# DisLLM: Distributed LLMs for Privacy Assurance in Resource-Constrained Environments

Sumudith Sadeepa*, Keshara Kavinda†, Emmanuel Hashika‡,
Chamara Sandeepa§, Tharindu Gamage¶, Madhusanka Liyanage‖
*† ‡ ¶Faculty of Engineering, University of Ruhuna, Sri Lanka
§‖School of Computer Science, University College Dublin, Ireland
*sadeepa_pmas_e21@engug.ruh.ac.lk, †kavinda_bgk_e21@engug.ruh.ac.lk, ‡hashika_wde_e21@engug.ruh.ac.lk
§abeysinghe.sandeepa@ucdconnect.ie, ¶tharindu@eie.ruh.ac.lk, ‖madhusanka@ucd.ie

***Abstract*—Large Language Models (LLMs) have revolutionized natural language processing, but deploying them in resource-constrained environments and privacy-sensitive domains remains challenging. This paper introduces the Distributed Large Language Model (DisLLM), a novel distributed learning framework that addresses privacy preservation and computational efficiency issues in LLM fine-tuning and inference. DisLLM leverages the Splitfed Learning (SFL) approach, combining Federated Learning (FL) and Split Learning (SL) benefits for privacy-preserving and computationally efficient LLM deployment. It splits the LLM into client-side and server-side components, allowing sensitive data to remain on client devices while distributing the computational load. An efficient fine-tuning technique called Low-Rank Adaptation (LoRA) is incorporated to reduce trainable parameters and memory footprint. Additionally, Local Differential Privacy (LDP) is used to add controlled noise to the sensitive data, further enhancing privacy preservation. Experimental evaluations on various text classification datasets demonstrate DisLLM's effectiveness in model accuracy, convergence rate, and computational efficiency. Factors like the number of clients, cut layer position, and model variants are thoroughly investigated. Results show that DisLLM achieves comparable accuracy to centralized fine-tuning while preserving data privacy and optimizing resource utilization. Moreover, its ability to handle multiclass classification tasks with many classes emphasizes its robustness and versatility. Therefore, DisLLM represents a significant step forward in the practical deployment of LLMs, ensuring efficiency and privacy preservation.**

***Index Terms*—LLMs, Finetuning, Distributed Learning System, Resource Constraint Environment, Splitfed, LoRA**

## I. Introduction

The surge of next-generation Artificial Intelligence (AI) has been observed in recent years with the introduction of LLMs. These complex AI models have revolutionized how computers interact with language, enabling them to understand the nuances of human language through extensive training on vast amounts of real-world text [1]–[3]. They excel in text generation, translation, summarisation, and question-answering tasks and can even comprehend images and text in files [4], [5]. A wide adoption of these models across many sectors can be observed, which also has enabled further research and development into LLMs.

Since the full capabilities of LLMs are yet to be discovered, there is an unforeseen challenge in terms of the security and privacy of utilizing these models in real-world applications. LLMs require a massive amount of data to be trained, and to make these models more practical for performing highly specific intelligent tasks, fine-tuning on domain-specific data is often necessary. However, this fine-tuning process is often outsourced to third-party service providers or cloud platforms due to the need for significant computational resources. This outsourcing raises concerns about privacy and security, as the data used for fine-tuning may include private and sensitive information, which could be exposed or mishandled during the process.

For example, an organization which owns sensitive information such as medical records or financial information may face difficulty in performing LLM-based fine-tuning by sending it to third parties. Transfer learning offers a potential solution to these challenges by leveraging knowledge from pre-trained models to enhance performance on specific tasks with limited data. This approach enables LLMs to adapt efficiently to specialized domains, requiring fewer data and computational resources than training from scratch. Despite the benefits of transfer learning, typical LLMs cannot be trained on client devices since they are massive models with billions of parameters that require immense computational demands during fine-tuning and inferencing. Therefore, a full deployment of LLMs on devices like mobile phones or laptops is particularly challenging due to their limited computational power and storage capacity.

Therefore, we introduce the DisLLM approach to address these issues via privacy-preserved distributed LLM training. Our DisLLM leverages the SFL approach [6] and is primarily designed for LLMs. This innovative approach introduces several novel contributions that enhance the utility and applicability of LLMs across various domains like healthcare and finance. DisLLM can fine-tune an LLM on sensitive data in a privacy-preserving manner while addressing the problem of computational demand. Therefore, both privacy and computational efficiency are enhanced. Ensuring that sensitive raw data remains on the client device minimizes the risk of privacy breaches.

Additionally, it optimizes the distribution of computational tasks between the client-side and server-side components, allowing each part of the system to operate within its capacity

and enhancing the overall efficiency and performance of the model. DisLLM-based models can fine-tuned dynamically for highly specific, customizable contexts, such as personal digital assistants or sensitive medical data analysis. This can be done without the need for sending user data to untrustworthy third parties, and facilitating low-end computing devices to join the training process with limited computational requirements. This adaptability ensures that DisLLM can meet the diverse needs of modern AI applications, making it a valuable advancement in the field of Machine Learning (ML).

### A. Our Contributions

We provide our key contributions in the paper as follows:

- *Novel approach*: To the best of our knowledge, this is the first attempt at splitting LLM in a FL environment, combining the benefits of both SL and FL techniques.
- *Efficient resource utilization*: Our approach allows for efficient resource utilization on end devices by distributing the computational load across end devices and servers while providing privacy preservation for training data.
- *Model splitting architecture*: We introduce a novel architecture that efficiently splits the LLM from a specified cut layer, enabling effective model distribution across multiple devices.
- *Comprehensive experimental evaluation*: We conduct extensive experiments to validate our proposed approach's performance, efficiency, and privacy-preserving capabilities in various scenarios.

