# Communication-Efficient Training Workload Balancing for Decentralized Multi-Agent Learning

Seyed Mahmoud Sajjadi Mohammadabadi
*Department of Computer Science and Engineering*
*University of Nevada, Reno*
Reno, NV, USA
mahmoud.sajjadi@unr.edu

Lei Yang
*Department of Computer Science and Engineering*
*University of Nevada, Reno*
Reno, NV, USA
leiy@unr.edu

Feng Yan
*Department of Computer Science*
*University of Houston*
Houston, TX, USA
fyan5@central.uh.edu

Junshan Zhang
*Department of Electrical and Computer Engineering*
*University of California, Davis*
Davis, CA, USA
jazh@ucdavis.edu

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00069

***Abstract*—Decentralized Multi-agent Learning (DML) enables collaborative model training while preserving data privacy. However, inherent heterogeneity in agents' resources (computation, communication, and task size) may lead to substantial variations in training time. This heterogeneity creates a bottleneck, lengthening the overall training time due to straggler effects and potentially wasting spare resources of faster agents. To minimize training time in heterogeneous environments, we present a Communication-Efficient Training Workload Balancing for Decentralized Multi-Agent Learning (ComDML), which balances the workload among agents through a decentralized approach. Leveraging local-loss split training, ComDML enables parallel updates, where slower agents offload part of their workload to faster agents. To minimize the overall training time, ComDML optimizes the workload balancing by jointly considering the communication and computation capacities of agents, which hinges upon integer programming. A dynamic decentralized pairing scheduler is developed to efficiently pair agents and determine optimal offloading amounts. We prove that in ComDML, both slower and faster agents' models converge, for convex and non-convex functions. Furthermore, extensive experimental results on popular datasets (CIFAR-10, CIFAR-100, and CINIC-10) and their non-I.I.D. variants, with large models such as ResNet-56 and ResNet-110, demonstrate that ComDML can significantly reduce the overall training time while maintaining model accuracy, compared to state-of-the-art methods. ComDML demonstrates robustness in heterogeneous environments, and privacy measures can be seamlessly integrated for enhanced data protection.**

***Index Terms*—decentralized multi-agent learning, federated learning, edge computing, heterogeneous agents, workload balancing, communication-efficient training**

## I. Introduction

Effective training of Deep Neural Networks (DNNs) often requires access to a vast amount of data typically unavailable on a single device. Transferring data from different devices (a.k.a., clients or agents) to a central server for training raises security and privacy concerns, as well as communication costs and challenges. To address these issues, there is a growing trend towards cooperative training of machine learning models across a network of devices, eliminating the need to transfer the local training data. Federated Learning (FL) [1] algorithms have gained substantial attention as a privacy-preserving distributed learning paradigm. In FL, a central server acts as a coordinator among participating agents, enabling them to update a global model using their locally trained weights. The training process of FL, however, causes a major challenge when dealing with real-world resource-constrained devices (e.g., mobile/IoT devices and edge servers) that often exhibit heterogeneous computation and communication capacities, along with varying dataset sizes. Such heterogeneity not only introduces substantial variations in training time across agents, leading to the straggler problem (i.e., some devices significantly lag behind others) but also wastes the available spare resources of faster agents.

To address the challenges due to the unbalanced workload on resource-constrained devices, different methods have been proposed recently. One popular approach involves splitting the global model into an agent-side model (consisting of the initial layers of a global model) and a server-side model (the remaining layers), where agents only need to train the smaller agent-side model using Split Learning (SL) [2], [3]. However, SL requires agents to wait for backpropagated gradients from the server to update their models, resulting in substantial communication overhead in each training round. To address the latency and communication issues of SL, a federated SL algorithm is developed by incorporating local-loss-based training [4]. However, their approach uses fixed agent-side models, limiting their adaptability to varying computation and communication resources in dynamic environments. Along another line, agents can be segmented into tiers based on their training speed, and agents from the same tier are selected in each training round to mitigate the straggler problem [5], [6]. However, existing tier-based approaches [5], [6] require agents to train the entire global model locally, which is not scalable for training large models. For these methods [2]–[6], a central server is required to coordinate the training of all

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00069

agents. A centralized server, both prone to latency bottlenecks [7], [8] and susceptible to failures and targeted attacks [9], can significantly undermine the reliability of the entire distributed learning process. To mitigate these issues, decentralized (peer-to-peer) systems have emerged as an alternative [10]–[12]. Distinct from distributed systems that utilize a central server for coordination, these systems rely on peer-to-peer communication, offering improved resilience and security due to the absence of a single point of failure. However, without the coordination of a centralized scheduler, workload balancing in these decentralized systems becomes challenging.

In this paper, we propose a novel Communication-Efficient Training Workload Balancing for Decentralized Multi-Agent Learning (ComDML) that effectively addresses the challenges of training workload balancing in decentralized systems, operating without a server or coordinator. In ComDML, the training workload is balanced by allowing slower agents to offload a portion of their workload to faster agents, ensuring efficient utilization of available resources (see Fig. 1). To reduce synchronization and communication overhead between paired agents, ComDML employs local-loss-based split training, where the paired agents can determine how to split the model and then train the split model in parallel. ComDML's core objective is to minimize the overall training time. ComDML achieves it by employing an integer programming formulation that balances agent workloads based on both computation and communication capacities. Recognizing heterogeneous environments where agent computation and communication capabilities fluctuate, ComDML leverages a dynamic, decentralized pairing scheduler. This scheduler pairs agents and assigns workloads based on observed capabilities, ensuring efficient computation and communication in heterogeneous settings. The scheduler prioritizes pairing the slowest agents first by maintaining a shared list of training times. This list guides agents to pair up at each round, starting with the slowest, to minimize the overall training time. This pairing strategy employs lightweight, low-overhead local split model profiling, which quantifies the communication overhead (in terms of intermediate data size) for various pairing options. To make workload offloading decisions, slower agents consider both local profiling and the communication and computation capacities of faster agents, ensuring optimal workload balancing for the shortest total training time. In this way, the proposed pairing scheduler operates in a decentralized manner with minimal information exchange among agents.

Using standard assumptions in FL [13], [14] and local-loss-based training techniques [4], [15], we show the convergence of ComDML for both convex and non-convex functions. The convergence analysis is novel as it considers multiple split models for each paired agent in a heterogeneous environment. Using ComDML, we conduct training experiments on large models (ResNet-56 and ResNet-110 [16]) across various numbers of agents using popular datasets including CIFAR-10 [17], CIFAR-100 [17], and CINIC-10 [18], alongside their non-identical and independent distribution (non-I.I.D.) variants. Our extensive experimental results demonstrate that ComDML achieves a remarkable reduction in overall training time by up to 71% while maintaining model accuracy comparable to state-of-the-art methods. We also evaluate its performance under various privacy measures. These measures include minimizing distance correlation between raw data and intermediate representations, shuffling data patches, and applying differential privacy to model parameters. Our results demonstrate that ComDML can effectively incorporate these privacy techniques with minimal impact on model accuracy.

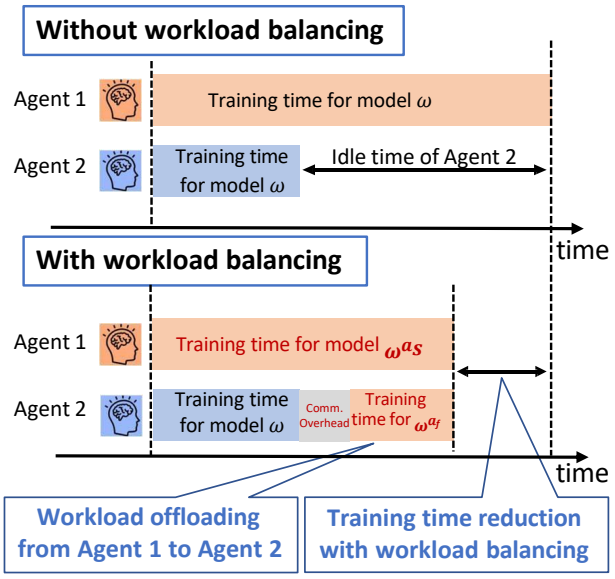


Fig. 1. Comparison of model training with and without workload balancing. Workload balancing reduces training time by offloading the workload from agent 1 to agent 2, which would otherwise be idle.

