ADAPTIVE SPLIT LEARNING

Anonymous Authors[1]

ABSTRACT

Federated learning (FL) is a popular distributed deep learning framework which enables personalized experiences across a wide range of web clients & mobile/IoT devices. However, FL-based methods are challenged by the compute resources on client devices given the exploding growth in size of state-of-the-art models (eg. billion parameter models). Split Learning (SL), a recent framework, reduces client compute load by *splitting* model training between client and server. This flexibility is useful for low-compute setups but is achieved at the cost of massive increase in bandwidth consumption. This split also results in sub-optimal performance, especially when data across clients is heterogeneous. The goal of this paper is to make SL a viable alternative to FL. Specifically, we introduce adaptive split learning (*AdaSplit*) which enables efficiently scaling SL to low-resource scenarios by reducing bandwidth consumption and improving performance across heterogenous clients. We validate the effectiveness of *AdaSplit* under limited resources through extensive experimental comparison with strong federated and split learning baselines. Finally, we also present sensitivity analyses of key design choices in *AdaSplit* which highlight the ability of *AdaSplit* to adapt to variable resource budgets. We anonymously release our code here.

1 INTRODUCTION

Distributed machine (deep) learning is characterized by a setting where many clients (web browsers, mobile/IoT devices) collaboratively train a model under the orchestration of a central server (eg. service provider), while keeping the training data decentralized. As strict regulations emerge for data capture and storage, such as GDPR (Goddard, 2017) and CCPA (Stallings, 2020), distributed deep learning is being used to enable privacy-aware personalization across a wide range of web clients and smart edge devices with varying resource constraints. For instance, distributed deep learning is replacing third-party cookies in the chrome *browser* for ad-personalization (Epasto et al., 2021), enabling next-word prediction on *mobile devices* (Hard et al., 2018), speaker verification on *smart home assistants* (Guliani et al., 2021), HIPPA-compliant diagnosis on *clinical devices* (Rieke et al., 2020) and real-time navigation in *vehicles* (Elbir et al., 2020).

A general distributed deep learning pipeline involves multiple rounds of *training* and *synchronization* steps where a model is *trained* with local client data in each round and updates made by multiple clients are *synchronized* by the server into a global model. Techniques have been proposed with the goal to maximize accuracy under constraints on resource (bandwidth, compute) consumption. Figure 1 compares our proposed *AdaSplit* (in yellow) with strong baselines (McMahan et al., 2016; Li et al., 2020; Karimireddy et al., 2021; Wang et al., 2020; Gupta & Raskar, 2018; Thapa et al., 2022) along these dimensions.

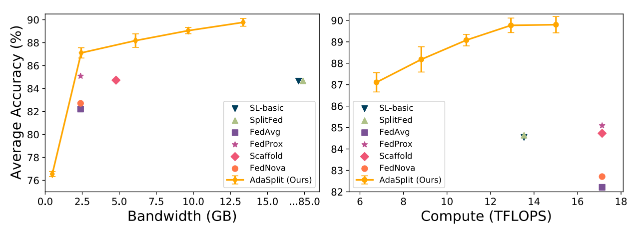

Figure 1. AdaSplit achieves improved accuracy under limited resources (bandwidth & compute) and can also adapt to variable resource budgets. Results on *Mixed-NonIID* dataset.

Federated Learning (FL) (McMahan et al., 2016) is one widely studied framework (McMahan et al., 2016; Li et al., 2020; Wang et al., 2020; He et al., 2020; Yu et al., 2021; Yang et al., 2021; Li & Zhan, 2021; Cheng et al., 2017). In each round of FL, *first*, all clients train a copy of the model locally on their device for several iterations and communicate the final model parameters with the server. The server then *synchronizes* updates across clients by *averaging* all clients model parameters and shares back the unified global model for next training round (figure 2). With entire model training done on each client, FL is **challenged** by the *compute budgets* of client devices. Specifically, ***i.)*** on-device model training needs resource-intensive clients (with high-performance GPUs to avoid stragglers) and is increasingly becoming impractical due to exploding growth in model sizes (eg. billion parameter models for language and image modeling (Radford et al., 2019; Devlin et al., 2018; Zhai et al., 2021)). ***ii.)*** as number of clients (and/or model sizes) scales, bandwidth requirements for the system may worsen as entire models need to be communicated between client and server. ***iii.)*** storing the entire trained model on-client can often have intellectual property implications that limit real-world usability.

Split Learning (SL) (Gupta & Raskar, 2018; Thapa et al., 2022; Poirot et al., 2019; Vepakomma et al., 2018) has emerged recently as a framework to alleviate some of the key concerns faced by FL. SL reduces client computation load by actively involving the server in the training process. In each round, clients take turns to interact with the server for multiple iterations where they update parameters of a local model on the client and a (shared) global model residing on the server. Specifically, at each iteration, the client model generates input activations that are communicated to the server. On the server, the activations are passed through the server model to make predictions and compute gradients for training both the server model (on server) and client model (by transmitting to the client). This is visualized in figure 2. While client computation is significantly reduced in SL versus FL, this comes at the cost of an increase in client-server communication and often sub-optimal performance. Specifically, ***i)*** *communication budgets* increase as the client interacts with the server in every iteration of a round (vs once-per-round in FL), as it is dependent upon the server to generate gradient updates for training the client model. This also blocks the server to train synchronously with each client. ***ii)*** as clients *sequentially* update shared parameters on the server, convergence may be inefficient or sub-optimal, especially when the data across clients is heterogeneous.

Contributions: The focus of this paper is to alleviate the above concerns and make SL a viable alternative to FL. We introduce *AdaSplit*, which enables SL to scale to low-resource scenarios. *First,* a key insight in AdaSplit is to eliminate client dependence on server gradients, which reduces communication cost and enables asynchronous (client-server) training. *Next*, motivated by the fact that neural networks are vastly overparameterized, *AdaSplit* is able to improve performance by constraining the heterogeneous clients to *only* update sparse partitions of the server model. As shown in figure 1, this enables AdaSplit to not only achieve improved performance under fixed resources (higher accuracy when similar bandwidth and compute), but also adapt to variable resource budgets (the trade-off curve). *Additionally*, to unify evaluation along these multiple metrics for distributed deep learning (DDL), we propose *C3-Score* to jointly benchmark performance under resource budgets. We validate the effectiveness of *AdaSplit* through extensive comparisons with state-of-the-art baselines (Table 1, 2) and sensitivity analyses of key design choices (Tables 3, 4, 5, 6).

2 Preliminaries

Here, we formalise the protocol and notation for the Split Learning (SL) framework. This is also visualized in figure 2 (bottom-left). For completeness, we also summarize the FL protocol in figure 2 (top-left). Due to limited space here, we refer the reader to (Kairouz et al., 2019) for a review of the FL protocol and to (Gupta & Raskar, 2018) for more background on SL.