These contributions highlight the novelty and significance of our work in advancing the field of LLM training and deployment in resource-constrained and privacy-sensitive environments.

The rest of the paper is organized as follows: Section II provides background information. Section III reviews related works associated with our work. Section IV details the system model of DisLLM. Section V describes the process of DisLLM. Section VI presents experimental results. Section VII concludes the paper and discusses future work.

## II. Background

This section outlines the background preliminaries that provides an overview of the proposed approach is based on.

### A. Fine-tuning of LLM-based Models

The introduction of Generative Pre-trained Transformers (GPT) revolutionized the AI landscape by utilizing unsupervised learning and vast datasets. These GPT models showcased remarkable abilities in understanding and generating natural language, leading to improved performance in various language-related tasks. The GPT-2, introduced by Radford et al. [3], is a large unsupervised language model that uses transformers as its core architecture. This model has been trained on a diverse dataset of 8 million web pages, known as the WebText dataset, and consists of 1.5 billion parameters. The training objective of GPT-2 is straightforward yet powerful: predict the next word given all the previous words within a text. Therefore, GPT-2 has been used as the base model in our paper.

Despite their potential, LLMs face significant challenges in fine-tuning, primarily due to computational costs and privacy concerns. Their immense size makes it difficult for resource-constrained users to fine-tune them locally. To address this issue, a parameter-efficient fine-tuning technique called LoRA [7] has been incorporated to reduce the number of trainable parameters during the fine-tuning process. Moreover, fine-tuning these models often requires the client's sensitive data, which must be transmitted to remote servers, risking data exposure and privacy leakages. These issues present significant drawbacks that need to be addressed to enable privacy-assured fine-tuning of LLMs.

### B. Distributed ML

Therefore, Distributed ML techniques such as FL [8], SL [9], and SFL [6] have emerged as potential solutions to address privacy preservation and computational load distribution in model training. These approaches aim to enable collaborative learning while keeping sensitive data localized, thus mitigating privacy risks associated with centralized data collection and processing. Additionally, the privacy-preserving framework LDP [10] has been used to address issues related to data reconstruction by malicious devices in distributed learning scenarios. These methods collectively enhance the security and privacy of distributed learning systems.

## III. Related Works

In this section, we provide further details of the current state-of-the art related works on the LLM fine-tuning, different distributed ML approaches and privacy preservation techniques used in ML.

*1) Parameter-Efficient Fine-Tuning:* Hu et al. [7] introduced LoRA, a parameter-efficient fine-tuning technique that significantly reduces the number of trainable parameters in LLMs. LoRA introduces trainable low-rank matrices to specific weights within the Transformer architecture, particularly focusing on the attention mechanism's query and value projection matrices. This approach allows for fine-tuning a small number of additional weights while keeping most of the pre-trained network parameters frozen. Howard and Ruder [11] demonstrated that LoRA reduces parameter overhead and supports efficient task-switching while maintaining the same inference latency as a fully fine-tuned model.

*2) Federated Learning:* According to McMahan et al. [12], FL involves training ML models concurrently on local devices using private data [8], [13] for a predetermined number of local epochs. It allows for parallel processing across many clients, which enhances the efficiency of model training. Subsequently, these local updates are relayed to a central server for aggregation to form a global model. The parameters of this global model are then distributed back to all clients for further training in the subsequent round.

This iterative process continues until the algorithm reaches a point of convergence. FL's main advantage is its ability to train models without sharing raw data, thus preserving user privacy.

*3) Split Learning:* In a different approach, Vepakomma et al. [9] proposed SL, which splits a deep learning network mainly into two parts: one for the client side and another for the server side, each processed and computed on different devices [14]. The clients, which hold the private data, are in charge of the client-side part of the network, while the server manages the server-side part. During forward propagation, communication between these parts involves transmitting activations, known as smashed data, from the client-side network's split layer, also called the cut layer, to the server side. During backward propagation, the server-side network's split layer provides the smashed data's gradients to the client. Hence, the complete model will be trained collaboratively. Furthermore, the learning process synchronizes among multiple clients through centralized or peer-to-peer mode. SL offers better model privacy than FL because the raw data does not leave the client's device. However, SL has a drawback in that it can cause clients' resources to be idle and increase training overhead with many clients due to its relay-based training approach.

*4) Splitfed Learning:* Thapa et al. [6] introduced SFL, which combines both FL and SL to eliminate their inherent drawbacks and enhance data privacy and model robustness while retaining the advantages of both methods [15]. SFL provides test accuracy and communication efficiency similar to SL but with significantly decreased computation time per global epoch for multiple clients. It also improves communication efficiency over FL as the number of clients increases. SFL is beneficial in resource-constrained environments where full model training is not feasible and fast model training is required. However, it is important to note that SFL is not currently implemented on LLMs. Existing SFL techniques are primarily focused on conventional deep learning models with limited parameters, and they are not applicable to the more complex and parameter-heavy LLMs.

*5) Privacy-Preserving Techniques:* Differential Privacy (DP), introduced by Abadi et al. [10], provides privacy by adding controlled noise to sensitive data, hence the individual data points cannot be distinguishable within aggregated outputs. LDP is a variant of DP that applies noise directly into individual records before data leaves the client's device in a distributed learning system, ensuring even the server-side can't access raw data. LDP calibrates noise based on the privacy budget ($\epsilon$) and function sensitivity ($S$). The privacy budget ($\epsilon$) balances privacy and accuracy, where a smaller $\epsilon$ provides stronger privacy but results in lower accuracy, while a larger $\epsilon$ allows for greater accuracy at the expense of reduced privacy, and ($S$) determines the amount of noise that must be added to protect privacy. These approaches ensure data security and regulatory compliance [16] by making outputs insensitive to individual records, thus preserving privacy while fine-tuning a distributed model.