## II. Background and Related Work

### A. Collaborative Multi-agent Learning

Federated learning (FL) is a privacy-preserving machine learning technique that allows multiple parties (clients or agents) to collaboratively train a machine learning model without having to share their data [1]. This is achieved through frequent communication with a central server for model exchange and updates, facilitating the collaborative learning process [19]. State-of-the-art deep learning models (e.g., ResNet or AlexNet) have become increasingly large in recent years, which can make the computation and communication cost of FL prohibitive. The computational cost of resource-constrained agents and the communication overhead of FL can become significant challenges, particularly for large models [20]. [21] reduces communication overhead by transmitting smaller models, leading to faster training and lower resource consumption. Meanwhile, [22] employs SL to split the global model across agents in different tiers, resulting in faster training times. In a related context, [23] specifically addresses device selection for communication purposes. In addition to communication optimization, [24] studies resource allocation

strategies. Leveraging network pruning on client models, [25] improves inference performance, by reducing model size and communication volume. Notably, these algorithms require a central server and are not designed for decentralized systems. In contrast, ComDML does not require a central coordinator, and it addresses heterogeneity by workload balancing.

### B. Server-less and Peer-to-Peer Decentralized Learning

Existing FL methods (see a comprehensive study of FL [19]) have traditionally relied on a central server to manage agent selection, model broadcasting, aggregation, and updating. In these methods, agents are required to repeatedly download and update the global model and send it back to the server. However, such processes face limitations when training large models on resource-constrained devices in heterogeneous environments, leading to issues like the straggler problem. Additionally, they are susceptible to vulnerabilities stemming from potential failures of the central server and network bottlenecks. To address the straggler problem, [13] selects a smaller set of agents for training in each global iteration, but at the cost of increased training rounds. [26] deals with stragglers by ignoring the slowest 30% of agents, while FedProx [27] assigns different numbers of training rounds to agents. These approaches face the challenge of determining optimal parameters (i.e., percentage of slowest agents and number of local epochs).

To tackle the challenge of central server failures, [10] proposes a server-less framework for cross-silo FL, targeting scenarios with a relatively small number of agents with powerful communication and computation capabilities to enable sequential operation. Recent research has focused on fully decentralized approaches, such as the work by [28], which aims to achieve fully decentralized learning. Gossip learning, as introduced in [29], offers an alternative approach to serverless decentralized learning by enabling devices to exchange model updates with their neighbors. Building on this concept, [11] explores the application of gossip learning as a substitute for FL. GossipFL [12] introduces a sparsification algorithm to reduce agent communication to a single peer with a compressed model in a gossip learning setting. Another way to eliminate the central server is the integration of blockchain technology. For instance, [30] proposes a blockchain-based solution with committee consensus, while [31] investigates the utilization of blockchain in combination with Smart Contracts. However, it is important to note that these methods do not explicitly consider agent heterogeneity and require agents to train the entire global model. The learning performance may degrade a lot due to the straggler problem, particularly in scenarios where agents have varying computation/communication capabilities or limited resources. In this paper, we address these issues by developing a decentralized workload balancing algorithm that does not rely on a central server and can effectively pair agents in dynamic heterogeneous environments.

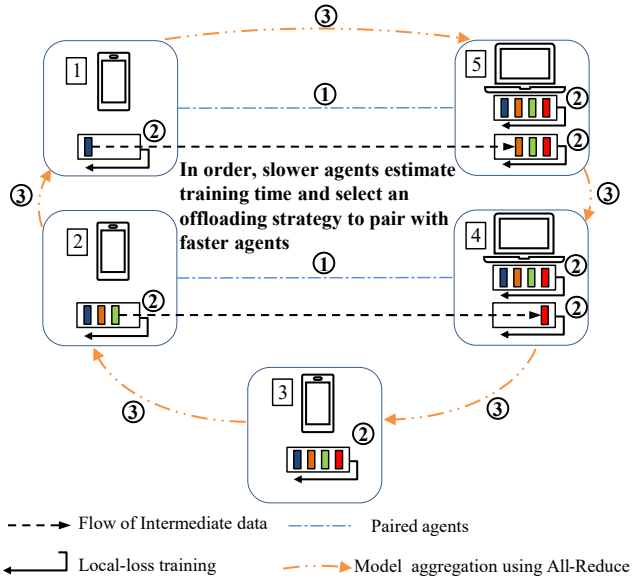


Fig. 2. Overview of the training process for ComDML. In each round, ComDML first pairs agents ①. The paired agents then carry out the local model update ②. After all agents complete the model update, ComDML uses a decentralized model aggregation to obtain the global model ③. In the above example, agents 1 and 2 offload work to faster agents 5 and 4, respectively, while agent 3 trains independently.

## III. Workload Balancing for Decentralized Multi-Agent Learning

This work focuses on DML systems as introduced by [32], where multiple agents collaborate on a learning task within a networked environment, absent a central coordinating server. Beyond applications in healthcare, mobile services, and vehicle networks, DML is fueling advancements in emerging areas like swarm robotics, smart cities, and the metaverse [7]. Within a DML system, multiple agents collaboratively train a global model using their local datasets and then synchronize with other agents to update the model. In practice, agents usually have heterogeneous computation resources as well as heterogeneous data, thus the training workload across agents can be highly different, which leads to very different training times. Therefore, faster agents may have to wait for slower agents (stragglers) for a long time to synchronize the model update. Such a bottleneck in synchronization may not only significantly increase the overall training time, but also result in a waste of the spare computation resources of faster agents.

To address this challenge, one promising solution is to balance the training workload based on the computation and communication resources by offloading the workload from slower agents to faster agents, to address the straggler problem. Fig. 2 illustrates this workflow. It is worth noting that the training workload of each agent is highly correlated with the local dataset size of each agent. When offloading the workload from slower agents to faster agents, faster agents still need to frequently communicate with slower agents to

exchange intermediate training information, and the amount of communication depends on how much workload is offloaded. Due to the heterogeneous communication capacity between agents, the communication overhead may offset the benefits of workload balancing. Therefore, optimal workload balancing requires jointly considering the heterogeneous communication and computation resources of agents. In this paper, we aim to develop a communication-efficient training workload balancing approach for collaborative multi-agent learning in a decentralized system.

### A. Decentralized Multi-agent Learning

Consider $K$ agents in a decentralized system (see Fig. 2), where $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N_i}$ denotes the dataset of agent $i$. Here, $\boldsymbol{x}_n$ represents the $n$th training sample, $y_n$ is the associated label, and $N_i$ is the number of samples in agent $i$'s dataset.

For convenience, define $f_i(\boldsymbol{w}) = \frac{1}{N_i}\sum_{n=1}^{N_i} \ell\left((\boldsymbol{x}_n, y_n); \boldsymbol{w}\right)$, with $\boldsymbol{w}$ being the model parameters. The DML problem can be formulated as a decentralized optimization problem:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \stackrel{\text{def}}{=} \min_{\boldsymbol{w}} \sum_{i=1}^{K} \frac{N_i}{N} \cdot f_i(\boldsymbol{w}) \tag{1}$$

where $N = \sum_{i=1}^{K} N_i$. $f(\boldsymbol{w})$ denotes the global objective function, and $f_i(\boldsymbol{w})$ represents the $i$th agent's local objective function, which measures the individual loss over its heterogeneous dataset using a loss function $\ell$. Each agent possesses its local data and collaborates with other agents to find the optimal $\boldsymbol{w}$ that minimizes the global objective (1).

Federated optimization techniques (e.g., [1], [27]) have been proposed to solve the problem (1). However, these FL methods encounter challenges in training models on resource-constrained devices, especially large models in heterogeneous environments. In particular, faster devices have to wait for straggler devices that take longer time to complete tasks, which would slow down the overall training process and waste the spare computation resources of faster devices. Furthermore, the central server, both a bottleneck and a prime target for potential attacks [33], poses a significant risk of disrupting the training process through failures or downtime. To address this challenge, we consider a DML system without a central server and enable workload balancing among agents.

### B. Workload Balancing via Local-loss based Split Training

To achieve efficient workload balancing for DML, we employ local-loss-based split training. Specifically, the model $\boldsymbol{w} = (\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{a_f^m})$ is split into two parts: a slow agent-side model $\boldsymbol{w}_i^{a_s^m}$ and a fast agent-side model $\boldsymbol{w}_i^{a_f^m}$. This split model allows slower agent $i$ to train only the slow agent-side model $\boldsymbol{w}_i^{a_s^m}$ and an auxiliary network $\boldsymbol{w}_i^{aux^m}$. The auxiliary network consists of additional layers connected to the slow agent-side model and is used to compute the local loss on the slow agent-side model. By incorporating the auxiliary network, we enable parallel model updates for each agent [4], avoiding the significant synchronization and communication overhead associated with split learning [2], which can significantly slow down the training process. In this paper, we adopt the approach of employing a few fully connected layers for the auxiliary network, following the approach in [4], [15]. Consider $M$ split models to determine how a slower agent splits the model for offloading to a faster agent. By offloading a portion of the model to the faster agent, the workload on the slower agent reduces, aiming to achieve equal training time for the paired agents.