SL - Protocol and Notations: Consider a distributed learning setup with N participating clients and one coordinating server. The key idea of split learning (SL) is to distribute (or *split*) the parameters of the training model across client and server. Each client i, for $i \in [1, 2, ..., N]$ is characterized by a local client dataset D_i, local client model M_i^c and a single server model M^s which is updated by all the clients. The training protocol is executed over R rounds of T iterations each. In each round, the N clients sequentially obtain access to interact with the server for model training over T iterations. In each iteration j (for $j \in [1, 2, , .., T]$), client i updates the parameters of both M^s and M_i^c. *First*, a mini-batch (x_i, y_i) is sampled from D_i and passed through layers of client model M_i^c to generate activations a_i $(= M_i^c(x_i))$. We may refer to a_i as *split activations*. *Second*, the pair of (a_i, y_i) is transmitted to the server. *Third*, at the server, a_i is passed through layers of server model M^s to generate predictions $\hat{y}_i$ $(= M^s(a_i))$. The loss function $L(y_i, \hat{y_i})$ is computed to generate gradients which are used to locally update parameters of M^s and then transmitted to the client to update parameters of M_i^c. In the classical setup, clients follow a round-robin mechanism where client $i + 1$ can start interacting with the server only after client i has completed its T iterations for the round. The global model is synchronized implicitly across clients by updating weights of the shared server model M^s. Furthermore, in some variants of SL, clients' models are transmitted between pairs of clients during a round (Gupta & Raskar, 2018) or averaged over all clients after the round ends (Thapa et al., 2022). Extensive research is focused on privacy in SL and, while beyond scope of this paper, we briefly discuss that in Section 8.

3 Setup and Motivation - 3C

While the FL and SL protocols may appear different, we posit that they are motivated by the same goal - to maximize performance (accuracy) of the global model, under constraints on resource consumption. Here, we make a step towards unifying their design choices along *three key design dimensions* which focus on how i) models are trained on local client data (*Computation*) and, ii) updates across the clients are synchronized, via the server, into a global model (*Communication* and *Collaboration*). This helps motivate our proposed *AdaSplit* for improving SL.

1. Computation: This governs how the model training using data at each client is executed. The computation cost can be defined as the total floating-point operations (FLOPs) executed across the client and server. **FL and SL differ in where the computation happens.** For N clients, this cost

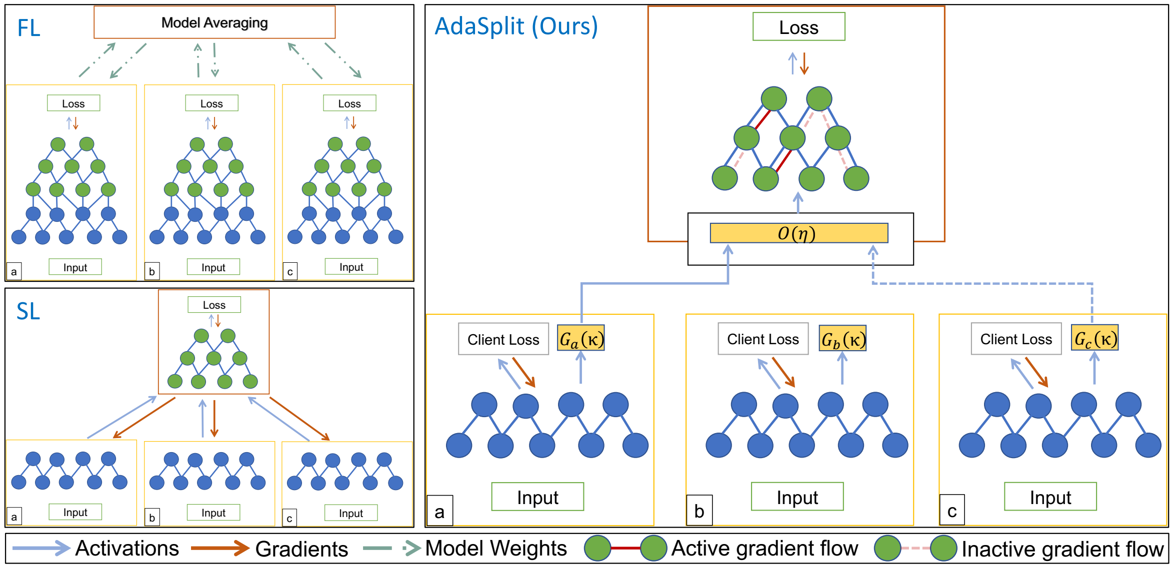

Figure 2. Training protocols with N=3 clients for federated learning (FL), split learning (SL) and our proposed AdaSplit which builds upon split learning framework. *AdaSplit* improves i) *Computation* using the local client gradient (with L_{client}) and training the server intermittently (using gate $G(.)$ parameterized by κ), ii) *Communication* by reducing payload size (no gradient flow from server-client) and interaction frequency (using $O(.)$ parameterized by η) and iii) *Collaboration* by allowing each client to update sparse partition of server parameters (on edges with active gradient flow). Specifically, in this figure, client (b) is in local phase and client (a,c) are in global phase. Client (a) is selected to train and it only updates a sparse partition of server model parameters corresponding to edges with active gradients on the server. The protocol is detailed in Section 4.

$(C1)$ can be represented as:

$$C1 = \sum_{i=1}^{N} R * (F_i^c * T_i^c + F_i^s * T_i^s) \quad (1)$$

where, F_i^c are the FLOPs executed on client for T_i^c iterations, F_i^s are FLOPs executed on server for T_i^s iterations when training with data for client i and R is number of rounds. F_i^c and F_i^s increase (or decrease) monotonically with increase (or decrease) in size of client model M_i^c and server model M^s respectively. **i) In FL**, $F_i^s = 0$ and $T_i^s = 0$ since the entire model is executed on client device ($M^s = 0$). In constrast, **ii) SL** allows to split the model and distribute F_i^c and F_i^s between client and server, based on resource availability. This flexibility of SL is key for scaling to low-resource setups where clients are compute constrained (but servers may scale horizontally). **To motivate AdaSplit**, we note that, in classical SL, this may increase computation load on the server and also block the server to train synchronously with each client.

2. Communication: This governs how client-and-server interact with each other. The communication cost can be defined as the total payload that is transmitted between each of the N client-server pairs over multiple rounds of training. **FL and SL differ in the type of payload and frequency of communication.** Without loss of generality, this cost $(C2)$ can be represented as:

$$C2 = \sum_{i=1}^{N} \sum_{j=1}^{R} \sum_{k=1}^{T} (P_{is} + P_{si}) * \sigma(i, j, k) \quad (2)$$

where N is number of clients, R is training rounds and T is iterations per round. P_{is} is the payload transmitted from client i to server s and P_{si} is the payload transmitted from server s to client i. $\sigma(i, j, k)$ denotes if client i interacts with server during iteration k of round j. **i) In FL**, client-server interact using model weights once-per-round. Hence, size of each P_{is}, P_{si} is size of the total model and $\sigma(i, j, k) = 1$ *only* for $k = T$ (last iteration of every round). **ii) In SL**, P_{is}, P_{si} is size of a batch of activations and gradients respectively and $\sigma(i, j, k) = 1 \; \forall i, j, k$ since client depends upon server for gradient. **To motivate AdaSplit**, we note that, even with smaller payload for SL (one activation batch vs full model), the high frequency of communication results in more bandwidth consumption than FL.

