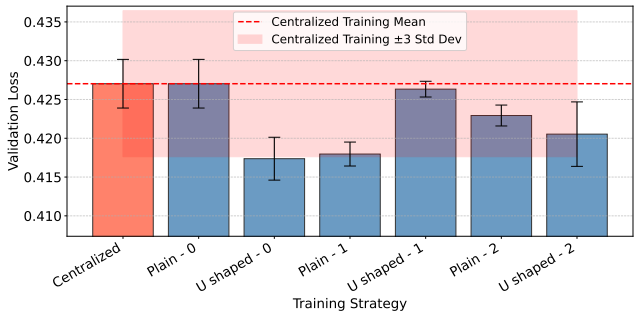

Fig. 4. Average validation loss comparison across training strategies, with centralized training as a reference.

as well as the limited precision of the floating-point representation.

B. Validation Loss

Fig. 4 shows the final validation loss after the fine-tuning process completes across all training configurations. We observe that the obtained models are similar, as indicated by the overlapping error bars in the plot. While the mean validation losses of these strategies vary slightly, the standard deviations (error bars) show considerable overlap, suggesting that the performance differences are within a margin of statistical uncertainty.

To further study whether there are significant differences in validation loss across the different training configurations, we conduct a one-way Analysis of Variance (ANOVA) test. The results yielded an F-statistic of 2.087 with a p-value of 0.12045. Since the p-value is greater than the conventional significance threshold of 0.05, we do not reject the null hypothesis, indicating that there is no statistically significant difference in validation loss among the evaluated training strategies. This suggests that all configurations – whether centralized or SL – achieve comparable performance in terms of validation loss.

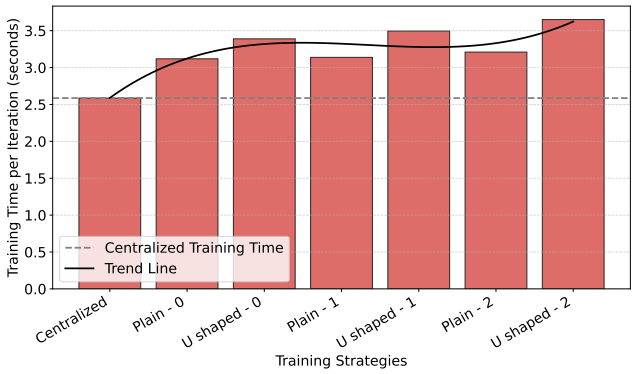

Fig. 5. Training time for different strategies, showing the effect of SL.

C. Training Time

Fig. 5 shows the training time for each strategy for 50 iterations. We report the training times for SL as well as centralized training. Note that centralized training values refer to training entirely on the cloud and thus serve as a lower bound for the training time. In our setting, the client cannot train the model purely locally as it does not have enough memory.

Centralized fine-tuning, utilizing 4 × A10G (24GB each), completes all computations locally on high-performance hardware without client-server communication overhead; thus, it is expected to be faster than SL approaches.

In SL, runtime behavior depends on the client-side workload and communication overhead. Fewer client layers reduce computation on the client, which leads to faster runtime as the client has more modest resources (Tesla T4, 16GB) than the server.

Between the two SL variants, the U-shaped approach generally incurs higher training time than plain SL due to the additional forward and backward pass computations at the client and the additional traverse of the network.

D. GPU Utilization

The GPU utilization shown in Fig. 6 illustrates the distribution of memory usage between the server and the client across different training strategies. In the Centralized setup, there is only a single server GPU, which utilizes around 55 GB of memory, as no client is involved in training. In SL configurations, a portion of the model is offloaded to the client, leading to a decrease in server memory usage and a corresponding increase in client memory usage.

However, the total GPU memory consumption (server + client) is not constant across configurations. This variation is primarily due to the difference in computation between plain and U-shaped SL. In U-shaped SL, the client must perform both the initial forward pass and the final backward pass, leading to higher activation storage and gradient computations,

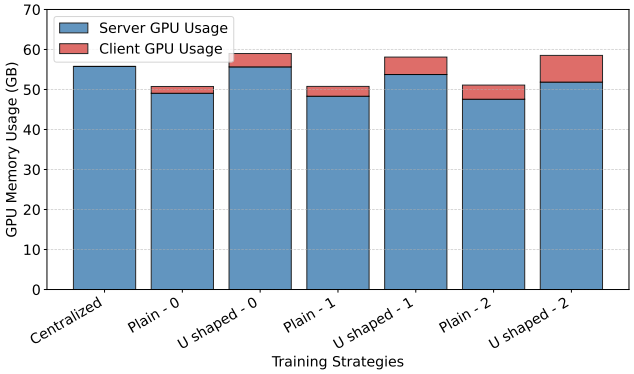

Fig. 6. GPU memory usage distribution between server and client across training strategies.