Given the model $\boldsymbol{w}$, we define $f_i^{a_s^m}(\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{aux^m})$ as the loss function of the slow agent-side and $f_i^{a_f^m}(\boldsymbol{w}_i^{a_f^m}, \boldsymbol{w}_i^{a_s^m})$ as the corresponding fast agent-side loss function for the paired agents, where $m$ denotes how the slower agent splits the model to be offloaded to the faster agent. The goal of the slower agent $i$ is to find $\boldsymbol{w}_i^{a_s^{m\star}}$ and $\boldsymbol{w}_i^{aux^m}$ that minimize the loss function on the slow agent-side for each paired agent.

$$\min_{\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{aux^m}} f_i^{a_s^m}(\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{aux^m}) = \min_{\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{aux^m}} \frac{1}{N_i}\sum_{n=1}^{N_i} \ell\left((\boldsymbol{x}_n, y_n); (\boldsymbol{w}_i^{a_s^m}, \boldsymbol{w}_i^{aux^m})\right) \tag{2}$$

The faster agent $j$ shares its computation resources to simultaneously optimize $\boldsymbol{w}_j$ by minimizing local objective function and find $\boldsymbol{w}_i^{a_f^{m\star}}$ that minimizes (3):

$$\min_{\boldsymbol{w}_i^{a_f^m}} f_i^{a_f^m}(\boldsymbol{w}_i^{a_f^m}, \boldsymbol{w}_i^{a_s^{m\star}}) = \min_{\boldsymbol{w}_i^{a_f^m}} \frac{1}{N_i}\sum_{n=1}^{N_i} \ell\left((\boldsymbol{z}_n, y_n); (\boldsymbol{w}_i^{a_f^m}, \boldsymbol{w}_i^{a_s^{m\star}})\right) \tag{3}$$

where $\boldsymbol{z}_n$ denotes the intermediate output of the slow agent-side model $\boldsymbol{w}_i^{a_s^{m\star}}$ given the input $\boldsymbol{x}_n$. This process continues until all agents have completed their training tasks in a training round. At the end of each round, the models are aggregated using the AllReduce method [34] (see Sec. IV-B).

### C. Optimization for Workload Balancing

In a decentralized system with multiple agents, the objective of workload balancing is to minimize the overall training time. To this end, two key questions need to be addressed: 1) how to pair a faster agent with a slower agent for workload balancing, and 2) how much workload to offload when two agents are paired. Using the proposed local-loss-based split training for workload balancing, we need to jointly optimize the communication and computation time when addressing these two key questions.

Specifically, let $\gamma_{ij} \in \{0, 1\}$ denote the workload balancing decision, and $\gamma_{ij} = 1$ means offloading agent $i$'s workload $\boldsymbol{w}_i^{a_f^m}$ to agent $j$ by using split model $m$. The overall training time $\tau_i$ for agent $i$ in each round can be presented as:

$$\tau_i = \begin{cases} \tau_i(\boldsymbol{w}) + \sum_j \gamma_{ji} \left[ \tau_{ji}(\boldsymbol{w}_i^{a_f^m}) + \tau_i(\boldsymbol{w}_j^{a_f^m}) \right], \\ \text{if } \sum_j \gamma_{ij} = 0 \text{ (Agent } i \text{ does not offload workload)} \\ \tau_i(\boldsymbol{w}_i^{a_s^m}), \\ \text{if } \sum_j \gamma_{ij} = 1 \text{ (Agent } i \text{ offloads workload to one agent),} \end{cases} \tag{4}$$

where $\tau_i(\boldsymbol{w})$ denotes agent $i$'s computation time of learning the model $\boldsymbol{w}$. $\tau_{ij}(\boldsymbol{w}_i^{a_f^m})$ denotes the communication time when agent $i$ offloads its workload to agent $j$, which depends on the speed of communication link between agents $i$ and $j$, as well as the amounts of intermediate data based on $\boldsymbol{w}_i^{a_s^m}$. In (4), if agent $i$ does not offload workload, the overall training time consists of $\tau_i(\boldsymbol{w})$ and the computation and communication time for processing the workload of slower agents $\sum_j \gamma_{ji} \left[ \tau_i(\boldsymbol{w}_j^{a_f^m}) + \tau_{ji}(\boldsymbol{w}_i^{a_f^m}) \right]$ if any; if agent $i$ offloads workload, the overall training time consists of the computation time of learning a slow agent-side model $\tau_i(\boldsymbol{w}_i^{a_s^m})$. and the corresponding communication time $\sum_j \gamma_{ij} \tau_{ij}(\boldsymbol{w}_i^{a_f^m})$ with a faster agent.

The problem of joint optimization of communication and computation for workload balancing can be formulated as minimizing the training time of the slowest agent (i.e., straggler):

$$\min_{\{\gamma_{ij}\}, \{\boldsymbol{w}_i^{a_f^m}\}} \max_i \tau_i, \quad \text{s.t. } \gamma_{ij} \in \{0, 1\} \quad \forall i, j. \tag{5}$$

In (5), we need to jointly optimize the workload balancing decisions $\{\gamma_{ij}\}$ and the offloaded workload (i.e., how to split the model $\boldsymbol{w} = (\boldsymbol{w}_i^{a_f^m}, \boldsymbol{w}_i^{a_s^m})$ when $\gamma_{ij} = 1$). Note that problem (5) is an integer programming problem, which is challenging to solve in a decentralized learning system without any centralized scheduler.

## IV. Decentralized Workload Balancing

### A. Agent Pairing

To effectively address the optimization problem in (5), we require a pairing strategy that dynamically pairs agents based on their computation and communication capacities in each training round. This dynamic approach is crucial due to the inherent variability of agent capabilities within heterogeneous environments. Static pairing assignments can lead to significant straggler problems, as agents with limited resources may be paired together, or slow agents may remain unpaired. Such pairings can inadvertently increase the overall training time. To mitigate these issues, we propose a dynamic decentralized pairing scheduler. This scheduler dynamically pairs agents in a decentralized manner to minimize overall training time based on agents' computation and communication capacities.

As agents train their models in parallel, the training time in each round is determined by the slowest agent, denoted as $\max_i \tau_i$. This training time for the slowest agent becomes a critical factor, as other agents must wait for it before model aggregation can commence. To minimize the overall training time, our objective is to minimize the maximum training time among all agents, as expressed in (5). Leveraging a list of individual training times, updated and maintained by each agent, agents dynamically pair themselves each round, prioritizing the slowest agent first, to minimize the overall training time for each pair. In this approach, agents are paired or trained independently based on their resources and available neighbors. To ensure optimal pairings, the process meticulously considers the communication link speed, processing speed, and dataset size of a faster agent. This decentralized approach empowers each agent to independently implement the pairing scheduler, fostering scalability and resilience without reliance on a central coordinator. Information exchange is minimized, as agents only need to share their processing speeds and local dataset sizes with their neighbors. Individual training times are calculated based on these shared metrics, and network speeds can be directly observed.

### B. Training Workflow

The training process of ComDML in each round is described in the following steps, which are detailed in Algorithm 1 and illustrated in Fig. 2. This algorithm achieves remarkable resource optimization with minimal overhead, enabling agents to fully leverage spare resources for significant performance gains.

To facilitate the decentralized agent pairing, each agent locally conducts split model profiling prior to the training process. The split model profiling calculates the relative training time (i.e., the training time compared to the case where the model is not split) and intermediate data size for each split model $m$. Specifically, for $M$ different split models, each agent calculates the relative training time of the slow agent-side $T^{a_s^m}$, the fast agent-side $T^{a_f^m}$ and the intermediate data size $\nu^m$ of each split model $m$ using a batch with the same size. The profiling helps each agent to estimate the overall training time of each split model based on the actual size of the dataset when pairing.

① **Agent pairing.** In each round, slower agents initially pair up in order of their estimated training times. The agent pairing algorithm ensures that each pair minimizes their training time and completes their tasks within a similar time. Specifically, all available agents broadcast their processing speed $p_j$ and individual training time $\tau_j$ (i.e., time required to complete its task without workload offloading) to all connected agents in their network. Through a greedy algorithm (function **Pairing**$(\cdot)$), agents are paired in order of their individual training times. Starting with the agent with the longest training time, each agent selects a faster agent that can significantly reduce its training time by offloading part of its workload. This pair then informs the next agent in the list to pair up. This ensures that paired agents complete their training in similar time. Agent $i$ estimates the overall training time if it offloads its workload to agent $j$, using the **AgentTrainingTime**$(\cdot)$ function. This function factors in the processing speeds of both agents $p_i$ and $p_j$, agent $j$'s estimated individual training time $\hat{\tau}_j$, the network speed $c_{ij}$, and and the data transfer size during offloading. To estimate the training time for split

model $m$, agent $i$ utilizes the split model profile to convert $p_i$ and $p_j$ into the processing speeds on the slow agent-side $p_i^m$ and the fast agent-side $p_j^m$ of model $m$, respectively. Let $\tilde{N}_i$ denote the number of data batches of agent $i$. Agent $i$ estimates the time for different split model $m$ as follows: $\hat{\tau}_{ij}^m = \max\left(\frac{\tilde{N}_i}{p_i^m}, \hat{\tau}_j + \frac{\tilde{N}_i \nu^m}{c_{ij}} + \frac{\tilde{N}_i}{p_j^m}\right)$, where $\hat{\tau}_{ij}^m$ represents the estimated training time for agent $i$ when using split model $m$. This process is implemented in the **Pairing**$(\cdot)$ function in Algorithm 1.