3. Collaboration: This governs how learning (or updates) from local data across the clients is synchronized in the global model. Unlike communication and computation, the cost is non-trivial to define but the impact is measured from the converged accuracy. If the client datasets D_i for $i \in [1, 2, .., N]$ could be centralized, the unified dataset D $(= D_1 \cup D_2 ... \cup D_N)$ can be used to train a performant model with gradient descent by sampling iid batches $b \sim D$. FL and SL require mechanisms to achieve convergence when this data is decentralized. **FL and SL differ in the input and protocol used by the server to aggregate updates across clients**. Abstractly, **i) FL** executes this by averaging client model parameters (or gradients) on the server after each round, and **ii) SL** executes this by requiring all clients to (sequentially) update shared parameters of the server during the round.

In federated training, the global model in a round r and consequently updated client models (M_i^c) are obtained as:

$$M^g = \sum_{i=1}^{N}(M_i^c * p_i^r); \quad M_i^c = M^g, \forall i \in [1, 2, ..., N] \tag{3}$$

where p_i^r is a weight assigned to client i in round r.

In split training during each round r, the server model (M^s) is updated sequentially by all client i for $\forall i \in [1, 2, ..., N]$ as:

$$M^s = M^s - \alpha * \nabla \hat{L}(M^s(a_i), y_i) \tag{4}$$

In some variants of SL such as (Thapa et al., 2022), local client models are also synchronized, at end of each round, as in FL using equation 3. Then, the global model is obtained by stacking the server and (averaged) client models. **To motivate AdaSplit**, we note that when data across clients is non-iid (common in real-world setup), inefficient or sub-optimal converged accuracy is observed. We posit that this happens since (gradient of) non-iid client activations *sequentially update the same parameters in M^s*, which is inconsistent with ERM (Vapnik, 1992).

4 ADASPLIT

Here, we delineate the design choices of *AdaSplit* along each of the three dimensions. The architecture is visualized in Fig. 2 (right). We also discuss corresponding trade-offs that enable AdaSplit to adapt to variable resource (communication, computation) budgets. The text follows the same notation as defined in Sec. 2.

4.1 Computation

Recall from Sec. 3 that in classical SL, splitting model between client and server decreases client computation load (vs FL) but increases computation load on the server and also blocks the server to train synchronously with each client as they depend on the server for gradient. AdaSplit alleviates this by: i) eliminating the dependence of the client model on server for gradient and ii) *only* training the server intermittently. *AdaSplit has the same on-client computation as SL but lower server computation by decreasing T_s (compute iterations on the server) – reducing total computation.*

Local Client Gradient: *First*, AdaSplit generates the gradient for training client model on-client itself using a local objective function L_{client} which is a supervised version of NT-Xent Loss (Sohn, 2016). Given an input batch, $b \sim D_i$, then for each input $(x_i, y_i) \sim b$, L_{client} is applied on a projection ($H(.)$) of the activations a_i generated by the client model ($= M_i^c(x_i)$). Let $q_i = H(a_i)$ be the corresponding embedding of an input x_i, and Q_+^i be the set of embeddings of other inputs with the same class as x_i in the batch b, the loss can be represented as below:

$$L_{client} = \sum_{i=0}^{|b|} \sum_{q_+ \in Q_+^i} - \log \frac{exp(q_i \cdot q_+/\tau)}{\sum_{j \neq i}^{|b|} exp(q_i \cdot q_j/\tau)} \tag{5}$$

Here, τ is a hyperparameter, which controls the ”margin” of closeness between embeddings. We set $\tau = 0.07$ in all our experiments. The pairs (anchor q_i, positive inputs q_+) required in L_{client} are sampled using the ground truth labels (y_i) locally on client.

Intermittent Server Training: *Second*, AdaSplit also *splits* the R round training into two phases: A) *Local Phase* B) *Global Phase*. *Local Phase* lasts for the first κ rounds when only the client model is trained, asynchronously and without interacting with the server, using L_{client}. After κ rounds (till end), the *Global Phase* starts where client continues to train locally and also interacts with the server by transmitting activations. The server model *only now* starts being trained using activations received from the clients. The server model M^s is optimized using a server loss function (L_{server}) which is cross-entropy (L_{ce}) for classification tasks. We note that in global phase (when server is training M^s), the client *does not* receive any gradient from the server but still continues to (asynchronously) train its client model M_i^c using only the local client loss L_{client}.

Discussion: AdaSplit can adapt to variable computation budgets by regulating two key hyperparameters: i) size of the client model (μ) (for client compute), ii) duration of local phase (κ) (for server compute). To clarify, μ helps regulate client (and server) computation, and κ regulates server computation but does not affect the client at all. We study the specific impact of these design choices in Sec. 7. In practice, we observe considerable reductions in total computation since κ can take relatively large values (0.8*R), where R is total training rounds, without significant loss of performance. We corroborate this with results in Sec. 6.

4.2 Communication

Recall from Sec. 3 that in classical SL, the high client-server interaction can be prohibitive for communication cost. AdaSplit alleviates this problem by reducing: i) the frequency of communications; and ii) the payload size.

Smaller Payload: *First,* we would like to highlight that eliminating client dependence on server gradient can potentially also reduce communication cost, in addition to the computation overhead. Unlike SL, in AdaSplit the server does not transmit gradients to the client and hence $P_{si} = 0$ (in equation 2 in Sec. 3) throughout training for each client i. Through sensitivity analysis in Sec. 7, we validate that this design choice marginally drops the performance while significantly reducing communication.

Infrequent Communication: *Second*, we note that two-phase training (introduced in Sec. 4.1) is also beneficial for reducing communication. In the *Local Phase*, there is no client-to-server communication and thus the payload $P_{is} = 0$ for all clients i (in equation 2 in Sec. 3). In the *Global Phase*, clients may start transmitting activations to the server. In this phase, only a subset of clients communicates with the server in each round. Specifically, we introduce an *Orchestrator* (*O*) which resides on the server and uses a running statistic of local client losses to select ηN (for some $0 \leq \eta \leq 1$) clients in each iteration, that communicate with the server. In AdaSplit, *O* uses a UCB (Auer, 2003) strategy to prioritize clients who need the server model to improve performance on their data (*exploitation*) while also ensuring that the final global model can generalize well to different client data distributions (*exploration*).

Let S_i^t be a binary flag denoting if client i is selected to transmit activations to the server at iteration t and L_i^t denote the server loss from activations (a_i) for the iteration. At each iteration t, selected clients (i.e. $S_i^t = 1$) transmit input activations to update server model and the loss L_i^t is stored. For unselected clients (i.e. $S_i^t = 0$), L_i^t is defined the average of their loss value in previous iterations ($L_i^t = \frac{L_i^{t-1} + L_i^{t-2}}{2}$), as in (Auer, 2003). Here, we note that L_i^t is only used for selection and the client model continues to train locally with L_{client}. *Finally, O* assigns a new score to each client using the advantage function described in the following and clients with the top-η scores are selected for the next iteration. The advantage function (A_i) for (Auer, 2003) is defined as $A_i = \frac{l_i}{s_i} + \sqrt{\frac{2 \log T}{s_i}}$; where, $l_i = \sum_{t=0}^{T} \gamma^{T-1-t} \cdot L_i^t$, $s_i = \sum_{t=0}^{T} \gamma^{T-1-t} \cdot S_i^t$ and T is total iterations in the round. $\gamma \in [0, 1]$ is a hyperparameter that determines the importance of historical losses. We initialize $L_i^t = 100$ for all clients for $t = 0$ and $t = 1$.

