This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

MAS: Towards Resource-Efficient Federated Multiple-Task Learning

Weiming Zhuang[1]* Yonggang Wen[2] Lingjuan Lyu[1] Shuai Zhang[3]
[1]Sony AI, [2]Nanyang Technological University, [3]SenseTime Research
{weiming.zhuang,lingjuan.lv}@sony.com,ygwen@ntu.edu.sg,zhangshuai@sensetime.com

Abstract

Federated learning (FL) is an emerging distributed machine learning method that empowers in-situ model training on decentralized edge devices. However, multiple simultaneous FL tasks could overload resource-constrained devices. In this work, we propose the first FL system to effectively coordinate and train multiple simultaneous FL tasks. We first formalize the problem of training simultaneous FL tasks. Then, we present our new approach, MAS (Merge and Split), to optimize the performance of training multiple simultaneous FL tasks. MAS starts by merging FL tasks into an all-in-one FL task with a multi-task architecture. After training for a few rounds, MAS splits the all-in-one FL task into two or more FL tasks by using the affinities among tasks measured during the all-in-one training. It then continues training each split of FL tasks based on model parameters from the all-in-one training. Extensive experiments demonstrate that MAS outperforms other methods while reducing training time by 2× and reducing energy consumption by 40%. We hope this work will inspire the community to further study and optimize training simultaneous FL tasks.

1. Introduction

Federated learning (FL) [34] has attracted considerable attention as it enables privacy-preserving distributed model training among decentralized devices. It is empowering growing numbers of applications in both academia and industry, such as medical imaging analysis [29, 41], Google Keyboard [16], and autonomous vehicles [55, 39]. Among them, some applications contain multiple application tasks. For example, autonomous vehicles are related to multiple resource-intensive computer vision (CV) tasks, including lane detection, object detection, and segmentation [20].

In fact, the majority of edge devices (e.g., NVIDIA Jeston TX2 and AGX Xavier) can only support one FL task at a time [30]. Multiple simultaneous FL tasks on the same device could overwhelm its memory, computation, and power capacities. Thus, it is important to navigate solutions to well coordinate these simultaneous FL tasks.

*most of the work done in S-Lab, Nanyang Technological University

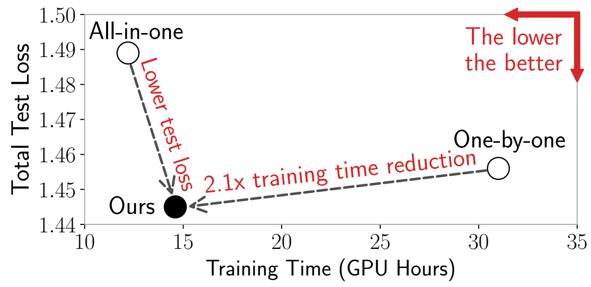


Figure 1: Existing methods suffer from a trade-off between training time and test loss (lower test loss means better performance) when training 9 simultaneous FL tasks, whereas our method navigates a sweet point, achieving the best test loss with 2.1× training time reduction. One-by-one trains FL tasks one after another; All-in-one combines tasks into a multi-task learning network before training it in FL.

A plethora of research on FL are mainly devoted to addressing challenges such as statistical heterogeneity [28, 48], system heterogeneity [6, 51, 56], communication efficiency [23, 61, 27, 50], and privacy issues [2, 19]. Most of the existing works only focus on one FL task, overlooking the fact that certain applications, such as self-driving cars or intelligent manufacturing robots, need to tackle multiple FL tasks simultaneously [20, 13]. To address this issue, Bonawitz et.al [4] designed multi-tenancy to prevent simultaneous FL tasks from overloading devices. However, their proposed method is slow in training because it regards these tasks as independent training tasks and trains them sequentially. This *one-by-one* training method only considers the differences among tasks, neglecting potential synergies.