② **Local model update.** Then, each pair of agents collaboratively perform the slower agent's task via local-loss-based split training (see Sec. III-B). Simultaneously, each faster agent also performs the model training using its local dataset.

③ **Model aggregation and update global model.** At the end of each round $r$, all agents participate in the AllReduce operation [34] to synchronize their models and obtain the average of all agents' models. Following the aggregation process (i.e., the **ModelAggregation**$(\cdot)$ function), all agents have the updated model parameters that represent the average of all $K$ agents. The AllReduce mechanism facilitates a key decentralized aspect of the aggregation process. It allows for the sharing and averaging of updated model weights among agents without the need for a centralized coordinator. Two well-known AllReduce algorithms suitable for bandwidth-limited scenarios are the recursive halving and doubling algorithm [35] and the ring algorithm [34]. In both algorithms, each agent sends and receives $2\frac{K-1}{K}b$ bytes of data, where $b$ represents the model size in bytes. The halving/doubling algorithm consists of $2\log_2(K)$ communication steps, while the ring algorithm involves $2(K-1)$ steps. Given that we are dealing with a large number of agents, we opt for the halving and doubling algorithms for the AllReduce operation. Other existing aggregation techniques (e.g., quantized gradients [36]) can also be integrated into the proposed training process to further reduce communication overhead.

**Algorithm 1 ComDML.**

**Initialize:** $R$ denotes the total global rounds, $T^{a_s^m}$ and $T^{a_f^m}$ denote the relative training time for the slow and fast agent-side sides, respectively, corresponding to model split $m$ with an intermediate data size of $\nu^m$, $\mathcal{A}$ is the list of descending order of agents by their task completion times, $\left[\hat{\tau}_{ij}^m\right]$ denotes the list of estimated training time.

**Main()**

**for** $r = 0$ to $R-1$ **do**
  Agents broadcast $p_j$ and $\hat{\tau}_j$ to all connected agents
  **for** agent $i$ in order $\mathcal{A}$ **do**
    **if** agent $i$ is not paired **then** $j^\star \leftarrow$ **Pairing**$(i)$
    **end if**
  **end for**
  **Local model update across agents in parallel**
  **ModelAggregation()** ▷ Decentralized aggregation
**end for**

**Pairing**$(i)$ ▷ Run on agent $i$

**for** all unpaired $j$ that are connected to $i$ **do**
// Estimate the training time of $i$ if it offloads to $j$
  $\hat{\tau}_{ij} \leftarrow$ **AgentTrainingTime**$(p_j, \hat{\tau}_j)$
**end for**
$j^\star \leftarrow \arg\min_j \left[\hat{\tau}_{ij}\right]$
**Return** $\gamma_{ij^\star}$

**AgentTrainingTime**$(p_j, \hat{\tau}_j)$ ▷ Run on agent $i$

**for** all split layer $m$ **do**
  $p_i^m \leftarrow \frac{p_i}{T^{a_s^m}}$
  $p_j^m \leftarrow \frac{p_j}{T^{a_f^m}}$
  $\hat{\tau}_{ij}^m \leftarrow \max\left(\frac{\tilde{N}_i}{p_i^m}, \hat{\tau}_j + \frac{\tilde{N}_i \nu^m}{c_{ij}} + \frac{\tilde{N}_i}{p_j^m}\right)$
**end for**
$\hat{\tau}_{ij} \leftarrow \min_m \left[\hat{\tau}_{ij}^m\right]$
$m^\star \leftarrow \arg\min_m \left[\hat{\tau}_{ij}^m\right]$
**Return** $\hat{\tau}_{ij}$

### C. Privacy Protection

While ComDML excels in reducing training time, it addresses privacy concerns arising from model intermediate data exchange. To mitigate model parameter attacks that aim to replicate models using dummy data inputs [37], ComDML restricts agents' access to external datasets, query services, and dummy data itself, effectively shielding sensitive information from potential adversaries. Furthermore, ComDML's model split architecture, solidified by AllReduce aggregation, inherently counters model inversion attacks by compartmentalizing model updates between slow and fast agents. This architectural design restricts model visibility, aligning with research that suggests such attacks often require full model access to succeed [38].

While ComDML's architecture inherently limits privacy leakage, it acknowledges potential vulnerabilities to strong eavesdropping attacks. To address this, it offers a versatile framework that seamlessly integrates with diverse privacy-preserving techniques: *i*) **Fast agents privacy**: Inherently protected through unidirectional communication (i.e., from slow agents to fast agents), fast agent updates remain confidential. For further privacy guarantees during model aggregation, techniques like differential privacy [39] and cryptography [40] can be integrated. *ii*) **Slow agents privacy**: ComDML prioritizes slow agent privacy with a customizable toolkit. Techniques like PixelDP noise layer [41], patch shuffling [42], distance correlation [43], and SplitGuard [44] directly shield intermediate data, while differential privacy or cryptography secure model aggregation. This flexibility empowers slow agents to tailor their protection, balancing privacy and performance.

### D. Convergence Analysis

We establish the convergence of slow and fast agent-side models, considering both convex and non-convex loss functions under standard assumptions. This is achieved through local-loss-based training adapted from [15], where input distributions for fast agents dynamically evolve based on the convergence of their slow counterparts. We define $A^{m,r}$ and $\mathcal{A}^{m,r}$ as the number and the set of agents with split model

$m$ at round $r$, respectively. The output of the slow agent-side model, $\boldsymbol{z}_n^{a_s^m,r}$, follows the density function $d^{a_s^m,r}$, where the converged density of the slow agent-side is represented as $d^{a_s^m,\star}$. We define $c^{a_s^m,r} \triangleq \int \left| d^{a_s^m,r}(\boldsymbol{z}) - d^{a_s^m,\star}(\boldsymbol{z}) \right| d\boldsymbol{z}$ as the distance between the density function of the output of the slow agent-side model and its converged state. In the following, we introduce the standard assumptions used in the analysis.

*Assumption 1 (L-smoothness):* The loss function $f$ is differentiable and $L$-smooth, i.e., $\|\nabla f_i(\boldsymbol{w}) - \nabla f_i(\boldsymbol{v})\| \leq L\|\boldsymbol{w} - \boldsymbol{v}\|$, $\forall\ f_i,\ \boldsymbol{w},\ \boldsymbol{v}$.

*Assumption 2 ($\mu$-convex):* $f_i$ is $\mu$-convex for $\mu \geq 0$ and satisfies: $f_i(\boldsymbol{w}) + (\boldsymbol{v} - \boldsymbol{w})^T \nabla f_i(\boldsymbol{w}) + \frac{\mu}{2}\|\boldsymbol{v} - \boldsymbol{w}\|^2 \leq f_i(\boldsymbol{v}), \forall f_i, \boldsymbol{w}, \boldsymbol{v}$.

*Assumption 3 (Bounded gradients):* Expected squared norms of gradients have upper bounds: $\mathbb{E}\left\|\nabla f_i(\boldsymbol{w})\right\|^2 \leq G_1^2, \forall f_i, \boldsymbol{w}$.

*Assumption 4 (Bounded variance):* The variance of stochastic gradients in each agent is bounded: $\mathbb{E}[\|\nabla f_i(\zeta_i^r, \boldsymbol{w}) - \nabla f_i(\boldsymbol{w})\|^2] \leq \sigma^2, \forall f, \boldsymbol{w}$, where $\zeta_i^r$ sampled from $k$-th agent local dataset.

*Assumption 5 (Bounded gradient dissimilarity):* For both slow and fast agent-sides and all split models, there are constants $G_2 \geq 0$; $B \geq 1$ such that $\frac{1}{K}\sum_{i=1}^{K}\left\|\nabla f_i(\boldsymbol{w})\right\|^2 \leq G_2^2 + B^2\|\nabla f(\boldsymbol{w})\|^2, \forall \boldsymbol{w}$.