We make a few statements here. *First,* note that subset selection has previously been used in FL to regulate communication cost (McMahan et al., 2016; Li et al., 2020; Cho et al., 2020b) where the global model after a round may be obtained from few clients only (p_i^r in equation 3). *However*, classical SL does not have a similar infrastructure since each client is entirely dependent on the server for gradient during each training iteration (of every round). Eliminating client dependence on the server gradient in AdaSplit helps unlock this benefit. *Finally*, we mention that this orchestrator is specialized for AdaSplit where it needs to be invoked in each iteration (vs rounds in FL) and selects client to transmit activations for training (vs model averaging in FL).

Discussion: AdaSplit can adapt to variable communication budgets by regulating two key hyperparameters: i) the fraction of selected clients (η), ii) the duration of the local phase (κ). We study the specific impact of these design choices in Sec. 7. In practice, we observe considerable reductions in communication cost since κ, η can assume large values ($\kappa = 0.8 * R, \eta = 0.6$) without significant loss of performance. We corroborate this with results in Sec. 6.

4.3 Collaboration

AdaSplit, like SL, synchronizes updates in the global model by requiring clients to sequentially update shared server model parameters. Recall from Sec. 3 that when inter-client data is heterogeneous, this often results in the global model converging to sub-optimal accuracy. To alleviate this, the intuitive goal is to prevent clients with different data distributions from "interfering" with each other during training of the server model. To achieve this, the key idea of AdaSplit is to have each client update *only* a partition of the server model (M_s) parameters. The motivating insight is that neural network models are vastly over-parameterized (Neyshabur et al., 2018) and only a small proportion of the parameters can learn each (client's) task with little loss in performance (Golkar et al., 2019; LeCun et al., 1990). Conventionally, this is used for model compression; in contrast, we leverage it to reduce interference in distributed DL with heterogeneous (non-iid) data across clients.

Update Sparse Partitions of Server Model: During the *global phase*, we add an L^1 weight regulator to promote sparsity in the server model M^s. Specifically, instead of pruning the network, we learn a client (i) specific multiplicative mask m_i which constrains the subset of M^s parameters client i can update. Given batch of activations a_i from client i, server model M^s is updated as:

$$M^s = M^s - \alpha * m_i * \nabla \hat{L}(M^s(a_i), y_i) \tag{6}$$

This simulates relative sparsity (for each client) in M^s without pruning any parameters since the goal is to increase server model capacity (to accommodate many diverse clients) rather than achieving compression. Here, m_i evolves during training and is forced to be extremely sparse using the below loss function on the server:

$$L_{server} = L_{ce}(\hat{y_i}, y_i) + \lambda * \omega(m_i) \tag{7}$$

where, $\omega(.)$ is an L^1 regularizer, $\hat{y_i} = M^s(M_i^c(x_i))$ and $L_{ce}(.;.)$ is the cross entropy loss. The λ hyperparameter regulates sparsity of the masks and can be intuitively visualized as controlling the extent of *collaboration between clients, via the server*. At inference, the effective server model for client i is $M^s * m_i$ where m_i is a highly sparse binary mask and can potentially be stored on client device. Results in Sec. 6 show that this strategy of regulating collaboration significantly improves performance. Finally, we note similarities between *each round* of collaboration in SL (and AdaSplit) and continual learning (Golkar et al., 2019), albeit AdaSplit works in activation space and is iterative. However, we anticipate exploring this connection may present interesting directions of future work.

4.4 Summary of Claims

Here, we briefly summarize key takeaways from this Section. To reiterate, the goal of AdaSplit is to improve SL, so that it can become a competitive alternative to FL. Conventional SL methods reduce on-client computation, which is a key bottleneck to FL, but increase server-computation, communication overhead and often achieve lower accuracy when compared to FL methods. AdaSplit is designed to help alleviate these concerns.
AdaSplit introduces the following ideas to SL: local client gradients (sec 4.1), intermittent server training (sec 4.1), infrequent communication with smaller payloads (sec 4.2) and sparse updates of the server model (sec 4.3). *Next*, sec 6 & 7 present results to show that these ideas enable AdaSplit to i) preserve the low on-client computation as other SL methods (the only shared aspect), ii) reduce server (and hence, total) computation cost (sec 4.1), iii) reduce communication cost (sec 4.2) and iv) improve collaboration between clients, evident via better accuracy (sec 4.3).

5 Experimental Setup

Here, we specify the datasets and baselines used, the evaluation protocols and implementation details for the results presented in this work. All our code is released here.

5.1 Datasets

To validate the efficacy of AdaSplit, we conduct extensive experiments on benchmark datasets and simulate varying levels of inter-client heterogeneity. Specifically, we use two experimental protocols, as described next: **a) Mixed-CIFAR:** We divide the 10 classes of CIFAR-10 into 5 subsets of 2 distinct classes each. Every client is assigned data from one of the 5 subsets. In this protocol, there is low and consistent heterogeneity between data across all pairs of clients. **b) Mixed-NonIID:** We use 5 benchmark datasets: i) MNIST ii) CIFAR-10 iii) FMNIST iv) CIFAR-100 v) Not-MNIST and each client receives samples from exactly one dataset. In this protocol, there is high and variable inter-client heterogeneity between client pairs. For instance, clients with FMNIST and MNIST have lower pairwise-heterogeneity between each other and high pairwise heterogeneity with clients containing CIFAR-100. For all experiments, the RGB images are scaled to 32x32 and grayscale images (in MNIST) stacked along channels.

5.2 Baselines

The key motivation behind AdaSplit is to make SL a viable alternative to FL. We compare with state-of-the-art SL and FL techniques. Specifically, for SL, we compare with SL-basic (Gupta & Raskar, 2018) and SplitFed (Thapa et al., 2022). To ensure validity of analysis and highlight efficacy of results, we also compare with popular FL techniques: FedAvg (McMahan et al., 2016), FedNova (Wang et al., 2020), Scaffold (Karimireddy et al., 2021) and FedProx (Li et al., 2020). These techniques are specially designed for heterogenous (non-iid) setups and provide strong benchmarking for the efficacy of AdaSplit.

5.3 Evaluation Metrics

We evaluate performance, both independently along multiple standard metrics as well as jointly using a unified metric.

i) Independent Evaluation: To evaluate along the design dimensions, we report the results using three metrics, *Accuracy*, *Bandwidth* and *Compute*. *Accuracy* is reported as mean and standard deviation over multiple independent runs with different seeds. *Bandwidth* is reported in GB and *Compute* in TFLOPS. We note that in many real-world cases, servers may scale horizontally and the bottleneck is often at the client side. For completeness, we seperately report both client compute and total (clients+server) compute. We highlight here that, to ensure fair comparison, we ensure results reported in Sec. 6 (Tables 1, 2) and Sec. 7 (Tables 3to 6) allow for independent comparison along each of these metrics.

ii) Joint Evaluation: For an effective distributed deep learning method, the goal is to maximize performance throughput, e.g., accuracy, while minimizing resource (bandwidth, compute) consumption. For practical use, however, we often need to jointly adhere to constraints on resource (bandwidth, compute) consumption and the achieved performance (accuracy). For instance, a 50% decrease in bandwidth use could be more important than a 5% increase in accuracy. Hence, it would help to use a unified metric that can encapsulate these three different metrics. We make a step towards introducing one such metric for distributed DL.