Another intuitive solution is to adopt multi-task learning (MTL) to train multiple FL tasks by combining these tasks into an *all-in-one* neural network. This network has one encoder shared among tasks and multiple task-specific decoders. It could prevent overloading devices and speeds up the training process as only one neural network is trained. However, it could result in worse performance because not all tasks are beneficial to the others when training together [22, 60]. Simply combining FL tasks together only takes into account their synergies while overlooking their distinctions. Figure 1 shows that either the all-in-one (only considers task synergies) or the one-by-one method (only considers task differences) suffers from a trade-off between training time and test loss.

In this work, we propose MAS (i.e. Merge and Split), the first FL system to effectively coordinate and train multiple simultaneous FL tasks under resource constraints by considering both synergies and differences among these tasks. We first formalize the problem of training multiple simultaneous FL tasks. To address this problem, we introduce MAS to optimize the performance. Specifically, MAS starts by merging these FL tasks into an all-in-one FL task with a multi-task architecture, which shares common layers and has specialized layers for each task. After training the all-in-one FL task for certain rounds, MAS splits this all-in-one task into two or more FL tasks based on their synergies and differences measured by affinity scores during training. Lastly, MAS continues training each split of FL tasks with models trained in the all-in-one process.

Figure 1 shows that MAS achieves the best test loss with $2.1\times$ training time reduction compared to the one-by-one method on training nine FL tasks. We also demonstrate that it reduces energy consumption by over 40% while achieving superior performance to other methods via extensive experiments on three different sets of FL tasks. We believe that MAS is beneficial for many real-world applications such as autonomous vehicles and robotics. We summarize our contributions as follows:

- We formalize the problem of training multiple simultaneous FL tasks. To the best of our knowledge, we are the first to conduct an in-depth investigation into the training of multiple simultaneous FL tasks.
- We propose MAS, a new FL system to effectively coordinate and train simultaneous FL tasks by considering both synergies and differences among these tasks.
- We establish baselines for training multiple simultaneous FL tasks and demonstrate that MAS elevates performance with significantly less training time and energy consumption via extensive empirical studies.

2. Related Work

In this section, we provide a literature review of federated learning and multi-task learning.

Federated Learning emerges as a privacy-aware and distributed learning paradigm that uses a central server to coordinate multiple decentralized clients to train models [34, 21]. The majority of studies aim to address the challenges of FL, including statistical heterogeneity [28, 48, 49, 66, 53, 58, 65, 45, 57, 11, 64], system heterogeneity [6, 51, 31], communication efficiency [34, 25, 23, 61], and privacy concerns [2, 19]. Numerous methods are proposed to cluster FL clients into groups to address statistical heterogeneity [14, 37, 63]. They aim to cluster models that are trained on clients with similar distribution, whereas our proposed MAS differs fundamentally from these methods as it splits simultaneous FL tasks into groups. Several other attempts have been made [42, 33] on *federated multi-task learning* in order to learn personalized models to tackle statistical heterogeneity. These personalized FL methods mainly focus on training one FL task of an application in a client. Training multiple simultaneous FL tasks is rarely explored. The prior work [4] designs multi-tenancy in an FL system to schedule and train these tasks sequentially. This one-by-one method is slow in training and only considers the differences among these FL tasks.

Multi-task Learning is a popular machine learning approach to learn models that can generalize on multiple tasks [46, 59]. A plethora of studies investigate parameter sharing approaches that share common layers of a similar architecture [5, 10, 3, 36]. Besides, many studies employ new techniques to address the negative transfer problem [22, 60] among tasks, including soft parameter sharing [9, 35], neural architecture search [32, 18, 47, 15, 44], and dynamic loss reweighting strategies [24, 7, 52]. Instead of training all tasks together, task grouping trains only similar tasks together. The early works of task grouping [22, 26] are not adaptable to DNN. Recently, several studies analyze task similarity [43] and task affinities [12] for task grouping. The state-of-the-art task grouping methods [43, 12], however, are unsuitable for training multiple simultaneous FL tasks because they mainly focus on inference efficiency. They would train a task multiple times as their task groups always contain overlapped tasks. This motivates us to exploit task merging and task splitting to group and train multiple simultaneous FL tasks.