If $\{f_i\}$ are convex, we can relax the assumption to $\frac{1}{K}\sum_{i=1}^{K}\left\|\nabla f_i(\boldsymbol{w})\right\|^2 \leq G_2^2 + 2LB^2\left(f(\boldsymbol{w}) - f^\star\right), \forall \boldsymbol{w}$.

*Assumption 6 (Bounded distance):* The time-varying parameter satisfies $\sum_r c^{a_s^m,r} < \infty$, $\forall m$.

Assumptions 1 to 6 have been widely employed in the literature for convergence analysis of machine learning (see [13]–[15] and the references therein). Under these standard assumptions, we establish the convergence properties of ComDML. The proof of Theorem 1 is given in the Appendix.

*Theorem 1 (Convergence of ComDML):* Suppose that $f^{a_s^m}$ and $f^{a_f^m}$ satisfy Assumptions 1, 3, 4, 5, and 6. The convergence properties of ComDML for both convex and non-convex functions are summarized as follows:

- **Convex:** If both $f^{a_s^m}$ and $f^{a_f^m}$ are $\mu$-convex with $\mu > 0$, $\eta \leq \frac{1}{8L(1+B^2)}$ and $R \geq \frac{4L(1+B^2)}{\mu}$, then the slow agent-side model converges at the rate of $\mathcal{O}\left(\frac{H_1^2}{RA^m} + D^2\exp\left(-R\right)\right)$ and the fast agent-side model converges at the rate of $\mathcal{O}\left(\frac{H_2\sqrt{F^{a_f^m 0}}}{\sqrt{RA^m}} + \frac{C_1 + F^{a_f^m 0}}{R}\right)$.
- **Non-convex:** If both $f^{a_s^m}$ and $f^{a_f^m}$ are non-convex with $\eta \leq \frac{1}{8L(1+B^2)}$, then the slow agent-side model converges at the rate of $\mathcal{O}\left(\frac{H_1\sqrt{F^{a_s^m 0}}}{\sqrt{RA^m}} + \frac{F^{a_s^m 0}}{R}\right)$ and the fast agent-side model converges at the rate of $\mathcal{O}\left(\frac{H_2\sqrt{F^{a_f^m 0}}}{\sqrt{RA^m}} + \frac{C_2 + F^{a_f^m 0}}{R}\right)$.

where $H_1$, $H_2$, $D$, $F^{a_s^m}$, $F^{a_f^m}$, and $A^m$ are constants whose definitions are provided in the Appendix for reference. Demonstrated slow agent convergence propagates to fast agents through $C_1$ and $C_2$, both exhibiting convergence.

We demonstrate that under standard assumptions, ComDML exhibits convergence for both convex and non-convex functions as the number of training rounds $R$ increases. This convergence behavior holds for both slow and fast agent-side models, albeit with potentially different convergence rates for fast and slow agent sides. It's crucial to note that ComDML's reliance on local-loss-based split training renders the convergence of the fast agent-side model contingent upon the convergence of the slow agent-side model. This dependence is explicitly characterized by constants $C_1$ and $C_2$ within the analysis.

## V. Experimental Evaluation

### A. Experimental Setup

**Dataset.** We conduct image classification experiments using three publicly available datasets: CIFAR-10 [17], CIFAR-100 [17], and CINIC-10 [18]. We also consider label distribution skew (i.e., how the distribution of labels varies across agents) to generate non-I.I.D. variants of these datasets. To maintain fairness, we used a fixed Dirichlet distribution (concentration parameter = 0.5) for the non-I.I.D. datasets. Global model performance is assessed using test images after each round.

**Baselines.** To the best of our knowledge, this study pioneers the introduction of workload balancing in server-less DML. Although FL methods like FedProx [27] aim to enhance performance in heterogeneous environments, they rely on a central server, incompatible with our focus on serverless decentralized machine learning. Therefore, we primarily compare ComDML with decentralized baselines: BrainTorrent [10], Gossip Learning [11], and decentralized AllReduce [34]. FedAvg [1], though server-dependent, is included as a baseline for comparing workload balancing. BrainTorrent is a peer-to-peer framework where agents take turns acting as the server and updating the global model. Gossip Learning [11] incorporates model averaging, enabling each agent to update its model based on information received from neighboring agents. In decentralized learning utilizing AllReduce aggregation, agents update their models independently and then employ AllReduce to aggregate them, eliminating the need for a central server.

**Implementation.** We conducted the experiment using Python 3.11.3 and the PyTorch library version 1.13.1. The source code is publicly available online at ComDML's GitHub repository [45]. ComDML and baseline models were deployed on a server with the following specifications: dual-socket Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, four NVIDIA GeForce GTX 1080 Ti GPUs, and 64 GB of memory. To replicate the heterogeneity of real-world systems, we designed a heterogeneous simulation environment where agents possess distinct computational and communication capabilities. Each agent is equipped with a simulated CPU and communication resources, mirroring varied computation and communication times. This setup effectively captures the complexities of real-world distributed systems, where agents often differ in their processing power and network connectivity. We simulated agents with CPU profiles spanning 4, 2, 1, 0.5, and 0.2 CPUs. Communication profiles were varied with 0, 10, 20, 50, and 100 Mbps links, with 0 representing disconnected agents. At the beginning of training, each agent was assigned

TABLE I

PERFORMANCE OF 2-AGENT DECENTRALIZED TRAINING WITH VARYING LAYER OFFLOADING. THE EVALUATION COMPARES THE FAST AGENT TRAINING TIME, COMMUNICATION TIME, COMBINED IDLE TIME OF AGENTS, AND TOTAL TRAINING TIME OF THE PROCESS (ALL IN SECONDS) REQUIRED TO ACHIEVE 90% ACCURACY ON CIFAR-10 USING THE RESNET-56.

| Layers Offloaded | Times in the 1st Setting (s) | | | | Times in the 2nd Setting (s) | | | |
|---|---|---|---|---|---|---|---|---|
| | Train | Comm. | Idle | Total | Train | Comm. | Idle | Total |
| 0 | 5573 | 34 | 14489 | 20096 | 5578 | 17 | 3560 | 9165 |
| 1 | 5781 | 655 | 14472 | 20909 | 5856 | 327 | 2966 | 9150 |
| 10 | 6740 | 3532 | 4787 | 15059 | 6364 | 1766 | 350 | 8481 |
| 19 | 7625 | 3544 | 1682 | 12851 | 6547 | 1772 | **137** | **8456** |
| 28 | 7906 | 1261 | 2049 | 11217 | 7859 | 630 | 3368 | 8490 |
| 37 | 8003 | 1265 | **84** | **9352** | 8275 | 632 | 7318 | 8908 |
| 46 | 8939 | 611 | 5042 | 9551 | 9334 | 305 | 8351 | 9640 |
| 55 | 10343 | 640 | 9833 | 10983 | 10101 | 320 | 9964 | 10421 |

TABLE II

COMPARISON OF TOTAL TRAINING TIME TO BASELINES WITH 10 AGENTS ON DIFFERENT DATASETS. THE NUMBER INDICATES THE TRAINING TIME (IN SECONDS) REQUIRED TO REACH THE TARGET ACCURACY. THE TARGET ACCURACIES ARE AS FOLLOWS: CIFAR-10 I.I.D. 90%, CIFAR-10 NON-I.I.D. 85%, CIFAR-100 I.I.D. 65%, CIFAR-100 NON-I.I.D. 60%, CINIC-10 I.I.D. 75%, AND CINIC-10 NON-I.I.D. 65%.

| Method | CIFAR-10 | | CIFAR-100 | | CINIC-10 | |
|---|---|---|---|---|---|---|
| | I.I.D. | non-I.I.D. | I.I.D. | non-I.I.D. | I.I.D. | non-I.I.D. |
| ComDML | **7211** | **4177** | **5589** | **8104** | **10229** | **17208** |
| Gossip Learning | 20337 | 15269 | 15262 | 28621 | 24636 | 56325 |
| BrainTorrent | 24639 | 14323 | 18046 | 25867 | 31992 | 51144 |
| AllReduce | 25153 | 13859 | 18462 | 26623 | 32652 | 53265 |
| FedAvg | 24174 | 13095 | 17630 | 25113 | 30601 | 49624 |

one profile representing its initial computation and communication resources. These resource profiles could dynamically shift throughout the training process, mimicking real-world variations in agents' resources. In all experiments, we consider simulated communication overhead for training time, including intermediate data transfer and model size. BrainTorrent operates through independent model updates followed by aggregation by a randomly selected agent. Gossip learning implementation aligns with [11]. For AllReduce experiments, independent model training is followed by aggregation via AllReduce.