Properties: While not exhaustive, some desirable properties for such a metric are: **i) Flexible**: explicitly incorporating resource budgets is important for practical use as it helps identifying the *best technique for a given resource budget*. For instance, research in differential privacy uses privacy budgets (defined via $\epsilon - \delta$ parameters) to contextualize comparison between different privacy mechanisms. **ii) Normalized**: the output score for every method should be bounded, for ease of comparison. **iii) Extensible**: it should be easy to extend to other resource dimensions. For instance, while we consider two resource budgets (bandwidth, compute) here, including privacy budget is an interesting future direction with techniques such as DP-SGD (Abadi et al., 2016) becoming relevant for FL and SL.

Realization: ***C3-Score*** is **one** such simple metric, that we

Table 1. Results on **Mixed-NonIID** dataset. AdaSplit achieves improved performance while reducing resource (bandwidth, compute) consumption. This is corroborated by the *C3-Score* (higher is better). Compute is reported as client (client + server).

Method	Accuracy	Bandwidth (GB)	Compute (TFLOPS)	**C3-Score**
FedAvg (McMahan et al., 2016)	82.21 ± 0.19	2.39	17.13 (17.13)	0.72
FedProx (Li et al., 2020)	$\mathbf{85.09 \pm 0.29}$	2.39	17.13 (17.13)	0.75
Scaffold (Karimireddy et al., 2021)	84.73 ± 0.17	4.78	17.13 (17.13)	0.74
FedNova (Wang et al., 2020)	82.71 ± 0.27	2.39	17.13 (17.13)	0.73
SL-basic (Gupta & Raskar, 2018)	84.65 ± 0.32	84.54	3.76 (15.14)	0.72
SplitFed (Thapa et al., 2022)	84.67 ± 0.24	84.64	3.76 (15.14)	0.73
AdaSplit (κ=0.6, η=0.6)	$\mathbf{88.88 \pm 0.27}$	9.71	5.38 (8.82)	**0.85**
AdaSplit(κ=0.75, η=0.6)	$\mathbf{87.11 \pm 0.59}$	2.43	5.38 (10.88)	**0.83**

Table 2. Results on **Mixed-CIFAR** dataset. AdaSplit achieves improved performance while reducing resource (bandwidth, compute) consumption. This is corroborated by the *C3-Score* (higher is better). Compute is reported as client (client + server).

Method	Accuracy	Bandwidth (GB)	Compute (TFLOPS)	**C3-Score**
FedAvg (McMahan et al., 2016)	91.31 ± 0.49	2.39	11.77 (11.77)	0.79
FedProx (Li et al., 2020)	$\mathbf{92.54 \pm 0.48}$	2.39	11.77 (11.77)	0.81
Scaffold (Karimireddy et al., 2021)	87.30 ± 1.36	4.79	11.77 (11.77)	0.76
FedNova (Wang et al., 2020)	88.94 ± 0.32	2.39	11.77 (11.77)	0.77
SL-basic (Gupta & Raskar, 2018)	67.90 ± 3.52	34.88	1.66 (13.76)	0.59
SplitFed (Thapa et al., 2022)	71.46 ± 2.13	35.94	1.66 (13.76)	0.62
AdaSplit (κ=0.6, η=0.6)	$\mathbf{91.92 \pm 1.88}$	2.85	2.38 (4.81)	**0.89**
AdaSplit (κ=0.3, η=0.6)	$\mathbf{92.91 \pm 0.91}$	6.57	2.38 (6.63)	**0.88**

propose here. Let B_{max}, C_{max} be the maximum resource budgets for bandwidth and client compute as defined by the evaluator. Then, for a method m with accuracy A_m, bandwidth consumption B_m and client compute consumption C_m, the *C3-Score* is defined as below:

$$C3-Score(A_m, B_m, C_m) = (A_m) * e^{-(\hat{B}_m + \hat{C}_m)/T}, \quad (8)$$

where $\hat{B}_m = B_m/B_{max}$, $\hat{C}_m = C_m/C_{max}$ and T is the temperature (= 10 for all methods and experiments). With this definition, the *C3-Score* metric is bounded between 0 and 1 and monotonic where a higher score represents a better (more efficient) method. We would like to note that the above *C3-Score* metric exponents the resource (bandwidth, compute) dimensions to: i) allow some separation between controllable (resources) and uncontrollable (performance) dimensions and ii) avoid collapse (if $\hat{C}_m$ or $\hat{B}_m \rightarrow 0$), while ensuring a multiplicative form of the metric for easy extensibility. However, we would like to highlight that this is not a unique metric, but just one simple form that captures the desired properties. Thus, to ensure validity and integrity of our study, we only use this *C3-Score* as an *additional* point of comparison in Table 1 and 2.

5.4 Implementation Details

All methods are trained for (R=20) rounds with 1 epoch per round using the same convolutional (LeNet) backbone. Results are reported for 5 (=N) clients, and over 5 runs. For the FL baselines, we use open-source implementations provided in (Li et al.). For robust comparison, we also tuned parameters for these baselines and note some performance gain was observed (over default values) which is then used for comparison. For all SL methods (including AdaSplit), we set the default client model size to 20% ($\mu = 0.2$) and use Adam optimizer with a learning rate of 1e-3, for both client and server. For AdaSplit, the default parameters are: a) κ = 0.6, η = 0.6, γ = 0.87, λ = 1e-5 (for Mixed-CIFAR) and 1e-3 (for Mixed-NonIID). For our study, we set C_{max}, B_{max} to be the respective costs for the worst-performing baselines on the corresponding datasets.

6 Results

We report performance on **Mixed-NonIID** in Table 1 and **Mixed-CIFAR** in Table 2. For purpose of our study here, we set the bandwidth and compute budgets for *C3-Score* to be $B_{max} = 35.94$ GB and $C_{max} = 11.77$ TLFOPS on *Mixed-CIFAR* and $B_{max} = 84.64$ GB and $C_{max} = 17.13$ TFLOPS on *Mixed-NonIID*. These values correspond to the max bandwidth and compute of all the methods on the specific datasets. The results on both datasets consistently support the following key observations:

❶ **AdaSplit** *outperforms other split learning techniques* and achieves significantly better accuracy while also reducing bandwidth consumption. For instance, on *Mixed-CIFAR* (Table 2), in comparison to SL-basic, AdaSplit **improves performance by 24%** and consumes **89% lower** bandwidth. Also, total compute decreases significantly in AdaSplit to 4.81 TFLOPS (versus 13.76), the marginal increase in client compute (2.38 vs 1.66) can be attributed to L_{client}. This is corroborated by an increase in C3-Score from 0.59 for SL-basic (Gupta & Raskar, 2018) to 0.89 for AdaSplit. Furthermore, similar trend is observed on *Mixed-NonIID* (Table 1). Specifically, AdaSplit achieves accuracy of 88.88 against 84.67 for SplitFed while consuming *75 GB less bandwidth*. The corresponding trend is also captured by the *C3-Score* which is 0.85 for AdaSplit as against 0.73 for SplitFed (Thapa et al., 2022).