3. Method

In this section, we start by providing problem definition for training multiple simultaneous FL tasks. Then, we propose Merge and Split (MAS) method that first merges tasks into an all-in-one FL task and then splits it into two or more splits for further training.

3.1. Problem Definition

In the federated learning setting, the majority of studies consider optimizing the following problem:

$$\min_{\omega \in \mathbb{R}^d} f(\omega) := \sum_{k=1}^{K} p_k f_k(\omega) := \sum_{k=1}^{K} p_k \mathbb{E}_{\xi_k \sim \mathcal{D}_k}[f_k(\omega; \xi_k)], \tag{1}$$

where ω is the optimization variable, K is the number of selected clients to execute training, $f_k(\omega)$ is the loss function of client k, p_k is the weight of client k in model aggregation, and ξ_k is the training data sampled from data distribution $\mathcal{D}_k$ of client k. `FedAvg` [34] is a popular federated learning algorithm, which sets p_k to be proportional to the dataset size of client k.

In fact, Equation 1 only illustrates the objective of training a single FL task. In real-world scenarios, an FL server could receive multiple simultaneous FL tasks, denoted as a set $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. These tasks aim to train a set of models $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_n\}$, where each model ω_i is for task α_i. By defining $\mathcal{M}(\alpha_i; \omega_i)$ as the performance measurement of each FL task α_i, the overall objective is to maximize the performance of all FL tasks $\sum_{i=1}^{n} \mathcal{M}(\alpha_i; \omega_i)$ with minimum training time, under the constraint that each client k has limited memory budget and computation budget. These budgets constrain the number of simultaneous FL tasks n_k on client k. Besides, as devices have limited battery life, it is important to minimize the energy consumption and training time to obtain $\mathcal{W}$ for FL tasks $\mathcal{A}$.

In this work, we assume that each client can execute one FL task at a time ($n_k = 1$). This is common for the majority of current edge devices[1]. Besides, we assume that the models $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_n\}$ share the same backbone architecture but have different decoder architectures. This is a practical assumption for many real-world applications and industrial practices [40, 13].

3.2. Architecture Overview

Figure 2 depicts the architecture overview and training process of our proposed MAS, which trains simultaneous FL tasks efficiently by considering both synergies and differences among these tasks. It contains a server to coordinate FL tasks and a pool of clients to execute training.

The training process of MAS is as follows: 1) The server receives multiple FL tasks $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ to train models $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_n\}$ and merges these FL tasks into an all-in-one FL task α_0 with a multi-task model ϕ. 2) The server schedules α_0 to train ϕ. 3) The server iteratively selects K clients from the client pool to train α_0 through FL process for R_0 rounds. In each round, the server sends model ϕ to the selected clients; the clients train ϕ and calculate task affinity scores $\hat{\mathcal{S}}$ before uploading these training updates to the server. 4) The server uses the affinity scores $\hat{\mathcal{S}}$ to split the all-in-one FL task α_0 into two or more FL task splits $\{\mathcal{A}_1, \mathcal{A}_2, \ldots\}$, where each split trains non-overlapping subset of $\mathcal{W}$. The number of splits can be determined by the inference budget for the number of concurrent models. 5) The server iterates steps 2 and 3 to train $\mathcal{A}_j$. We summarize MAS in Algorithm 1 and the network architecture changes in Figure 3.

[1]Edges devices, e.g., NVIDIA Jetson TX2 and AGX Xavier, have only one GPU; GPU virtualization [17] that enables concurrent training on the same GPU currently are mainly for the cloud stack.