TABLE I
FLOPs Distribution for Different Client Layers and Centralized Training

Approach	Client FLOPs	Server FLOPs	Total FLOPs
Centralized	0.00e+00	4.32e+13	4.32e+13
0 layer	0.00e+00	4.32e+13	4.32e+13
1 layer	1.08e+12	4.21e+13	4.32e+13
2 layers	2.16e+12	4.11e+13	4.32e+13

which increase its memory usage. In contrast, Plain SL offloads all backward computations to the server, reducing the client's memory footprint. For example, in U-shaped - 2, the client memory usage is significantly higher compared to Plain - 2, reflecting the additional processing burden.

Table I shows the distribution of the Floating Point Operations per Second (FLOPs) across different training strategies. Note that the total FLOPs remains constant, as expected.

E. Computational and Communication Cost Analysis

We now present the computational and communication cost analysis for fine-tuning the full dataset, which consists of 2.5 million data points. The results offer insights into the total training cost, factoring in both the computing server expenses and data transfer costs within a high-performance cloud infrastructure. Computing costs $5.672 per hour and egress data transfer costs 0.01$ per GB. The U-shaped SL approach requires 62.5 MB of data transfer per training iteration, whereas the Plain SL approach requires 31.25 MB per iteration. By comparing different training configurations, we evaluate the efficiency and financial implications of centralized and SL approaches. Table II summarizes the cost analysis for the full dataset.

F. Key Findings

The primary finding from these experiments is the validation loss and training loss behavior are nearly identical across all methods, suggesting that SL can achieve the same model performance as centralized learning, with no significant accuracy degradation.

Increasing the number of client-side layers in SL leads to longer training times, as more computations are performed on the client instead of the high-performance server. While plain SL initially shows a slight advantage in speed, U-shaped SL incurs additional overhead due to handling both the lowest and highest layers on the client. This trade-off highlights the impact of model partitioning choices on training efficiency. In contrast, centralized fine-tuning is faster than all SL configurations since it runs entirely on the high-performance server without relying on a weaker client device.

TABLE II
Cost Analysis Across Training Strategies in a Leader Cloud Hosting for Full Dataset

Strategy	Time (h)	Data Transfer (GB)	Total Cost (USD)
Centralized	224.51	N/A	1273.41
Plain-0	270.73	4882.8125	1584.43
Plain-1	272.44	4882.8125	1594.11
Plain-2	278.66	4882.8125	1629.37
U-Shaped-0	294.22	9765.625	1766.48
U-Shaped-1	303.42	9765.625	1818.66
U-Shaped-2	316.91	9765.625	1895.19

G. Implications of Findings

The results indicate that SL strategies – especially U-shaped – offer several key advantages without compromising performance:

Privacy Preservation: One of the most significant advantages of SL, particularly the U-shaped strategy, is its ability to limit direct access to raw data. Since only intermediate representations are shared with the server, SL reduces direct exposure of sensitive inputs. However, it is important to acknowledge that SL is not entirely immune to privacy attacks, such as reconstruction attacks where adversaries attempt to recover input data from embeddings. Further research is needed to assess the extent to which raw data can be reconstructed in SL and mitigate such risks effectively.

Scalability: SL methods provide a scalable approach to training large models. By distributing the model across multiple servers that are able to handle large models that may not fit on a single machine.

Resource Efficiency: Fine-tuning the model requires approximately 55 GB of GPU memory, which far exceeds the client's 16 GB capacity, making centralized training on the client impossible due to out of memory issue. SL resolves this limitation by offloading the largest portion of the model to the server and retaining only a smaller subset of layers on the client, SL drastically reduces the client's memory load. This division enables the client to train effectively within its memory limit, demonstrating that SL is a powerful solution for low-memory environments, especially when resource-constrained devices are involved.

VI. Conclusion and Future Work

Our study explores SL for fine-tuning LLMs while preserving privacy. We evaluated centralized, plain SL, and U-shaped SL, showing that SL achieves validation loss comparable to centralized training. The U-shaped strategy further enhances privacy by keeping both data and labels on the client side.

While our experiments used AceGPT, the proposed SL techniques can generalize to other LLMs. Future work should optimize training efficiency, particularly in distributed settings, and explore scalability for larger models and multi-client environments. Adaptive SL architectures that dynamically distribute computation could further improve efficiency and privacy.

Additionally, SL should be compared to anonymization-based approaches, which lack formal privacy guarantees and degrade text quality [30]. A systematic comparison could clarify trade-offs and inform better privacy-preserving fine-tuning strategies. Finally, real-world evaluations in privacy-sensitive domains like healthcare and finance would provide valuable insights into SL's practical viability.

ACKNOWLEDGMENT

This work was supported by the SDAIA-KAUST Center of Excellence in Data Science and Artificial Intelligence (SDAIA-KAUST AI). We are thankful to Fatima Al-Rammah for her contributions to the project.