❷ **AdaSplit** *makes split learning a competitive alternative to federated learning*. On both datasets, we observe that AdaSplit consistently achieves higher (or similar) accuracy with significantly lower client compute and similar bandwidth. For instance, on *Mixed-NonIID*, AdaSplit achieves 87.11% accuracy with 2.43 GB bandwidth and 5.38 TFLOPS compute. In comparison, the closest FL baseline, FedProx, achieves 85% accuracy but consumes 17.13 TFLOPS (3x of AdaSplit) and similar bandwidth (2.39 GB). This is corroborated with a better C3-Score of 0.85 AdaSplit against 0.75 for FedProx.

❸ **AdaSplit** *consistently provides the* ***best trade-off among all*** *of federated and split learning baselines*. For instance, on *Mixed-CIFAR*, AdaSplit achieves a *C3-Score* of 0.89 with the closest FL baseline (FedProx) (Li et al., 2020) is at 0.81, FedAvg (McMahan et al., 2016) at 0.79 and SplitFed at 0.62. Furthermore, similar trend is observed on *Mixed-NonIID* where AdaSplit achieves a *C3-Score* of 0.85 with the closest baseline FedProx at 0.75, Scaffold (Karimireddy et al., 2021) at 0.74 and SL-basic (Gupta & Raskar, 2018) at 0.72.

❹ **AdaSplit** *can adapt to variable resource budgets*. From results on **Mixed-NonIID** (Table 5), we can see that given a higher communication budget (13.36 GB), AdaSplit can further improve accuracy to 89.77% which corresponds to a 5% improvement over FedProx (Li et al., 2020). Figure 1 visualizes how AdaSplit allows to trade-off accuracy by (seperately) varying bandwidth and compute budgets.

Note on Figure 1: *First,* please note that these trade-off curves over bandwidth and compute are obtained while respectively keeping compute and bandwidth budgets fixed. *Second,* we only vary design parameters that are unique to AdaSplit and hence, the same curves cannot be realised for FL or SL baselines. Specifically, we vary duration of local phase (κ), presence of client gradient, and activation sparsity which we discuss in more detail in the next section. *For instance,* client model size (μ) and number of clients (η) are design parameter shared between AdaSplit and other SL methods, and are hence fixed ($\eta = 0.6$, $\mu = 0.2$) for figure 1.

Table 3. Results on *Mixed-CIFAR10*. Varying number of client layers (μ) enables AdaSplit to adapt to variable client computation budgets. Compute is reported as client (client + server).

μ	Accuracy	Bandwidth (GB)	Compute (TFLOPS)
0.2	91.92 ± 1.88	2.85	2.38 (4.81)
0.4	92.12 ± 1.61	1.18	9.04 (9.85)
0.6	86.37 ± 6.74	1.08	11.58 (11.68)
0.8	90.14 ± 2.80	1.05	11.95 (11.97)

7 Discussion

In this section, we conduct sensitivity analyses of key design choices in AdaSplit and analyze the consequent impact on accuracy and resource consumption. Results validate the ability of AdaSplit to efficiently adapt to variable resource budgets. Unless specified otherwise, the hyperparameters used are $\kappa = 0.6$, $\eta = 0.6$, $\mu = 0.2$.

❶ **Varying Size of Client Model:** Table 3 presents results from varying number of layers on client for *Mixed-CIFAR10* dataset. We observe that *Computation* on client increases monotonically with the number of client layers. We also observe a decrease in *Communication* cost as evident from lower bandwidth. This can be attributed to the convolution design of the model where *split activations* becomes smaller with depth (reducing payload P_{is}). Also, we note marginal gain in performance for larger server model since it provides more parameters for *Collaboration*. We observe similar trends on *Mixed-NonIID* and include results in the appendix. Hence, *AdaSplit adapts to variable client computation budgets*.

❷ **Varying Duration of Local Phase:** Table 4 presents results from varying κ on *Mixed-CIFAR10* dataset. We observe that *Communication* cost decreases as k increases. This is because $P_{is} = 0$ for all rounds $r < \kappa$ on given client i. *Computation* cost of the server also decreases on increasing κ though client compute is unchanged. Note that marginal decrease in accuracy is due to the fact that larger κ allows for fewer training iterations of the server model. Specifically, increasing κ from 0.3 to 0.9 decrease accuracy from 89.80% to 87.11%, while bandwidth falls drastically from 17.22 GB to 2.43 GB. This trend is also corroborated on *Mixed-NonIID* dataset, as shown in Table 5. Hence, *AdaSplit adapts to variable communication and server computation budgets*.

❸ Eliminating Gradient Dependence: Table 5 studies the impact of training client model without gradient from server on *Mixed-CIFAR10* dataset. We observe *Communication* cost decreases significantly with bandwidth reduced by one-half. We observe accuracy is generally insensitive though there is slight increase in standard deviation. Hence, *AdaSplit adapts to variable communication budget* and provides consistent performance.

❹ Further Reducing Payload Size: While we sparsify server model parameters to improve collaboration in *AdaSplit*, here we consider sparsification of split activations to reduce communication. Specifically, we train the client model with an additional L^1 regularizer that regulates magnitude of split activations. Results are presented on *Mixed-NonIID* in Table 6. *Computation* remains unchanged. *Communication* decreases as payload (P_{ij}) becomes sparse. For instance, AdaSplit can train with only 0.76 GB of bandwidth and achieve 85.79% accuracy. From Table 1, (Li et al., 2020) achieves 85.09% and consumes 2.39 GB budget. Hence, *AdaSplit adapts to extremely low communication budgets*.

8 Related Work

Here, we review the general landscape of literature in distributed deep learning, along the three dimensions from section 3, as well as delineate specific research and applications in split learning.

Distributed Deep Learning Federated Learning (FL) (McMahan et al., 2016; Kairouz et al., 2019; Karimireddy et al., 2021; Li et al., 2020) and Split Learning (SL) (Gupta & Raskar, 2018; Poirot et al., 2019; Thapa et al., 2022; Singh et al., 2021) are the two main paradigms. While our research contributions are primarily focused on SL, we include relevant literature in both FL and SL and organize the same in context to the three design choices introduced in Sec 3. We refer the reader to (Kairouz et al., 2019) for an extensive review of recent progress and open problems in FL (including a brief survey of SL) and to (Thapa et al., 2021) for a detailed review of SL along with extensive comparisons to FL.

1. Computation: In conventional FL (McMahan et al., 2016; Li et al., 2020; Wang et al., 2020; Karimireddy et al., 2021), computation at each client involves model training during a round, and computation at the server involves synchronization (averaging) of the multiple clients' models after every round. Hence, FL-based methods are challenged by compute resources on client devices given the exploding growth in the size of state-of-the-art models. Some recent work has sought to reduce total computation by training only a part of the model in every round (Diao et al., 2020), pruning the clients' models (Li et al., 2022; Zhou et al., 2021; Jiang et al., 2020) and training the model intermit-

Table 5. Results on *Mixed-NonIID*. In each Accuracy cell, Row-1 trains client with L_{client} and Row-2 trains client with $L_{client} + L_{server}$. Accuracy is largely insensitive to server gradient across various κ

κ	Accuracy	Bandwidth (GB)
0.3	89.80 ± 0.38 89.96 ± 0.23	17.22 34.84
0.45	89.77 ± 0.34 89.47 ± 0.21	13.36 27.18
0.60	89.08 ± 0.38 89.03 ± 0.28	9.65 19.79
0.75	88.17 ± 0.59 88.31 ± 0.40	6.10 12.06
0.90	87.11 ± 0.45 87.05 ± 0.39	2.43 4.89

Table 4. Results on *Mixed-CIFAR10*. Varying duration of local phase (κ) enables AdaSplit to adapt to variable communication and server computation budget. Compute is reported as client (client + server).