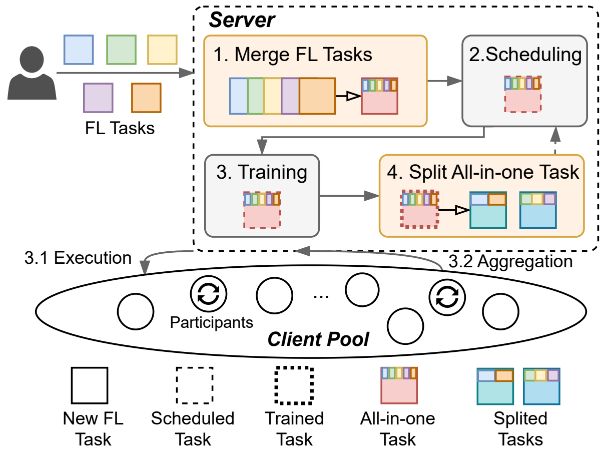


Figure 2: The architecture and workflow of our proposed Merge and Split (MAS). The server first merges multiple simultaneous FL tasks into an all-in-one FL task. After scheduling and training this task in FL for certain rounds, MAS splits the all-in-one FL task into two or more splits based on task affinities measured during training. It considers both synergies and differences among FL tasks by splitting the tasks with higher synergies in the same split.

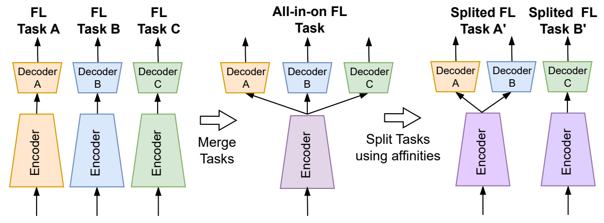


Figure 3: Illustration of network architecture changes in MAS. Initially, each FL task employs an encoder and a decoder. MAS first merges these FL tasks into an all-in-one FL task with a multi-task architecture. Then, it splits the all-in-one FL task into two or more splits.

3.3. Merge FL Tasks Into an All-in-one Task

The server receives multiple FL tasks and merges them into an all-in-one FL task with multi-task architecture. This is based on the practical assumption discussed in Section 3.1 that the models of these FL tasks could share similar model architecture – sharing the same backbone architecture and having task-specific decoders. We can merge these FL tasks into an all-in-one FL task α_0 that trains a multi-task model $\phi = \{\theta_s\} \cup \{\theta_{\alpha_i} | \alpha_i \in \mathcal{A}\}$, where θ_s is the shared model parameters and θ_{α_i} is the specific parameters for FL task $\alpha_i \in \mathcal{A}$. The loss function for training the all-in-one FL task in each client is as followed:

$$\mathcal{L}(\mathcal{X}, \theta_s, \{\theta_{\alpha_i}\}) = \sum_{\alpha_i \in \mathcal{A}} \mathcal{L}_{\alpha_i}(\mathcal{X}, \theta_s, \theta_{\alpha_i}), \qquad (2)$$

where $\mathcal{X}$ is batch of data and $\mathcal{L}_{\alpha_i}$ denotes the loss function of each FL task $\alpha_i \in \mathcal{A}$. This formulation is generally applicable to different loss weights, but we set the loss weight to one for simplicity of notations.

Merging FL tasks into an all-in-one task effectively reduces the training time (Figure 1), as we only need to train one task instead of multiple tasks sequentially. However, simply training with the all-in-one FL task leads to unsatisfactory performance on total test loss, because it only considers synergies among tasks, neglecting the negative transfer problems [22, 60] in multi-task learning. Consequently, we further propose to split the all-in-one FL task considering both synergies and differences among these tasks.