**Model Architecture.** ComDML can effectively support various models, from Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) to large language models (LLMs) like BERT. In our experiments, we used two prominent CNN models: ResNet-56 and ResNet-110 [16], which have demonstrated good accuracy on the experimented datasets. To balance computational demands across agents, we partition the global model at varying split layers $m$. Slow agents locally update the model up to their designated layer $m$, while a fast agent handles the remaining layers. To facilitate local loss training within slow agents, we introduce an auxiliary network comprising a fully connected layer and an average pooling layer. The input dimension of the f.c. layer is adjusted to match the output of each slow agent-side model.

**Hyper-parameters.** We used the Stochastic Gradient Descent (SGD) optimizer with a momentum of 0.9 for all datasets. The initial learning rate (i.e., $\eta_0$) was set to 0.001 for all datasets. Upon the accuracy reached a plateau, the learning rate was reduced by a factor of 0.2 when there were 10 agents. For scenarios with 20, 50, and 100 agents, we implemented a reduction factor of 0.5. The local batch size for each agent was set to 100, and the local epoch was consistently set to one for all experiments.

### B. Experimental Results

*1) Impact of heterogeneity on workload offloading decisions:* Table I compares the performance of local-loss-based split training between two agents, using various portions of the model offloaded from a slower agent to a faster agent in two settings: 1) one agent with 2 CPUs and one agent with 0.25 CPU, with a communication speed of 50 Mbps, and 2) one agent with 2 CPUs and one agent with 1 CPU, with a communication speed of 100 Mbps. We evaluate training time for the CIFAR-10 dataset's classification task, targeting 90% accuracy. Table I presents the fast agent's training time, communication time, the combined idle time of both agents and the overall training time under different workload offloading decisions. The results reveal the significant impact of heterogeneous computation and communication resources on the optimal workload offloading and the total training time. In ComDML, offloading 0 layers signifies performing the training task independently without assistance.

Offloading the training workload from the slower agent to the faster agent can effectively reduce the total training time, as demonstrated in Table I. For instance, in the 1st setting, offloading 37 layers of the model to the faster agent resulted in a significant 53% decrease in overall training time compared to the scenario without workload offloading. Note that as an agent offloads more layers, the model size on its side decreases, reducing the computational workload. However, this offloading of more layers may increase the data transmitted (i.e., the size of the intermediate data and partial model). As shown in Table I, the optimal number of layers to offload is non-trivial, as it depends on various factors, such as the communication link speed between agents, the computation power of each agent, and the workload offloading strategy. Hence, dynamically pairing agents with suitable offloading strategies during the training process is of paramount importance to achieve substantial reductions in total training time.

*2) Training time improvement against baselines:* In Table II, we compare the training time of ComDML with baselines by training a ResNet-56 with 10 agents on heterogeneous agents with diverse computation and communication capacities. We created these heterogenous agents by randomly assigning 20% of the agents to each CPU and communication speed profile combination. All agents participated in the entire training process. To better simulate a dynamic environment, we randomly changed the profile of 20% of the agents after

TABLE III
PERFORMANCE EVALUATION OF TRAINING TIME (IN SECONDS) WITH VARYING NUMBERS OF AGENTS ON I.I.D. CIFAR-10 DATASET AND COMPARISON WITH OTHER BASELINES FOR RESNET-56 AND RESNET-110 MODELS TO REACH A TARGET ACCURACY OF 80%.

| Model | Agents | Training Method | | | | |
|---|---|---|---|---|---|---|
| | | ComDML | Gossip L. | BrainTorrent | AllReduce | FedAvg |
| ResNet-56 | 20 | **7618** | 12637 | 14822 | 15660 | 14409 |
| | 50 | **9539** | 17716 | 20337 | 21339 | 19681 |
| | 100 | **10461** | 19465 | 22825 | 23652 | 22577 |
| ResNet-110 | 20 | **11799** | 18834 | 20234 | 19559 | 19322 |
| | 50 | **15014** | 25574 | 27753 | 28117 | 27191 |
| | 100 | **15843** | 28825 | 31526 | 30085 | 29494 |

100 rounds.

The corresponding training times for each method to achieve specific target accuracies are provided in Table II. Notably, ComDML consistently demonstrates significant reductions in training time compared to the baselines, while preserving model accuracy across all scenarios. For example, ComDML achieves a remarkable 70% reduction in training time compared to FedAvg and a substantial 71% reduction compared to BrainTorrent on the I.I.D. CIFAR-10 dataset. Unlike FedAvg, which can be hampered by communication delays with a central server, ComDML eliminates this bottleneck by enabling direct peer-to-peer communication. This not only accelerates model updates but also strengthens robustness against server failures and network disruptions, ultimately enhancing the overall learning process.

*3) Performance of ComDML with different numbers of agents:* To evaluate the scalability of ComDML, we assessed its performance across varying numbers of agents. Table III shows the training time of ComDML in comparison with baselines on the I.I.D. CIFAR-10 dataset, using different numbers of agents to achieve a target accuracy of 80% for both ResNet-56 and ResNet-110 models. We employed a 20% sampling rate for agent participation in each training round. As observed from Table III, increasing the number of agents does not negatively impact ComDML's performance, underscoring its robust scalability.

*4) Integration of privacy protection methods:* ComDML smoothly integrates privacy-preserving methods with minimal overhead, effectively maintaining model accuracy. Our experiments on the CIFAR-10 dataset, employing ResNet-56 with 100 agents, demonstrate its ability to integrate privacy-preserving techniques without compromising either accuracy or training time. Remarkably, we obtained model accuracies of 81.7% with distance correlation ($\alpha = 0.5$) [43], 83.2% with patch shuffling [42], and 77.6% with differential privacy (using Laplace noise, $\epsilon = 0.5$, $\delta = 10^{-5}$) [39] over experiments with 100 rounds (Other configuration details mirrored those in the referenced papers). These results compellingly showcase ComDML's flexibility in balancing privacy and performance without sacrificing efficiency.

*5) ComDML's performance with different network topologies:* ComDML's decentralized design, adaptable to diverse network structures from random to ring topologies [7], even copes with limited connections. It dynamically adjusts communication strategies for efficiency, outperforming centralized methods like FedAvg that are vulnerable to server bottlenecks. By meticulously balancing communication overhead against computation, ComDML achieves faster overall training times compared to other baselines. It even adapts to extreme scenarios with poor links, allowing independent training if needed. This flexibility renders ComDML efficient across a wider range of network conditions than its counterparts.

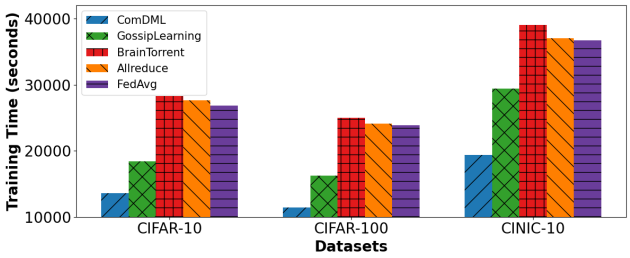


Fig. 3. Comparison of total training time (in seconds) against baseline models under a 20% link connectivity in random topology.

Fig. 3 highlights ComDML's training efficiency compared to baselines under simulated limited communication links, where agents are randomly connected through only 20% of the links present in a full graph. This setting mirrors those used in the 50-agent experiments across diverse I.I.D. datasets. ComDML's decentralized design maintains its efficiency even in scenarios with network disruptions or bandwidth limitations, ensuring persistent learning progress.

## VI. CONCLUSION

In this paper, we developed ComDML, an effective solution to address the challenges of collaboratively training large models in DML systems with diverse computational resources, communication bandwidth, and dataset sizes. This serverless framework enables slower agents to offload workloads to faster ones. By employing local-loss-based split learning, ComDML balances workloads, facilitating parallel updates and conquering resource constraints and straggler issues. We provided theoretical guarantees for the convergence of ComDML. Through extensive experiments on different datasets with heterogeneous agents, ComDML demonstrates remarkable reductions in training time without compromising model accuracy. It seamlessly integrates privacy-preserving techniques without sacrificing speed, ensuring both efficiency and confidentiality. Importantly, ComDML achieves these results without relying on a central server, thus offering a more scalable and efficient alternative to state-of-the-art DML methods.

## ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under Grants OIA-2148788, CAREER-2305491, CNS-2203239, CNS-2203412, and CCSS-2203238.

## APPENDIX

### A. Proof of Theorem 1

We show the convergence for both slow and fast agent sides, encompassing convex and non-convex functions. Slow agent-sides converge independently, while fast agent-sides rely on slow agent convergence for their own convergence, with rates presented. Due to space, we show only the convergence for convex functions. The convergence for non-convex functions can be easily derived using the same techniques presented in the following.

First, we introduce a lemma that will be useful for proving the convergence of the fast agent-side later on.