Our work, DisLLM, builds upon these existing techniques by combining the benefits of SFL, LoRA, and LDP to create a novel distributed learning framework for privacy-preserving and computationally efficient fine-tuning of LLMs in resource-constrained environments. Table I provides a further comparison of our approach with the existing related works.

TABLE I: Summary contribution of our work.

| **Characteristics** | **FL** [12] | **SL** [9] | **SFL** [6] | **Ours** |
|---|---|---|---|---|
| LLM-based fine-tuning | - | - | - | ✓ |
| Efficient model splitting | - | ✓ | ✓ | ✓ |
| Model aggregation | ✓ | - | ✓ | ✓ |
| Parameter-efficient fine-tuning | - | - | - | ✓ |
| Use of multiple clients | ✓ | ✓ | ✓ | ✓ |
| Parallel client-side training | ✓ | - | ✓ | ✓ |
| Raw data remains private | ✓ | ✓ | ✓ | ✓ |

## IV. System Model

The DisLLM describes how a distributed architecture collaboratively fine-tunes an LLM using multiple clients, a main server, and a Fed Server. The core concept involves splitting the LLM into two parts: one for client-side tasks and another for server-side processing. This division optimizes computational resources while safeguarding user data.

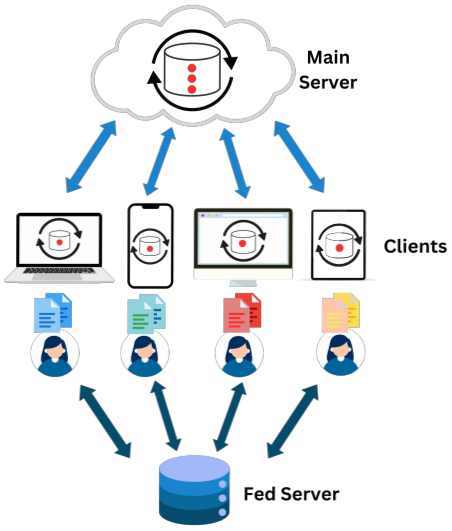


Fig. 1: System model of DisLLM architecture

In this architecture, clients play an active role in both pre-processing tasks and fine-tuning their respective client-side model portions on resource-constrained devices. The output of the client-side model, referred to as smashed data, is sent to the main server. Meanwhile, the server efficiently manages the receipt of smashed data from clients and facilitates the fine-tuning process using its available higher computational resources.

Fed Server is responsible for averaging the client-side models and creating the global client-side model. After each epoch, the updated local client-side models from all clients are sent to the Fed Server. The Fed Server then computes the

average of these models to form the global client-side model, which is subsequently distributed back to all clients for the next epoch of training.

The communication between clients, the main server, and the Fed Server is crucial. Clients transmit processed data, known as smashed data, to the main server. The main server processes this data and provides gradients of the smashed data for backpropagation. The Fed Server averages the updated client-side models to create the global client-side model.

The architecture diagram in Figure 1 illustrates this system's structure and data flow. We have considered a system of 12 clients and a main server. It clarifies the roles of clients, the main server, and the Fed Server, emphasizing the split model approach and the exchange of data between them. The diagram also highlights how the Fed Server integrates into the architecture to ensure efficient model averaging and distribution.

## V. The Distributed LLM Process

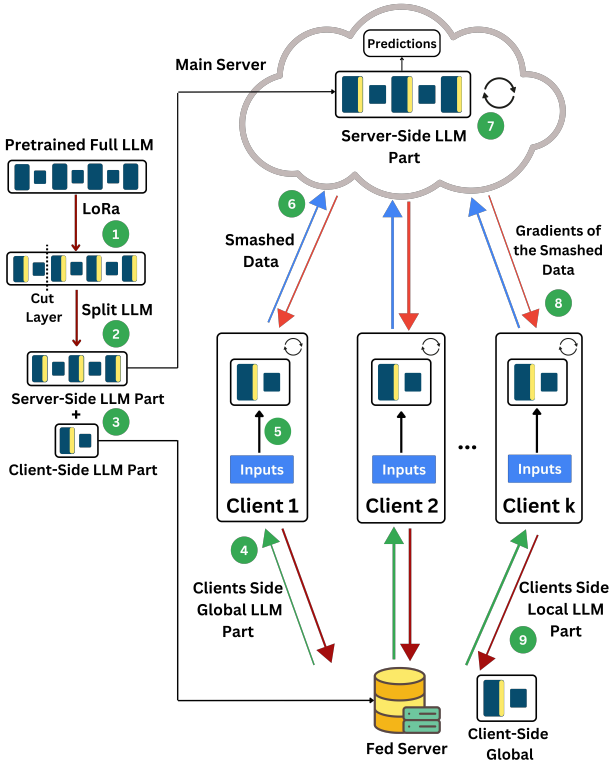


Fig. 2: The DisLLM process overview

The DisLLM process begins by taking an LLM and assigning pre-trained weights provided by the manufacturers of the LLM. Figure 2 shows a visual representation of the DisLLM architecture and process flow, with numbered steps corresponding to the process described below. To optimize the computational resources required for fine-tuning, step 1 applies LoRA, a parameter-efficient fine-tuning method. This approach reduces the number of trainable parameters.