REFERENCES

[1] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon, "Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing," *ACM Trans. Comput. Heal.*, 2022.

[2] G. Yang, E. J. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao, "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer," 2022.

[3] A. S. Luccioni, S. Viguier, and A. Ligozat, "Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model," *J. Mach. Learn. Res.*, 2023.

[4] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-Efficient Transfer Learning for NLP," in *ICML*, 2019.

[5] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in *ICLR*, 2022.

[6] J. He, C. Zhou, X. Ma, T. Berg-Kirkpatrick, and G. Neubig, "Towards a Unified View of Parameter-Efficient Transfer Learning," in *ICLR*, 2022.

[7] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, "Privacy preservation in federated learning: An insightful survey from the GDPR perspective," *Comput. Secur.*, 2021.

[8] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, 2018.

[9] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018.

[10] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split Learning for collaborative deep learning in healthcare," 2019.

[11] K. Palanisamy, V. Khimani, M. H. Moti, and D. Chatzopoulos, "SplitEasy: A Practical Approach for Training ML models on Mobile Devices," in *HotMobile*, 2021.

[12] S. Zhang, Z. Zhao, D. Liu, Y. Cao, H. Tang, and S. You, "Edge-assisted U-shaped split federated learning with privacy-preserving for Internet of Things," *Expert Syst. Appl.*, 2025.

[13] C. Thapa, M. A. P. Chamikara, S. Camtepe, and L. Sun, "SplitFed: When Federated Learning Meets Split Learning," in *AAAI*, 2022.

[14] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal, "Evaluation and Optimization of Distributed Machine Learning Techniques for Internet of Things," *IEEE Trans. Computers*, 2022.

[15] M. Gawali, C. S. Arvind, S. Suryavanshi, H. Madaan, A. Gaikwad, K. N. B. Prakash, V. Kulkarni, and A. Pant, "Comparison of Privacy-Preserving Distributed Deep Learning Methods in Healthcare," in *MIUA*, 2021.

[16] J. Jeon and J. Kim, "Privacy-Sensitive Parallel Split Learning," in *ICOIN*, 2020.

[17] A. Chopra, S. K. Sahu, A. Singh, A. Java, P. Vepakomma, V. Sharma, and R. Raskar, "AdaSplit: Adaptive Trade-offs for Resource-constrained Distributed Deep Learning," 2021.

[18] A. Ayad, M. Renner, and A. Schmeink, "Improving the Communication and Computation Efficiency of Split Learning for IoT Applications," in *GLOBECOM*, 2021.

[19] D.-J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating federated learning with split learning on locally generated losses," in *FL-ICML*, 2021.

[20] M. Kohankhaki, A. Ayad, M. Barhoush, and A. Schmeink, "Parallel Split Learning with Global Sampling," 2024.

[21] S. Oh, J. Park, P. Vepakomma, S. Baek, R. Raskar, M. Bennis, and S. Kim, "LocFedMix-SL: Localize, Federate, and Mix for Improved Scalability, Convergence, and Latency in Split Learning," in *WWW*, 2022.

[22] E. Samikwa, A. D. Maio, and T. Braun, "ARES: Adaptive Resource-Aware Split Learning for Internet of Things," *Comput. Networks*, 2022.

[23] W. Fan, P. Chen, X. Chun, and Y. Liu, "MADRL-Based Model Partitioning, Aggregation Control, and Resource Allocation for Cloud-Edge-Device Collaborative Split Federated Learning," *IEEE Trans. Mob. Comput.*, 2025.

[24] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Çamtepe, Y. Gao, H. Kim, and S. Nepal, "Can We Use Split Learning on 1D CNN Models for Privacy Preserving Training?" in *ASIA CCS*, 2020.

[25] Z. Lin, X. Hu, Y. Zhang, Z. Chen, Z. Fang, X. Chen, A. Li, P. Vepakomma, and Y. Gao, "SplitLoRA: A Split Parameter-Efficient Fine-Tuning Framework for Large Language Models," 2024.

[26] C. Jeong, "Domain-specialized LLM: Financial fine-tuning and utilization method using Mistral 7B," *J. Intell. Inf. Syst.*, 2024.

[27] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018.

[28] H. Huang, F. Yu, J. Zhu, X. Sun, H. Cheng, D. Song, Z. Chen, M. Alharthi, B. An, J. He, Z. Liu, J. Chen, J. Li, B. Wang, L. Zhang, R. Sun, X. Wan, H. Li, and J. Xu, "AceGPT, Localizing Large Language Models in Arabic," in *NAACL*, 2024.

[29] B. Radovic, M. Canini, S. Horváth, V. Pejovic, and P. Vepakomma, "Towards a Unified Framework for Split Learning," in *EuroMLSys*, 2025.

[30] T. Yang, X. Zhu, and I. Gurevych, "Robust Utility-Preserving Text Anonymization Based on Large Language Models," 2024.