3.4. Split All-in-one FL Task Into Multiple Splits

MAS divides the all-in-one FL task α_0 into multiple splits after it is trained for certain rounds. Essentially, we aim to split $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ into multiple non-overlapping groups such that FL tasks within a group have better synergy. Let $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m\}$ be subsets of $\mathcal{A}$, we aim to find a disjoint set I of $\mathcal{A}$, where $I \subseteq \{1, 2, \ldots, m\}$, $|I| \leq |\mathcal{A}|$, $\bigcup_{j \in I} \mathcal{A}_j = \mathcal{A}$, and $\bigcap_{j \in I} \mathcal{A}_j = \emptyset$. Each split $\mathcal{A}_j$ trains a model $\phi_j = \{\theta_s^j\} \cup \{\theta_{\alpha_i} | \alpha_i \in \mathcal{A}_j\}$, which is a multi-task network when $\mathcal{A}_j$ contains more than one FL task, where θ_s^j is the shared model parameters and θ_{α_i} is the specific parameters for FL task $\alpha_i \in \mathcal{A}_j$. The core question is how to determine set I to split these FL tasks considering their synergies and differences.

Inspired by TAG [12] that measures task affinities for task grouping, we employ affinities among multiple simultaneous FL tasks for splitting via four stages: 1) Each client measures affinities among FL tasks during *all-in-one* training every ρ batchs and averages them over E local epoch; 2) The server obtains affinity scores by aggregating the affinities over K participating clients; 3) The system computes the affinity scores of different combinations of subsets of FL tasks and select the best subset that achieve the highest affinity score; 4) The server splits the all-in-one model ϕ following the combination of tasks and continues training each split with its model initialized with parameters obtained from all-in-one training. Particularly, during training of all-in-one FL task α_0, we measure the affinity of FL task α_i onto α_j at time step t in client k with the equation:

$$\mathcal{S}^{k,t}_{\alpha_i \to \alpha_j} = 1 - \frac{\mathcal{L}_{\alpha_j}(\mathcal{X}^{k,t}, \theta^{k,t+1}_{s,\alpha_i}, \theta^{k,t}_{\alpha_j})}{\mathcal{L}_{\alpha_j}(\mathcal{X}^{k,t}, \theta^{k,t}_{s}, \theta^{k,t}_{\alpha_j})}, \qquad (3)$$

where $\mathcal{L}_{\alpha_j}$ is the loss function of α_j, $\mathcal{X}^{k,t}$ is a batch of training data, and $\theta_s^{k,t}$ and $\theta^{k,t+1}_{s,\alpha_i}$ are the shared model parameters *before* and *after* updated by α_i, respectively. A positive value of $\mathcal{S}^{k,t}_{\alpha_i \to \alpha_j}$ means that task α_i helps reduce the loss of α_j; the higher value of $\mathcal{S}^{k,t}_{\alpha_i \to \alpha_j}$ suggests that these two tasks are better to train together. This equation measures the affinity of one time step of one client. We approximate affinity scores for each round by averaging the values over T time steps in E local epochs and K selected clients: $\hat{\mathcal{S}}_{\alpha_i \to \alpha_j} = \frac{1}{KET} \sum_{k=1}^{K} \sum_{e=1}^{E} \sum_{t=1}^{T} \mathcal{S}^{k,t}_{\alpha_i \to \alpha_j}$, where T is the total time steps determined by the frequency ρ of calculating Equation 3, e.g., $\rho = 5$ means measuring the affinity in each client in every five batches.

These affinity scores measure pair-wise affinities between FL tasks. We next use them to calculate total affinity scores of a split with $\sum_{i=1}^{n} \hat{\mathcal{S}}_{\alpha_i}$, where $\hat{\mathcal{S}}_{\alpha_i}$ is the averaged affinity score onto each FL task. For example, a grouping of two splits among five FL tasks is $\{\alpha_1, \alpha_2\} and \{\alpha_3, \alpha_4, \alpha_5\}$, where $\{,\}$ denotes a split. The affinity score onto α_1 is $\hat{\mathcal{S}}_{\alpha_1} = \hat{\mathcal{S}}_{\alpha_2 \to \alpha_1}$ and the affinity score onto α_3 is $\hat{\mathcal{S}}_{\alpha_3} = (\hat{\mathcal{S}}_{\alpha_4 \to \alpha_3} + \hat{\mathcal{S}}_{\alpha_5 \to \alpha_3})/2$. Consequently, we can find the set I with $|I|$ elements for subsets of $\mathcal{A}$ that maximize $\sum_{i=1}^{n} \hat{\mathcal{S}}_{\alpha_i}$, where $|I|$ defines the number of elements.