κ	Accuracy	Bandwidth (GB)	Compute (TFLOPS)
0.3	92.91 ± 0.91	6.57	2.38 (6.63)
0.45	90.97 ± 1.02	4.72	2.38 (5.72)
0.6	89.77 ± 1.62	3.56	2.38 (4.81)
0.75	88.62 ± 3.68	2.15	2.38 (3.90)
0.90	88.02 ± 0.91	0.89	2.38 (2.98)

Table 6. Results on *Mixed-CIFAR10* dataset. Sparsification of split activations enables AdaSplit to adapt to extremely low communication budgets.

β	Accuracy	Bandwidth (GB)
0	91.09 ± 1.48	3.45
1e-7	90.52 ± 2.16	3.25
1e-6	91.92 ± 1.89	2.85
5e-6	87.6 ± 4.82	1.19
1e-5	85.79 ± 4.10	0.76
0.0001	79.18 ± 4.81	0.08
0.1	51.00 ± 0.42	0.0044

tently (McMahan et al., 2016). These methods improve the overall efficiency of training (or inference) but still need compute-intensive clients to store and train large models - even if intermittently or iteratively execute the add-on compression logic. Recent work is exploring methods to allow heterogeneous models across clients (Li & Wang, 2019), but in the process increases computation load on the server, which now needs to train models (i.e., model distillation) for synchronization. In contrast, SL (Gupta & Raskar, 2018; Thapa et al., 2022) is more flexible and significantly reduces on-client computation by splitting the model between the client and server. In conventional SL (Thapa et al., 2022; Abedi & Khan, 2020; Gupta & Raskar, 2018), however, this benefit is achieved at the cost of an increase in server computation. *AdaSplit* reduces server computation, while preserving the low on-client computation of SL, by introducing local client gradient and training the server intermittently.

2. Communication: In FL, client and server communicate once every training round, and this is executed through weights (or gradients) of the local clients' models. This cost scales with the size of the model and the number of clients in the system, which can become prohibitive. Methods have been proposed to reduce this through compression on client (Konečný et al., 2016; Malekijoo et al., 2021; Hamer et al., 2020), client subset selection (Cho et al., 2020a; Nishio & Yonetani, 2019; Balakrishnan et al., 2020) as well as greedy federated training of client models (Nishio & Yonetani, 2019; Mo et al., 2021). In SL (Gupta & Raskar, 2018; Vepakomma et al., 2018; Poirot et al., 2019), the client and server communicate in each training iteration (of every round) using mini-batch activations and transmit the client models' during (Gupta & Raskar, 2018) or after the round (Thapa et al., 2022; Gawali et al., 2021). *AdaSplit* significantly reduces communication cost in SL by reducing payload size and frequency of client-server interaction.

3. Collaboration: Conventional FL methods (McMahan et al., 2016; Li et al., 2020; Jiang et al., 2020) execute this by averaging models' parameters (or gradients) on the server, after each round. Recent work in heterogenous FL relies on model distillation training on the server (Li & Wang, 2019). The key challenge is with non-iid clients, and this has been extensively investigated in federated learning, where several techniques have proposed (Li & Wang, 2019; Wang et al., 2020; Li et al., 2020; Karimireddy et al., 2021; Zhao et al., 2018). Similar challenges are also observed for conventional SL methods (Gupta & Raskar, 2018; Thapa et al., 2022; 2021) which perform poorly in non-iid setups as evident from sub-optimal or inefficient performance. We posit that this happens since (gradients from) non-iid client activations *sequentially* update shared parameters on the server model. AdaSplit improves performance by constraining clients to only update sparse partitions of the server model.

Split Learning: Research and Applications Split Learning (SL), first introduced in (Gupta & Raskar, 2018; Vepakomma et al., 2018), has become an active direction of research with work across systems (Gupta & Raskar, 2018; Vepakomma et al., 2018; Thapa et al., 2022; Abedi & Khan, 2020), privacy (Pasquini et al., 2020; Singh et al., 2021) and applications (Sharma et al., 2019; Palanisamy et al., 2021; Park et al., 2020; Poirot et al., 2019). In particular, (Vepakomma et al., 2018) summarizes several configurations for model splitting - for executing forward and backward passes, and (Romanini et al., 2021) explores research for (horizontal and vertical) data splitting. Recent works have also integrated federated and split learning architectures (Thapa et al., 2022; Gawali et al., 2021; Abedi & Khan, 2020) to achieve better trade-offs. We refer the reader to (Thapa et al., 2021) for a detailed comparison between the design of FL and SL and (Singh et al., 2019) for a comparison on the communication efficiency of the two protocols. Beyond systems research mentioned here and discussed throughout our paper, split learning also enables distributed/split inference which is not possible with federated learning. Consequently, there is interest in protecting the privacy of both training and testing data with active research in attack (Pasquini et al., 2020; Madaan et al., 2021) and defense (Mireshghallah et al., 2019; Vepakomma et al., 2020; Singh et al., 2021; Samragh et al., 2020) mechanisms. Finally, this has resulted in diverse applications across healthcare (Poirot et al., 2019), model selection (Sharma et al., 2019), IoT (Park et al., 2020) and edge computing (Palanisamy et al., 2021).

9 Conclusion

The goal of this paper is to make split learning (SL) a competitive alternative for federated learning (FL). Conventional SL methods reduce on-client computation, which is a crucial bottleneck to FL, but increase server-computation communication overhead and often achieve lower accuracy when compared to FL methods. Our adaptive split learning (*AdaSplit*) preserves the low on-client computation as other SL methods while i) reducing server computation by eliminating client-dependence on server gradient and training the server intermittently, ii) reducing communication overhead by decreasing payload size and client-server interaction frequency, and iii) improving collaboration by constraining the heterogeneous client to only update sparse partitions of the server model, enabling AdaSplit to improve performance under limited resources and adapt to variable resource budgets. Further, we also propose a metric (C3-Score) to evaluate distributed deep learning methods under resource budgets jointly. Finally, we validate the effectiveness of AdaSplit through comparisons with strong FL and SL baselines as well as via sensitivity analyses of key design choices.

REFERENCES

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. *ACM SIGSAC*, 2016. doi: 10.1145/2976749.2978318. URL http://dx.doi.org/10.1145/2976749.2978318.

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

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *JMLR*, 2003.

Balakrishnan, R., Li, T., Zhou, T., Himayat, N., Smith, V., and Bilmes, J. Diverse client selection for federated learning: Submodularity and convergence analysis. In *ICML Workshops*, 2020.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. A survey of model compression and acceleration for deep neural networks. *arXiv:1710.09282*, 2017.

Cho, Y. J., Gupta, S., Joshi, G., and Yağan, O. Bandit-based communication-efficient client selection strategies for federated learning. In *Asilomar Conference on Signals, Systems, and Computers*, 2020a.