In step 2, the LLM is split into two parts: one for the client side and one for the server side. This split is performed with careful consideration of the computational resources available on the client devices, ensuring that the client-side model remains lightweight enough to run on resource-constrained devices. As step 3, the client-side model is sent to the FedServer, where it is referred to as the global client-side model, while the server-side model is sent to the main server. Step 4 involves the FedServer, then distributes the global client-side model to every client. Within each client, the LLM model portion is called the local client-side model, allowing the distributed fine-tuning process to begin efficiently without compromising performance or privacy.

Clients preprocess their local datasets, performing essential tasks such as tokenization and padding. For text data preparation, the tiktoken tokenizer [3] is used to convert raw text into token IDs, with a maximum length limit to truncate excessively long sequences. Padding is applied to handle varying text lengths and ensure uniform input size, which standardizes the data and maintains consistent batch dimensions across the dataset. Following the preprocessing, step 5 involves clients conducting a forward pass through the client-side model. To ensure the protection of sensitive client data and prevent any attempt to reconstruct it from the output of the client-side model, we apply LDP by adding Laplacian noise to the embeddings after the positional embedding layer, where the token embedding layer converts discrete input tokens into dense vector representations, capturing the semantic information essential for meaningful text processing. Adding noise before this conversion could undermine the model's ability to accurately capture semantic relationships. Therefore, the noise is introduced after the positional embeddings to maintain the integrity of the token representations while enhancing privacy.

First, we can calculate the scale parameter $b$ of the Laplace distribution based on the sensitivity $S$ and privacy budget $\epsilon$, as shown in Equation 1:

$$b = \frac{S}{\epsilon} \tag{1}$$

Then, Laplacian noise $\mathcal{L}(0, b)$ can be added to the embeddings to ensure DP, as shown in Equation 2:

$$e_{\text{noisy}} = e + \mathcal{L}(0, b) \tag{2}$$

Here, $e$ represents the original output of embedding and $\mathcal{L}(0, b)$ denotes the Laplace distribution with mean 0 and scale $b$.

Equations 1 and 2 illustrate the calculation of the scale parameter and the addition of Laplacian noise, respectively.

In step 6, the output from the final layer of the client-side model, known as smashed data, is sent to the main server, completing the forward pass at the client level. Step 7 begins as the main server receives smashed data from clients, and the server processes these data. For each client's smashed data, the server performs a forward pass through the server-side model, obtaining the predictions. These predictions are then sent back to the respective clients.

**Algorithm 1** DisLLM Fine-Tuning

**Notations:** $S_t$ is a set of $K$ clients at time instance $t$; $D_{k,t}$ is the smashed data of client $k$ at time $t$; $Y_k$ and $\hat{Y}^k$ are the true and predicted labels of client $k$; $\nabla \ell_k$ is the gradient of the loss for client $k$; $W^C$ and $W^S$ are the client-side and server-side LLM model parts; $\Delta D_{k,t}$ gradients of smashed data; (7) $e_{k,t}$ and $e_{k,t}^{noisy}$ are the outputs of the positional embedding layer ($P_{k,t}$) before and after adding noise.

**Execution on FedServer:**
**if** $t = 0$ **then**
  Initialize global $W_t^C$
  Send $W_t^C$ to all $K$ clients
**else**
  Receive $W_{t+1}^C$ from all $K$ clients
  Update the global $W_{t+1}^C \leftarrow \frac{1}{k} \sum_{k=1}^{K} W_{k,t}^C$
  Send updated global $W_{t+1}^C$ to all $K$ clients
**end if**
**Execution on Clients:**
**if** $t = 0$ **then**
  Receive initial $W_{k,t}^C$
**else**
  **for** each client $k \in S_t$ in parallel **do**
    Forward pass until $P_{k,t}$: $W_{k,t}^C \rightarrow e_{k,t}$
    Add noise $e_{k,t} + Noise \rightarrow e_{k,t}^{noisy}$
    Forward pass: $e_{k,t}^{noisy} \rightarrow D_{k,t}$
    Send $D_{k,t}$ to main server
    Receive $\hat{Y}^k$ from main server
    Calculate loss: $\mathcal{L}(Y^k, \hat{Y}^k)$
    Sending loss to the main server
    Receive $\Delta D_{k,t}$ from main server
    $W_{k,t}^C \leftarrow$ Client Backpropagate $(\Delta D_{k,t})$
    Send updated $W_{k,t}^C$ to FedServer
  **end for**
**end if**
**Execution on Main Server:**
**for** each client $k$ in $S_t$ in parallel **do**
  Receive $D_{k,t}$
  Forward pass: $W_{k,t}^S \rightarrow \hat{Y}_k$
  Send $\hat{Y}_k$ to client
  Receive loss
  Backpropagate: Compute $\nabla \ell_k(W_t^S; D_t^S)$
  Send $\Delta D_{k,t}$ to client
  Update server-side model:
    $W_{t+1}^S \leftarrow W_t^S - \eta \nabla \ell_k(W_t^S)$
**end for**

Using the labels and the predictions received from the main server, each client calculates the loss and sends it back to the main server. The server then performs backpropagation on the server-side model, updating its parameters based on the gradients of the server-side model parameters with respect to the loss. The server also calculates the gradients of the smashed data with respect to the loss and sends these gradients back to the clients as step 8. The clients use these gradients to perform backpropagation and update their local model parameters, ensuring synchronized updates across the system.

As step 9, after each epoch, the updated client-side models are sent to the Fed Server. Once all the client updates are received, the Fed Server performs model averaging to create a new global client-side model. This updated global client-side model is then sent to all clients, and the next epoch begins. This process is repeated for every epoch, resulting in a well-tuned client-side model. We summarize the DisLLM process in Algorithm 1.