It is important to note the differences between our method and TAG [12]. Firstly, TAG focuses on inference efficiency, thus it allows overlapping task grouping that could train one task multiple times. In contrast, our focus is fundamentally different: we focus on training efficiency and consider only non-overlapping splitting of FL tasks. Secondly, TAG is computation-intensive for higher numbers of splits, e.g., it fails to produce results of five splits of nine tasks in a week, whereas we only need seconds of computation. Thirdly, TAG rules out the possibility that a split contains only one task. The calculated value of $\hat{\mathcal{S}}\alpha_i \to \alpha_i$ from Equation 3 is larger than the values of $\hat{\mathcal{S}}\alpha_i \to \alpha_j$, where $i \neq j$. As a result, one task $\alpha_i \in \mathcal{A}$ consistently receives the highest score during splitting and is always assigned as a group. TAG sets $\hat{\mathcal{S}}_{\alpha_i \to \alpha_i} = 1\mathrm{e}^{-6}$, resulting in no group contains only a single task because the scores of other combinations are larger than $1e^{-6}$. To overcome these issues, we propose a new method to calculate the value as follows:

$$\hat{\mathcal{S}}_{\alpha_i \to \alpha_i} = \sum_{j \in \mathcal{N} \setminus \{i\}} \frac{(\hat{\mathcal{S}}_{\alpha_i \to \alpha_j} + \hat{\mathcal{S}}_{\alpha_j \to \alpha_i})}{2n - 2}, \qquad (4)$$

where $\mathcal{N} = \{1, 2, \ldots, n\}$. The intuition of this equation is to measure the normalized affinity of task α_i to other tasks and other tasks to α_i, thus, it is dubbed self-affinity. Equation 4 overrides the values in Equation 3 in affinity score calculation. Fourthly, we focus on training multiple simultaneous FL tasks, thus, we further aggregate affinity scores over K selected clients. Finnaly, TAG trains each set $\mathcal{A}_j$ from scratch, whereas we initialize their models with the parameters obtained from all-in-one training. Table 1 shows that this change significantly boosts the performance.