*Lemma 1:* Fubini's theorem, in conjunction with Assumption 6, enables us to make the following observation:

$$\sum_r c^{a_s^m,r} = \sum_r \int \left| d^{a_s^m,r}(\boldsymbol{z}) - d^{a_s^m,\star}(\boldsymbol{z}) \right| = \int \sum_r \left| d^{a_s^m,r}(\boldsymbol{z}) - d^{a_s^m,\star}(\boldsymbol{z}) \right| d\boldsymbol{z} < \infty.$$

Since $\sum_r \left| d^{a_s^m,r}(\boldsymbol{z}) - d^{a_s^m,\star}(\boldsymbol{z}) \right|$ is convergent, we have $\left| d^{a_s^m,r}(\boldsymbol{z}) - d^{a_s^m,\star}(\boldsymbol{z}) \right| \to 0$.

*1) Slow agent-side convergence:* Assume that the slow agent-side functions satisfy Assumptions 1, 2, 4, and 5. Considering the model update at round $r$, we have:

$$\Delta \boldsymbol{w}^{a_s^m} = -\frac{\eta}{A^m} \sum_{i \in \mathcal{A}^m} g_k^{a_s^m}\left(\boldsymbol{w}_k^{a_s^m}\right)$$
$$\Rightarrow \mathbb{E}[\Delta \boldsymbol{w}^{a_s^m}] = -\frac{\eta}{K} \sum_i \mathbb{E}\left[\nabla f_k^{a_s^m}\left(\boldsymbol{w}_k^{a_s^m}\right)\right].$$

We use the notation $\mathbb{E}$ to denote the expectation over all the randomness generated in the prior round. Building on the above, we separate the mean and variance by applying Lemma 4 of [14].

$$\mathbb{E}\left\| \boldsymbol{w}^{a_s^m} + \Delta \boldsymbol{w}^{a_s^m} - \boldsymbol{w}^{a_s^m\star} \right\|^2 \leq \left\| \boldsymbol{w}^{a_s^m} - \boldsymbol{w}^{a_s^m\star} \right\|^2 \underbrace{-\frac{2\eta}{K} \sum_i \left\langle \nabla f_k^{a_s^m}\left(\boldsymbol{w}_k^{a_s^m}\right), \boldsymbol{w}^{a_s^m} - \boldsymbol{w}^{a_s^m\star} \right\rangle}_{\mathcal{T}_1} + \underbrace{\eta^2 \mathbb{E}\left\| \frac{1}{A^{m,r}} \sum_{i \in \mathcal{A}^{m,r}} \nabla f_k^{a_s^m}\left(\boldsymbol{w}_k^{a_s^m}\right) \right\|^2}_{\mathcal{T}_2} + \frac{\eta^2 \sigma^2}{A^{m,r}}. \tag{6}$$

By applying Lemma 5 of [14] to $\mathcal{T}_1$, it is observed that:

$$\mathcal{T}_1 \leq -2\eta \left( f^{a_s^m}(\boldsymbol{w}^{a_s^m}) - f^{a_s^m}\left(\boldsymbol{w}^{a_s^m\star}\right) + \frac{\mu}{4} \left\| \boldsymbol{w}^{a_s^m} - \boldsymbol{w}^{a_s^m\star} \right\|^2 \right).$$

The bound on $\mathcal{T}_2$ is obtained by combining the relaxed triangle inequality, Lemma 3 of [14], and Assumption 5 as follows:

$$\mathcal{T}_2 \leq 8\eta^2 L \left(B^2+1\right) \left( f^{a_s^m}(\boldsymbol{w}^{a_s^m}) - f^{a_s^m}\left(\boldsymbol{w}^{a_s^m\star}\right) \right) + \left(1 - \frac{A^{m,r}}{K}\right) \frac{4\eta^2}{A^{m,r}} G_2^2.$$

After plugging back the bounds $\mathcal{T}_1$ and $\mathcal{T}_2$ into (6), we rearrange and move the expression $(f^{a_s^m}(\boldsymbol{w}^{a_s^m}) - f^{a_s^m}(\boldsymbol{w}^{a_s^m\star}))$ and then divide throughout by $\eta$, while assuming $4L\eta(B^2+1) \leq 1$.

$$f^{a_s^m}(\boldsymbol{w}^{a_s^m,r}) - f^{a_s^m}(\boldsymbol{w}^{a_s^m,\star}) \leq \frac{1}{\eta}\left(1 - \frac{\eta\mu}{4}\right) \left\| \boldsymbol{w}^{a_s^m,r} - \boldsymbol{w}^{a_s^m,\star} \right\|^2 - \frac{1}{\eta} \left\| \boldsymbol{w}^{a_s^m,r+1} - \boldsymbol{w}^{a_s^m,\star} \right\|^2 + \eta \left( \frac{\sigma^2}{A^{m,r}} + \frac{4G_2^2}{A^{m,r}} \left(1 - \frac{A^{m,r}}{K}\right) \right).$$

Applying the linear convergence rate lemma (Lemma 1 of [14]), we obtain the desired convergence rate for the convex case.

$$\mathbb{E}\left[ f^{a_s^m}\left(\overline{\boldsymbol{w}^{a_s^m}}^R\right) \right] - f^{a_s^m}\left(\boldsymbol{w}^{a_s^m\star}\right) = \mathcal{O}\left( \frac{\eta H_1^2}{\mu R A^{m,r}} + \mu D^2 \exp\left( -\frac{\mu}{L(1+B^2)} R \right) \right),$$

where $H_1^2 := \sigma^2 + \left(1 - \frac{A^m}{K}\right) G_2^2$, $D := \left\| \boldsymbol{w}^{a_s^m 0} - \boldsymbol{w}^{a_s^m\star} \right\|$, and $A^m = \min_r\{A^{m,r} > 0\}$.

The proof for the non-convex case follows a similar approach, with the key difference that Assumption 2 is disregarded. It relies on the sub-linear convergence rate lemma (Lemma 2 of [14]). This leads to:

$$\mathbb{E}\left[ \left\| \nabla f^{a_s^m}\left(\overline{\boldsymbol{w}^{a_s^m}}^R\right) \right\|^2 \right] = \mathcal{O}\left( \frac{LH_1\sqrt{F^{a_s^m}}}{\sqrt{RA^{m,r}}} + \frac{B^2 L F^{a_s^m}}{R} \right),$$

where $F^{a_s^m} := f^{a_s^m}\left(\boldsymbol{w}^{a_s^m 0}\right) - f^{a_s^m\star}$.

*2) Fast agent-side convergence:* Now we provide convergence analysis of fast agent-side models. Assume that the fast agent-side functions satisfy Assumptions 1, 2, 4, and 5. The model's update adheres to the following equation:

$$\Delta \boldsymbol{w}^{a_f^m} = -\frac{\eta}{A^{m,r}} \sum_{i \in \mathcal{A}^{m,r}} g_k^{a_f^m}\left(\boldsymbol{w}_k^{a_f^m}\right)$$
$$\Rightarrow \mathbb{E}[\Delta \boldsymbol{w}^{a_f^m}] = -\frac{\eta}{K} \sum_i \mathbb{E}\left[\nabla f_k^{a_f^m}\left(\boldsymbol{w}_k^{a_f^m}\right)\right].$$

By quadratic upper bound of Assumption 1 (L-smoothness), we have:

$$f^{a_f^m}(\boldsymbol{w}^{a_f^m,r+1}) \leq f^{a_f^m}(\boldsymbol{w}^{a_f^m,r}) + \frac{L}{2}\|\boldsymbol{w}^{a_f^m,r+1} - \boldsymbol{w}^{a_f^m,r}\|^2 + \nabla f^{a_f^m}(\boldsymbol{w}^{a_f^m,r})^T(\boldsymbol{w}^{a_f^m,r+1} - \boldsymbol{w}^{a_f^m,r}).$$

We incorporate the weight update formula into the above inequality, followed by taking expectation across all randomness. This process yields the following:

$$\mathbb{E}\left[f^{a_f^m}(\boldsymbol{w}^{a_f^m,r+1})\right] \leq \mathbb{E}\left[f^{a_f^m}(\boldsymbol{w}^{a_f^m r})\right] \underbrace{-\eta\mathbb{E}\left[\nabla f^{a_f^m}(\boldsymbol{w}^{a_f^m,r})^T\left(\frac{1}{A^{m,r}}\sum_{\boldsymbol{A}^{m,r}}\nabla f_k^{a_f^m}\left(\boldsymbol{u}^{a_s^m,r};\boldsymbol{w}^{a_f^m,r}\right)\right)\right]}_{\mathcal{T}_3} + \frac{L}{2}\eta^2\underbrace{\mathbb{E}\left[\left\|\frac{1}{A^{m,r}}\sum_{\boldsymbol{A}^{m,r}}\nabla f_k^{a_f^m}\left(\boldsymbol{u}^{a_s^m,r};\boldsymbol{w}^{a_f^m,r}\right)\right\|^2\right]}_{\mathcal{T}_4}.$$