The DisLLM architecture prioritizes client privacy through a multi-layered approach. At its core, DisLLM implements "privacy-by-design," ensuring raw input data never leaves the client's device. This distributed approach localizes data processing to each client's environment, mitigating risks of unauthorized access or data breaches. Additionally, DisLLM applies Laplacian noise to the data before transmission, providing an extra layer of security against potential reconstruction attempts. The server only handles "smashed data" or privacy-protected intermediate outputs. This combination of localized processing, noise injection, and sharing only processed outputs ensures robust protection of sensitive information throughout the fine-tuning process. By keeping raw data local and adding noise to processed data, DisLLM maintains data confidentiality even in privacy-critical applications.

## VI. Experimental Results

This section provides information on the experiments performed on the DisLLM and highlights the key observations from the obtained results.

### A. Experimental Setup and Dataset Information

In our DisLLM implementation, we conducted experiments simulating a system of 10 clients and a server, with some experiments utilizing up to 12 clients, to imitate a small to medium-sized real-world distributed learning scenario. This setup allowed us to validate DisLLM's functionality and performance, particularly for fine-tuning a substantial model like GPT-2. The experiments were developed using PyTorch in the Google Colab environment, utilizing 15GB T4 Graphical Processing Unit (GPU) and 12GB Random Access Memory (RAM), which enabled us to simulate both clients and servers within the same environment. Datasets were distributed across multiple clients to represent an accurately distributed learning setup.

In the experiments, we used GPT-2 small with 124 million parameters and GPT-2 medium with 355 million parameters. The GPT-2 small model has 12 transformer blocks, while the medium model has 24. Both client-side and server-side models were trained with a learning rate of 1e-5 to ensure stable convergence and optimal performance across all experiments.

We used four text classification datasets, specifically selected for their relevance to privacy protection. The datasets

are detailed in Table II. The Phone Dialogue Dataset [17] is a comprehensive collection of simulated telephonic conversations, categorized into eight distinct classes: delivery, social security number (SSN), support, reward, wrong, telemarketing, refund, and insurance. Similarly, the "Perverted and Normal Chats" dataset [18] is a collection of short text conversations labelled as either "0" for normal chats or "1" for perverted or inappropriate chats. This dataset comprises various chat messages, ranging from casual conversations about daily activities to more explicit or suggestive content. Furthermore, we utilized the DBpedia Classes dataset [19] as a text classification dataset containing 70 and 219 classes. The data comes from the DBpedia project, which extracts structured information from Wikipedia articles. Lastly, we incorporated the SMS Spam Collection dataset [20], a public dataset of SMS messages that have been labelled as either "spam" or "ham" (legitimate). It was compiled for mobile phone spam research and contains SMS messages in English.

### *B. Effect of the Number of Clients on Model Accuracy Convergence*

As the initial experiment, we investigate the impact of the number of clients on model accuracy in a DisLLM setup, considering four scenarios: 4, 6, 8 and 12 clients. As illustrated in Figure 3, all client setups initially demonstrated very close testing accuracy. However, as the number of epochs increases, the setup with the fewest clients (4 clients) demonstrates the highest rate of accuracy improvement. The setups with 6 and 8 clients exhibit a more gradual accuracy improvement, and the 12-client setup demonstrates the slowest and most steady improvement over the epochs. With fewer clients, accuracy tends to increase rapidly at first but slows down as the number of epochs progresses. In contrast, more clients result in a more consistent and steady improvement.

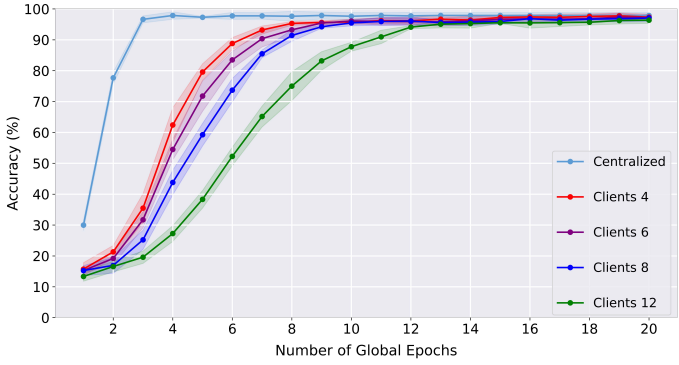


Fig. 3: DisLLM testing accuracy variation across varying epochs for different numbers of clients

This behavior can be attributed to the data distribution among clients. With fewer clients, each client receives a larger portion of the total dataset, allowing for more comprehensive learning from a broader data subset in each epoch. This leads to quicker convergence as each client can contribute more meaningful updates to the global client-side model. Conversely, as the number of clients increases, the amount of data per client decreases. This results in each client having a smaller, potentially less representative sample of the total dataset. Consequently, more epochs are required for the global client-side model to effectively learn from the entire dataset, as each client's contribution in a single epoch is based on a smaller data subset. Despite these variations, all configurations eventually converge to similar accuracy levels by the end of 12 global epochs, demonstrating the DisLLM framework's effectiveness across different client distributions.

### *C. Performance of finetuning centralized and DisLLM*

In this experiment, we compare the accuracy of the DisLLM model with the traditional centralized fine-tuning scenario. In the centralized setup, the entire model is fine-tuned on a remote server with access to all the sensitive client-side data. In contrast, the DisLLM model divides the fine-tuning process between a client and a central server, making the sensitive data inaccessible at remote servers.

For this, we employed ten clients to participate in the fine-tuning process, spanning 20 global epochs with a batch size of 32 samples. Referring to Figure 3, it is observed that the centralized setup reaches the optimal accuracy within fewer epochs than the DisLLM setups with varying clients. However, after several epochs, all the DisLLM setups with varying clients achieve the same optimal accuracy as the centralized setup. Thus, the final accuracies are similar regardless of the centralized or DisLLM approach. Although the centralized approach achieves optimal accuracy in fewer epochs, it does not provide privacy for sensitive client-side data. In contrast, DisLLM offers complete privacy for sensitive client-side data.