Algorithm 1 Our Proposed MAS

```
1: Input: FL tasks $\mathcal{A} = \{\alpha_1, \dots, \alpha_n\}$, available clients $\mathcal{C}$, number of selected clients $K$, local epoch $E$, aggregation weight of client $k$ $p_k$, training rounds $R$, all-in-one training rounds $R_0$, the number of splits $x$, frequency of computing affinities $\rho$, batch size $B$
2: Output: models $\mathcal{W} = \{\omega_1, \omega_2, \dots, \omega_n\}$
3:
4: ServerExecution:
5: Receive FL tasks $\mathcal{A}$ and merge models $\mathcal{W}$ into a multi-task model $\phi^0 = \{\theta_s\} \cup \{\theta_{\alpha_i} | \alpha_i \in \mathcal{A}\}$    ▷ Merging
6: Initialize $\phi^0$
7: for each round $r = 0, 1, ..., R_0 - 1$ do
8:     $\mathcal{C}^r \leftarrow$ (Randomly select K clients from $\mathcal{C}$)
9:     for client $k \in \mathcal{C}^r$ in parallel do
10:        $\phi^{k,r}, \hat{\mathcal{S}}^{k,r}_{\alpha_i \to \alpha_j} \leftarrow$ ClientExecution($\phi^r$, $\mathcal{A}$, $\rho$)
11:    $\phi^{r+1} \leftarrow \sum_{k \in \mathcal{C}^r} p_k \phi^{k,r}$
12:    $\hat{\mathcal{S}}^r_{\alpha_i \to \alpha_j} \leftarrow \frac{1}{K} \sum_{k \in \mathcal{C}^r} \hat{\mathcal{S}}^{k,r}_{\alpha_i \to \alpha_j}$
13: Compute self-affinity $\hat{\mathcal{S}}^r_{\alpha_i \to \alpha_i}$ using Eqn. 4
14: Compute a disjoint partition set $I$ of FL tasks $\mathcal{A}$ for $x$ splits $\{\mathcal{A}_j | j \in I\}$ that maximizes $\hat{\mathcal{S}}^r_{\alpha_i}$ using affinity scores $\hat{\mathcal{S}}^r_{\alpha_i \to \alpha_j}, \forall \alpha_i, \alpha_j \in \mathcal{A}$    ▷ Splitting
15: for each element $j \in I$ do    ▷ Schedule to train
16:    Initialize $\phi_j = \{\theta^j_s\} \cup \{\theta_{\alpha_i} | \alpha_i \in \mathcal{A}_j\}$ with parameters of $\phi$
17:    for each round $r = 0, 1, ..., R - R_0 - 1$ do
18:        $\mathcal{C}^r \leftarrow$ (Random select K from $\mathcal{C}$)
19:        for client $k \in \mathcal{C}^r$ in parallel do
20:            $\phi^{k,r}_j$, _ $\leftarrow$ ClientExecution($\phi^r_j$, $\mathcal{A}_j$, 0)
21:        $\phi^{r+1}_j \leftarrow \sum_{k \in \mathcal{C}^r} p_k \phi^{k,r}_j$
22: Reconstruct $\mathcal{W} = \{\omega_1, \omega_2, \dots, \omega_n\}$ from $\{\phi_j | j \in I\}$
23: Return $\mathcal{W}$
24:
25: ClientExecution ($\phi$, $\mathcal{A}$, $\rho$):
26: $T = \lfloor \frac{B}{\rho} \rfloor$ if $\rho \neq 0$ else 0
27: for local epoch $e = 1, ..., E$ do
28:    Update model parameters $\phi$ w.r.t FL tasks $\mathcal{A}$
29:    for each time-step $t = 1, ..., T$ (every $\rho$ batches) do
30:        Compute $\mathcal{S}^t_{\alpha_i \to \alpha_j}$ using Eqn. 3, $\forall \alpha_i, \alpha_j \in \mathcal{A}$
31: $\hat{\mathcal{S}}_{\alpha_i \to \alpha_j} = \frac{1}{ET} \sum_{e=1}^{E} \sum_{t=1}^{T} \mathcal{S}^t_{\alpha_i \to \alpha_j}, \forall \alpha_i, \alpha_j \in \mathcal{A}$
32: Return $\phi$, $\hat{\mathcal{S}}_{\alpha_i \to \alpha_j}$
```

4. Experiments

We evaluate the performance and resource usage of MAS and study the following questions: 1) How effective are the splits from MAS? 2) When to split the all-in-one FL task? 3) How much MAS can outperform the baseline of training each client independently? 4) What are the impacts of local epoch and the number of selected clients?

4.1. Experiment Setup

Dataset and Federated Simulation. We construct our experiments using Taskonomy dataset [54], which is a large and challenging computer vision dataset of indoor scenes of buildings. We run experiments with $N = 32$ clients, where each client contains a dataset of one building to simulate the statistical heterogeneity. Figure 4 shows the data amount distribution over 32 clients; some clients have only 4,000 images, whereas some clients have over 16,000 images. We design three sets of FL tasks to evaluate the robustness of MAS under different combintations and different numbers of CV tasks. These three sets are `sdnkt`, `erckt`, and `sdnkterca`; each character represents an FL task[2]. The `sdnkterca` set is especially challenging with 9 FL tasks.