Cho, Y. J., Wang, J., and Joshi, G. Client selection in federated learning: Convergence analysis and power-of-choice selection strategies. *arXiv:2010.01243*, 2020b.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

Diao, E., Ding, J., and Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv:2010.01264*, 2020.

Elbir, A. M., Soner, B., and Coleri, S. Federated learning in vehicular networks. *arXiv:2006.01412*, 2020.

Epasto, A., Mahdian, M., Mirrokni, V., and Zhong, P. Massively parallel and dynamic algorithms for minimum size clustering. *arXiv:2106.02685*, 2021.

Gawali, M., Arvind, C., Suryavanshi, S., Madaan, H., Gaikwad, A., Prakash, K. B., Kulkarni, V., and Pant, A. Comparison of privacy-preserving distributed deep learning methods in healthcare. In *MIUA*, 2021.

Goddard, M. The eu general data protection regulation (gdpr): European regulation that has a global impact. *IJMR*, pp. 703–705, 2017.

Golkar, S., Kagan, M., and Cho, K. Continual learning via neural pruning, 2019.

Guliani, D., Beaufays, F., and Motta, G. Training speech recognition models with federated learning: A quality/cost framework. In *ICASSP*, 2021.

Gupta, O. and Raskar, R. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 2018.

Hamer, J., Mohri, M., and Suresh, A. T. Fedboost: A communication-efficient algorithm for federated learning. In *ICML*, 2020.

Hard, A. et al. Federated learning for mobile keyboard prediction. *arXiv:1811.03604*, 2018.

He, C. et al. Fedml: A research library and benchmark for federated machine learning. *arXiv:2007.13518*, 2020.

Jiang, Y., Wang, S., Valls, V., Ko, B. J., Lee, W.-H., Leung, K. K., and Tassiulas, L. Model pruning enables efficient federated learning on edge devices, 2020.

Kairouz, P. et al. Advances and open problems in federated learning. *arXiv:1912.04977*, 2019.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. Scaffold: Stochastic controlled averaging for federated learning, 2021.

Konečnỳ, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., and Bacon, D. Federated learning: Strategies for improving communication efficiency. *arXiv:1610.05492*, 2016.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In *NIPS*. Morgan-Kaufmann, 1990.

Li, A., Sun, J., Li, P., Pu, Y., Li, H., and Chen, Y. Hermes: An efficient federated learning framework for heterogeneous mobile clients. 2022.

Li, D. and Wang, J. Fedmd: Heterogenous federated learning via model distillation, 2019.

Li, Q., Diao, Y., Chen, Q., and He, B. Federated learning on non-iid data silos: An experimental study. *arXiv:2102.02079*.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks, 2020.

Li, X.-C. and Zhan, D.-C. *FedRS: Federated Learning with Restricted Softmax for Label Distribution Non-IID Data*, pp. 995–1005. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450383325. URL https://doi.org/10.1145/3447548.3467254.

Madaan, H., Gawali, M., Kulkarni, V., and Pant, A. Vulnerability due to training order in split learning. *arXiv:2103.14291*, 2021.

Malekijoo, A. et al. Fedzip: A compression framework for communication-efficient federated learning. *arXiv:2102.01593*, 2021.

McMahan, H. B., Moore, E., Ramage, D., and y Arcas, B. A. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016.

Mireshghallah, F. et al. Shredder: Learning noise to protect privacy with partial DNN inference on the edge. *CoRR*, 1905.11814, 2019.

Mo, F., Haddadi, H., Katevas, K., Marin, E., Perino, D., and Kourtellis, N. Ppfl: privacy-preserving federated learning with trusted execution environments. *rXiv:2104.14380*, 2021.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. Towards understanding the role of over-parametrization in generalization of neural networks, 2018.

Nishio, T. and Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC*, 2019.

Palanisamy, K. et al. Spliteasy: A practical approach for training ml models on mobile devices. In *ACM HotMobile*, 2021.

Park, J. et al. Communication-efficient and distributed learning over wireless networks: Principles and applications. *arXiv:2008.02608*, 2020.

Pasquini, D., Ateniese, G., and Bernaschi, M. Unleashing the tiger: Inference attacks on split learning. *arXiv:2012.02670*, 2020.

Poirot, M. G., Vepakomma, P., Chang, K., Kalpathy-Cramer, J., Gupta, R., and Raskar, R. Split learning for collaborative deep learning in healthcare. *arXiv:1912.12115*, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Rieke, N. et al. The future of digital health with federated learning. *NPJ digital medicine*, 2020.

Romanini, D. et al. Pyvertical: A vertical federated learning framework for multi-headed splitnn. *arXiv:2104.00489*, 2021.

Samragh, M., Hosseini, H., Azarian, K., and Soriaga, J. Private split inference of deep networks. 2020.

Sharma, V., Vepakomma, P., Swedish, T., Chang, K., Kalpathy-Cramer, J., and Raskar, R. Expertmatcher: Automating ml model selection for users in resource constrained countries. *arXiv:1910.02312*, 2019.

Singh, A., Vepakomma, P., Gupta, O., and Raskar, R. Detailed comparison of communication efficiency of split learning and federated learning. *arXiv:1909.09145*, 2019.

Singh, A., Chopra, A., Garza, E., Zhang, E., Vepakomma, P., Sharma, V., and Raskar, R. Disco: Dynamic and invariant sensitive channel obfuscation for deep neural networks. In *CVPR*, 2021.

Sohn, K. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016.

Stallings, W. Handling of personal information and deidentified, aggregated, and pseudonymized information under the california consumer privacy act. *IEEE Security & Privacy*, 2020.

Thapa, C., Chamikara, M. A. P., and Camtepe, S. A. Advancements of federated learning towards privacy preservation: from federated learning to split learning. In *Federated Learning Systems*. 2021.

Thapa, C., Mahawaga Arachchige, P. C., Camtepe, S., and Sun, L. Splitfed: When federated learning meets split learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8485–8493, Jun. 2022. doi: 10.1609/aaai.v36i8.20825. URL https://ojs.aaai.org/index.php/AAAI/article/view/20825.

Vapnik, V. Principles of risk minimization for learning theory. In *NIPS*, 1992.

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

Vepakomma, P., Singh, A., Gupta, O., and Raskar, R. Nopeek: Information leakage reduction to share activations in distributed deep learning. *arXiv:2008.09161*, 2020.

Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. Tackling the objective inconsistency problem in heterogeneous federated optimization, 2020.

Yang, Q., Zhang, J., Hao, W., Spell, G. P., and Carin, L. *FLOP: Federated Learning on Medical Datasets Using Partial Networks*, pp. 3845–3853. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450383325. URL https://doi.org/10.1145/3447548.3467185.

Yu, F., Zhang, W., Qin, Z., Xu, Z., Wang, D., Liu, C., Tian, Z., and Chen, X. *Fed2: Feature-Aligned Federated Learning*, pp. 2066–2074. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450383325. URL https://doi.org/10.1145/3447548.3467309.

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. Scaling vision transformers. *arXiv:2106.04560*, 2021.

Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., and Chandra, V. Federated learning with non-iid data. *arXiv:1806.00582*, 2018.

Zhou, G., Xu, K., Li, Q., Liu, Y., and Zhao, Y. Adaptcl: Efficient collaborative learning with dynamic and adaptive pruning. *arXiv:2106.14126*, 2021.