When the DisLLM setup was implemented in physically separated systems, a communication delay time will be introduced into the complete system. which has not been considered in our research experiments. Specifically, our simulations were conducted in the Google Colab environment, where there is no communication overhead in the virtual simulations. Additionally, in real implementations, the communication delay time in DisLLM is significantly lesser than in traditional FL systems, as DisLLM transmits only the cut layer activations through communication channels, rather than sending complete model parameters like in FL systems. However, the centralized system requires no communication time as the complete model resides in one server. This communication time arises due to the distributed nature of the DisLLM architecture, wherein data exchanges between clients and the central server require additional network latency.

Therefore, the choice between a centralized or DisLLM approach depends on the priority assigned to privacy versus convergence speed. DisLLM provides enhanced privacy but requires more epochs to reach optimal accuracy. Hence, it is more suitable for privacy-crucial applications.

### D. GPU consumption for finetuning of split-fed and DisLLM

Here, we compare the GPU usage of the DisLLM setup with the split-fed setup for different numbers of clients. Both models divide the fine-tuning process between a client and a central server, making the sensitive data inaccessible to remote servers. The GPU consumption reported in this experiment represents the total GPU usage for the entire setup, including both the server and all participating clients. We conducted experiments with 1, 5, and 10 clients participating in the fine-tuning process, spanning 20 global epochs with a batch size of 32 samples.

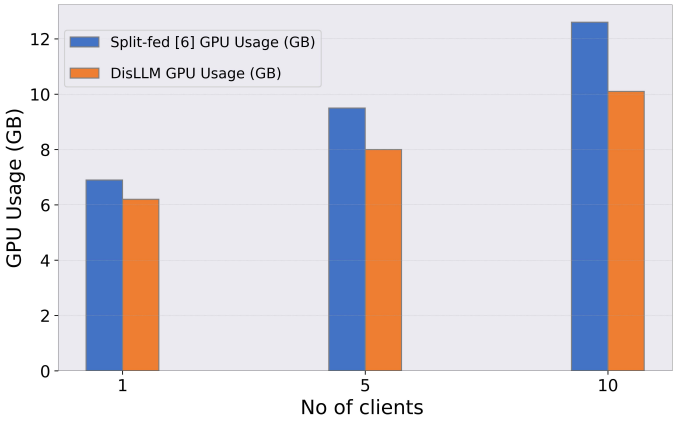


Fig. 4: Comparison of GPU Usage between Split-fed [6] and DisLLM methods across different numbers of clients

The results from the histogram in Figure 4 show that DisLLM consistently demonstrates more efficient GPU usage than the Split-fed model across varying client numbers. As the number of clients increases to 1, 5, and 10, the additional GPU requirement needed for split-fed compared to DisLLM increases to 0.7 GB, 1.5 GB, and 2.5 GB, respectively. In percentage terms, DisLLM achieves approximately 10.1% lower GPU memory usage for 1 client, 15.8% lower for 5 clients, and 19.8% lower for 10 clients compared to the split-fed model. Therefore, the DisLLM architecture demonstrates an increasing reduction in GPU usage compared to the split-fed model as the number of clients grows, and this will be particularly noticeable as the number of clients increases, consistently requiring less GPU memory than the split-fed model. This efficiency gain can be attributed to DisLLM's incorporation of LoRA, which reduces the number of trainable parameters.

These findings indicate that the DisLLM architecture offers more advantages in GPU resource utilization, especially in scenarios involving multiple clients, and is crucial for optimizing resource allocation and performance in distributed systems, where efficient GPU usage significantly impacts overall system performance and cost-effectiveness.

### E. Impact of Cut Layer Position on Model Performance

As the next experiment, we investigated the impact of the cut layer's position on model accuracy within the DisLLM architecture. We have utilized the GPT-2 model, which has 12 transformer blocks, for our experiment. The cut layer's position determines the layer from which the model is divided between the client and the server, and this position significantly impacts the efficiency of the training process. Therefore, we have conducted a series of experiments by varying the cut layer position and evaluating the model's accuracy after each fine-tuning epoch.

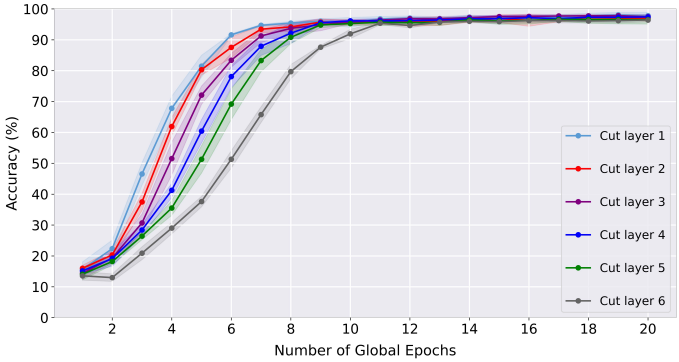


Fig. 5: DisLLM testing accuracy variation across varying epochs for different cut layer positions

The experiments were started by assigning the initial part of the model with the first transformer block to the client side and the remaining blocks to the server side. This allowed us to analyze how the split position affects the overall fine-tuning process. As we shift the cut layer deeper into the model, from cut layer 1 to 6, the client-side model complexity increases from 8.33% to 50% of the total model, while the server-side model complexity decreases from 91.67% to 50%. However, the majority of the model always resided on the server side in all configurations.