We now prove the boundedness of $\mathcal{T}_3$ and $\mathcal{T}_4$. By applying Cauchy-Schwartz and Jensen's inequality, and leveraging Assumption 2 along with Lemma 1, we obtain the following bound for $\mathcal{T}_3$:

$$\mathcal{T}_3 \leq \eta\sqrt{2G_1^2(G_2^2+2LB^2(f^{a_f^m}(\boldsymbol{w}^{a_f^{m,r}})-f^{a_f^m\star}))c^{a_s^m,r}} - \eta\mathbb{E}\left[\left\|\nabla f^{a_f^m}(\boldsymbol{w}^{a_f^{m,r}})\right\|^2\right].$$

Leveraging Assumption 2 and mirroring the approach for the slow agent-side function, we establish the bound for $\mathcal{T}_4$:

$$\mathcal{T}_4 \leq 8\eta^2 L\left(B^2+1\right)\left(f_k^{a_f^m}(\boldsymbol{w}^{a_f^m})-f_k^{a_f^m}\left(\boldsymbol{w}^{a_f^m\star}\right)\right) + \left(1-\frac{A^{m,r}}{K}\right)\frac{4\eta^2}{A^{m,r}}G_2^2.$$

By using the bounds of $\mathcal{T}_3$ and $\mathcal{T}_4$ and assuming $4L\eta(B^2+1)\leq 1$, we have:

$$\mathbb{E}\left[\left\|\nabla f^{a_f^m}(\boldsymbol{w}^{a_f^{m,r}})\right\|^2\right] \leq \frac{\mathbb{E}\left[f^{a_f^m}(\boldsymbol{w}^{a_f^{m,r}})\right]}{\eta} - \frac{\mathbb{E}\left[f^{a_f^m}(\boldsymbol{w}^{s_f^{m,r+1}})\right]}{\eta} + 2\eta L^3\left(B^2+1\right)\left(f^{a_f^m}(\boldsymbol{w}^{a_f^m 0})-f^{a_f^m\star}\right) + \left(1-\frac{A^m}{K}\right)\frac{\eta L^2}{A^m}G_2^2 + \sqrt{2G_1^2(G_2^2+2LB^2(f^{a_f^m}(\boldsymbol{w}^{a_f^{m,r}})-f^{s_f^m\star}))c^{a_s^m,r}}.$$

By applying Lemma 1, Lemma 2 from [14], defining $F^{a_f^m} := \max_r\{f^{a_f^m}\left(\boldsymbol{w}^{a_f^{m,r}}\right)-f^{a_f^m\star}\}$, and averaging the summation over the third term, we obtain:

$$\mathbb{E}\left[\left\|\nabla f^{a_f^m}(\boldsymbol{w}^{a_f^m,r})\right\|^2\right] = \mathcal{O}\left(\frac{C_1}{R}+\frac{H_2\sqrt{F^{a_f^m 0}}}{\sqrt{RA^{m,r}}}+\frac{F^{a_f^m 0}}{\eta R}\right),$$

where $H_2^2 := L^3\left(B^2+1\right)F^{a_f^m 0} + \left(1-\frac{A^m}{K}\right)L^2G_2^2$, $F^{a_f^m} := f^{a_f^m}\left(\boldsymbol{w}^{a_f^m 0}\right) - f^{a_f^m\star}$, and $C_1 = G_1\sqrt{G_2^2+2LB^2F^{a_f^m 0}\sum_r c^{a_s^m,r}}$.

The proof for the non-convex case follows a similar approach by disregarding Assumption 2. Using telescoping sum, the proof for non-convex functions yields the following rate.

$$\mathbb{E}\left[\left\|\nabla f^{a_f^m}(\boldsymbol{w}^{a_f^m,r})\right\|^2\right] = \mathcal{O}\left(\frac{C_2}{R}+\frac{H_2\sqrt{F^{a_f^m 0}}}{\sqrt{RA^{m,r}}}+\frac{F^{a_f^m 0}}{\eta R}\right),$$

where $C_2 = G_1\sqrt{G_2^2+B^2G_1^2\sum_r c^{a_s^m,r}}$. The fast agent-side bound has an extra term due to its dependence on the slow agent-side model convergence, leading to a looser bound.

## REFERENCES


[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[3] 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.

[4] D.-J. Han, H. I. Bhatti, J. Lee, and J. Moon, "Accelerating federated learning with split learning on locally generated losses," in *ICML 2021 Workshop on Federated Learning for User Privacy and Data Confidentiality. ICML Board*, 2021.

[5] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng, "Tifl: A tier-based federated learning system," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 125–136.

[6] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. Association for Computing Machinery, 2021.

[7] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[8] R. Zong, Y. Qin, F. Wu, Z. Tang, and K. Li, "Fedcs: Efficient communication scheduling in decentralized federated learning," *Information Fusion*, vol. 102, p. 102028, 2024.

[9] C. Ma, J. Li, M. Ding, H. H. Yang, F. Shu, T. Q. Quek, and H. V. Poor, "On safeguarding privacy and security in the framework of federated learning," *IEEE network*, vol. 34, no. 4, pp. 242–248, 2020.

[10] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

[11] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip learning as a decentralized alternative to federated learning," in *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019, Proceedings 19*. Springer, 2019, pp. 74–90.

[12] Z. Tang, S. Shi, B. Li, and X. Chu, "Gossipfl: A decentralized federated learning framework with sparsified and adaptive communication," *IEEE Transactions on Parallel & Distributed Systems*, vol. 34, no. 03, pp. 909–922, 2023.

[13] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2019.

[14] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.

[15] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of cnns," in *International Conference on Machine Learning*. PMLR, 2020, pp. 736–745.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[17] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[18] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

[19] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[20] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," in *International Conference on Learning Representations*, 2020.

[21] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nature communications*, vol. 13, no. 1, p. 2032, 2022.

[22] S. M. S. Mohammadabadi, S. Zawad, F. Yan, and L. Yang, "Speed up federated learning in heterogeneous environment: A dynamic tiering approach," *arXiv preprint arXiv:2312.05642*, 2023.

[23] M. Chen, N. Shlezinger, H. V. Poor, Y. C. Eldar, and S. Cui, "Communication-efficient federated learning," *Proceedings of the National Academy of Sciences*, vol. 118, no. 17, p. e2024789118, 2021.

[24] W. Zhang, D. Yang, W. Wu, H. Peng, N. Zhang, H. Zhang, and X. Shen, "Optimizing federated learning in distributed industrial iot: A multi-agent approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3688–3703, 2021.

[25] A. Li, M. Markovic, P. Edwards, and G. Leontidis, "Model pruning enables localized and efficient federated learning for yield forecasting and data sharing," *Expert Systems with Applications*, vol. 242, p. 122847, 2024.

[26] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[27] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[28] T. Sun, D. Li, and B. Wang, "Decentralized federated averaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[29] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 556–571, 2013.

[30] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Network*, vol. 35, no. 1, pp. 234–241, 2020.

[31] P. Ramanan and K. Nakayama, "Baffle: Blockchain based aggregator free federated learning," in *2020 IEEE international conference on blockchain (Blockchain)*. IEEE, 2020, pp. 72–81.

[32] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," *arXiv preprint arXiv:1901.11173*, 2019.

[33] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *arXiv preprint arXiv:2003.02133*, 2020.

[34] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[35] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[36] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[37] J. Shen, N. Cheng, X. Wang, F. Lyu, W. Xu, Z. Liu, K. Aldubaikhy, and X. Shen, "Ringsfl: An adaptive split federated learning towards taming client heterogeneity," *IEEE Transactions on Mobile Computing*, 2023.

[38] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 337–16 346.

[39] 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*, 2016, pp. 308–318.

[40] H. U. Sami and B. Güler, "Secure aggregation for clustered federated learning," in *2023 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2023, pp. 186–191.

[41] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 656–672.

[42] D. Yao, L. Xiang, H. Xu, H. Ye, and Y. Chen, "Privacy-preserving split learning via patch shuffling over transformers," in *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2022, pp. 638–647.

[43] P. Vepakomma, A. Singh, O. Gupta, and R. Raskar, "Nopeek: Information leakage reduction to share activations in distributed deep learning," in *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2020, pp. 933–942.

[44] E. Erdogan, A. Küpçü, and A. E. Cicek, "Splitguard: Detecting and mitigating training-hijacking attacks in split learning," in *Proceedings of the 21st Workshop on Privacy in the Electronic Society*, 2022, pp. 125–137.

[45] S. M. S. Mohammadabadi, "Communication-Efficient Training Workload Balancing," https://github.com/mahmoudsajjadi/ComDML, accessed on 1 May 2024.