Our results, illustrated in Figure 5, showed that the model's accuracy was initially low across all cut layer positions and gradually improved as training progressed by completing more epochs. Whereas shallower-cut layers experience quicker accuracy increments than deeper-cut layers. Eventually, all cut layer setups converged to the same accuracy level, while deeper cut layers required more epochs to converge than shallower cut layers. This observation suggests that with deeper cut layers, clients have more layers to update independently before averaging. This can lead to more diverse updates across clients, especially in the early epochs. When these diverse updates are averaged, it may take more epochs for the global client-side model to converge to an optimal state that balances the contributions from all clients.

Therefore, this highlights the trade-off between the distribution of computational workload and the number of epochs needed to achieve optimal accuracy when choosing the cut layer position.

### F. Impact of Number of Label Classes on DisLLM Accuracy

To evaluate the effectiveness and accuracy of our proposed DisLLM-based LLM architecture for solving multiclass problems, we conducted a series of experiments using several datasets with varying numbers of label classes. The primary

objective was to highlight the significance of having LLMs architectures compared to conventional ML architectures like LSTM or GRU and how the accuracy behaves with datasets with different classes. The DisLLM model was trained for each dataset using the same hyperparameters and training procedure for 10 clients.

| Dataset | No. of classes | Model Accuracies (%) | | |
|---|---|---|---|---|
| | | GRU | LSTM | LLM |
| Perverted and normal | 2 | 83.42 | 84.18 | **96.97** |
| Spam and ham | 2 | 97.97 | **98.68** | 96.23 |
| Phone Dialogue Dataset | 8 | 82.81 | 85.42 | **97.39** |
| DBPedia | 70 | 77.57 | 78.31 | **86.50** |
| DBPedia | 219 | 63.87 | 65.48 | **89.38** |

TABLE II: Testing Accuracies Across Various Datasets for Different Models

The results, shown in Table 1, demonstrate that the DisLLM architecture achieves relatively high accuracy across most of the tested datasets. However, there is a noticeable trend of overall accuracy decreasing as the number of classes in the dataset increases across all model types. The LLM obtains a significantly high accuracy of over 96% for the binary classification tasks, but the accuracy is lower at 89.38% for the 219-class dataset (DBPedia 219 classes dataset). This behaviour depicts that classification becomes more challenging as the number of possible classes grows, even with the use of LLM.

However, the DisLLM architecture demonstrates comparatively much better accuracy for datasets with many classes, maintaining accuracy above 89% on a 219-class problem relative to other models with only limited to 65%, highlighting its robustness and capacity to handle complex classification tasks better.

### *G. Effect of GPT-2 Variants on Accuracy and Convergence Rate*

This study examined the accuracy of two GPT-2 models with distinct parameter counts: 124 million and 324 million. Incorporating LoRA adapters significantly decreases the number of trainable parameters to 1.3 million for the 124 million parameter model and 3.5 million for the 324 million parameter model. This adjustment aims to streamline the training process while maintaining the models' robust capabilities.

As illustrated in Figure 6, the model with more parameters initially shows lower accuracy after the first training epoch. However, as training progresses, the larger model's accuracy improves largely per epoch than the smaller model. After each subsequent epoch, the higher-parameter model consistently achieves better accuracy than the lower-parameter model. This indicates that the model with more parameters converges to its optimal performance level with fewer epochs as the training progresses. The medium and small models reached their convergence points after the 6th and 8th epochs, respectively, beyond which further training yielded no additional improvements in accuracy. Hence, the two

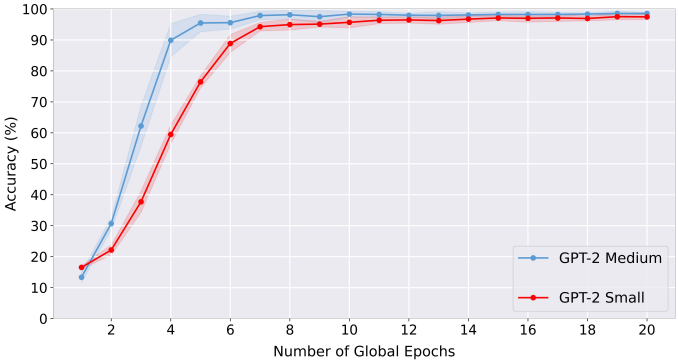


Fig. 6: DisLLM testing accuracy variation across varying epochs for different GPT-2 variants

models attain and maintain their maximum accuracy for the remaining epochs.

These findings can be explained primarily by two factors. Firstly, the larger model's initial lower accuracy likely results from its increased complexity, requiring more initial training to effectively organize its larger parameter space. This explains the slower start for the larger model. Secondly, the larger model's capacity to capture more complex patterns enables it to reach optimal performance faster, needing fewer epochs to converge.

### *H. Effect of Local Differential Privacy on Testing Accuracy*

In this experiment, we analyzed the impact of LDP using different levels of privacy budgets ($\epsilon$) for the testing accuracy of our model. The experiments were conducted under a setup similar to previous experiments with 10 clients, where noise was added to the embeddings to ensure privacy. The graph presents the testing accuracy recorded for experiments with different values of $\epsilon$, specifically 2.0, 3.0, 3.5, 4.0, 4.5, 5.0, and without noise.

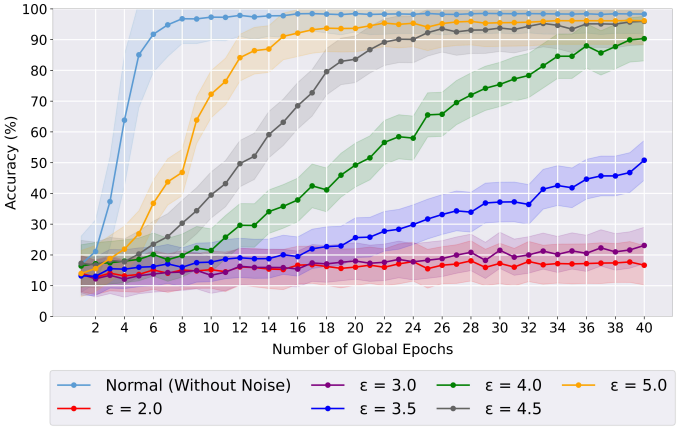


Fig. 7: DisLLM testing accuracy variation across varying epochs for different privacy budget values ($\epsilon$)

From the graph in Figure 7, it is evident that with ($\epsilon = 5.0$), the model achieves the highest accuracy, reaching close to 90% by the end of the training epochs. As the privacy budget decreases, the accuracy also decreases, with ($\epsilon = 2.0$) resulting in significantly lower accuracy, ranging around 20% for most of the training period. Moderate privacy budgets

($\epsilon = 3.5$ to $\epsilon = 4.5$) show a balanced trade-off, maintaining reasonable accuracy levels between 50% to 70%. Therefore, higher privacy budgets ($\epsilon = 5.0$) result in better accuracy, while lower privacy budgets ($\epsilon = 2.0$) degrade the model's performance due to the added noise.

These results demonstrate that stronger privacy (lower $\epsilon$) ensures better protection against data reconstruction, and it also introduces more noise, which can adversely affect the model's learning ability and performance. Conversely, a higher privacy budget (higher $\epsilon$) provides less noise interference, allowing the model to achieve higher accuracy but with weaker privacy guarantees. Therefore, selecting an appropriate privacy budget is crucial in balancing the trade-off between privacy and accuracy. Depending on the application and the required privacy levels, this balance can be adjusted to achieve optimal performance while ensuring the necessary privacy protection for the users' data.

## VII. Conclusion and Future Works

In this research, we introduced DisLLM, a novel distributed learning framework that addresses the challenges of privacy preservation and computational efficiency in fine-tuning LLMs within resource-constrained environments. DisLLM provides a robust solution for fine-tuning any LLM model by innovatively combining the benefits of SL and FL with LoRA and incorporating LDP. Hence, DisLLM efficiently utilizes computational resources while maintaining the privacy of sensitive data.

Our experiments show that DisLLM performs similarly to traditional centralized models in terms of accuracy and efficiency while providing enhanced privacy protection. The inherent distributed architecture of DisLLM itself provides privacy, which is enhanced by applying LDP, adding further protection against potential privacy breaches. The DisLLM's adaptability across various scenarios and datasets highlights its potential for real-world applications where data privacy and resource limitations are prevalent concerns. Moreover, the versatility of DisLLM is further emphasized by its applicability to different domains, such as healthcare and finance, where privacy concerns are paramount. This adaptability, combined with enhanced privacy features and reduced computational demand, positions DisLLM as a valuable advancement in using distributed LLMs.

**Future Works:** Several areas can be explored further to enhance the DisLLM framework's capabilities.

- *Dynamic resource allocation*: Develop methods for dynamically adjusting the split between client and server based on available resources and privacy requirements.
- *Adaptive privacy mechanisms*: Investigate advanced techniques for dynamically adjusting privacy noise in DisLLM based on data sensitivity or the specific requirements of the clients.
- *Communication efficiency*: Improve data transfer mechanisms, focusing on optimizing transmission of large data tensors between clients and server in DisLLM. Explore compression techniques and adaptive protocols to reduce latency.
- *Privacy-preserving AI applications*: Explore the integration of DisLLM into various privacy-sensitive applications that require real-time inference like Voice Assistants, leveraging the framework's ability to keep sensitive user data securely on client devices.

## Acknowledgments

This research is partly supported by the European Union in CONFIDENTIAL-6G (Grant No: 101096435), and the Science Foundation Ireland under CONNECT phase 2 (Grant no. 13/RC/2077_P2) project.

## References

[1] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49313245

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[4] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2024.

[5] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," 2023.

[6] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8485–8493.

[7] 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," 2021.

[8] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," 2019.

[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. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'16. ACM, Oct. 2016. [Online]. Available: http://dx.doi.org/10.1145/2976749.2978318

[11] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," 2018.

[12] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[13] S. Augenstein, H. B. McMahan, D. Ramage, S. Ramaswamy, P. Kairouz, M. Chen, R. Mathews, and B. A. y Arcas, "Generative models for effective ml on private, decentralized datasets," 2020.

[14] I. Ceballos, V. Sharma, E. Mugica, A. Singh, A. Roman, P. Vepakomma, and R. Raskar, "Splitnn-driven vertical partitioning," 2020.

[15] P. Joshi, C. Thapa, S. Camtepe, M. Hasanuzzaman, T. Scully, and H. Afli, "Performance and information leakage in splitfed learning and multi-head split learning in healthcare data and beyond," 2022. [Online]. Available: https://www.mdpi.com/2409-9279/5/4/60

[16] Centers for Medicare & Medicaid Services, "The Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Online at http://www.cms.hhs.gov/hipaa/, 1996.

[17] "BothBosu/scam-dialogue · Datasets at Hugging Face." [Online]. Available: https://huggingface.co/datasets/BothBosu/scam-dialogue

[18] "Perverted and normal chats," 4 2021. [Online]. Available: https://www.kaggle.com/datasets/namangarg110/preverted-and-normal-chats

[19] "DBPedia Classes," 7 2019. [Online]. Available: https://www.kaggle.com/datasets/danofer/dbpedia-classes/data

[20] "UCI Machine Learning Repository." [Online]. Available: https://archive.ics.uci.edu/dataset/228/sms+spam+collection