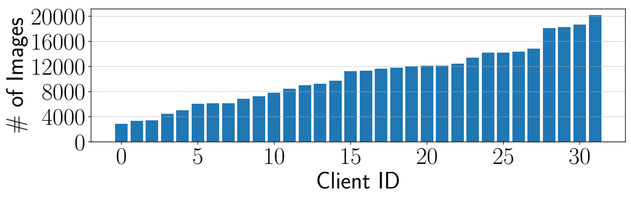
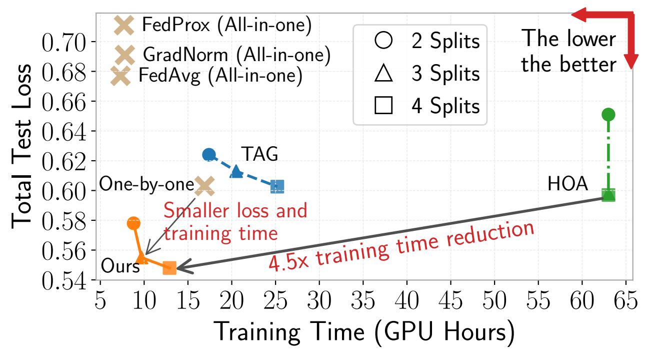


Figure 4: The data amount distribution of 32 FL clients. It simulates the challenging statistical heterogeneity.

Implementation Details. We implement our proposed MAS by PyTorch [38] and EasyFL [62] [3]. We simulate the FL training on a cluster of NVIDIA Tesla V100 GPUs, where each node in the cluster contains 8 GPUs. In each round, each selected client is allocated to a GPU to conduct training; these clients communicate via the NCCL backend. Besides, we employ FedAvg [34] for the server aggregation. By default, we randomly select $K = 4$ clients to train for $E = 1$ local epochs in each round and train for $R = 100$ rounds. The batch size is $B = 64$ for `sdnkt` and `erckt` and $B = 32$ for `sdnkterca`. We use the modified Xception Network [8] as the encoder for FL tasks of `sdnkt` and `erckt`, and *half size* of the network (half amount of parameters) for FL tasks of `sdnkterca`. The decoders contain four deconvolution layers and four convolution layers. The optimizer is stochastic gradient descent (SGD), with momentum of 0.9 and weight decay $1e^{-4}$. The learning rate is initiated as $\eta = 0.1$ and is updated with polynomial learning rate decay $(1 - \frac{r}{R})^{0.9}$ in each round, where r is the number

[2]The meaning of each character in `sdnkterca` are as follows; `s`: semantic segmentation, `d`: depth estimation, `n`: normals, `k`: keypoint, `t`: edge texture, `e`: edge occlusion, `r`: reshaping, `c`: principle curvature, `a`: auto-encoder.

[3]Our codes are available at https://github.com/EasyFL-AI/EasyFL/



Method	Test Loss	Training Time (GPU × hours)	Energy (Kwh)
One-by-one	0.603 ± 0.030	16.9 ± 0.5	8.4 ± 0.1
FedAvg [a]	0.677 ± 0.018	**7.3** ± 0.3	**3.7** ± 0.1
FedProx [a]	0.711 ± 0.070	7.7± 0.5	4.4 ± 0.7
GradNorm [a]	0.691 ± 0.013	7.8 ± 0.6	4.1 ± 0.4
HOA-2	0.651 ± 0.029	63.0 ± 0.9	31.0 ± 0.5
HOA-3	0.598 ± 0.029	63.0 ± 0.9	31.0 ± 0.5
HOA-4	0.597 ± 0.015	63.0 ± 0.9	31.0 ± 0.5
TAG-2	0.624 ± 0.015	17.4 ± 0.5	9.8 ± 0.3
TAG-3	0.613 ± 0.032	20.5 ± 0.7	11.3 ± 0.2
TAG-4	0.603 ± 0.027	25.2 ± 0.8	13.7 ± 0.3
MAS-2	**0.578** ± 0.015	8.8 ± 0.5	4.9 ± 0.3
MAS-3	**0.555** ± 0.015	9.7 ± 0.5	5.4 ± 0.3
MAS-4	**0.548** ± 0.001	12.9 ± 0.6	6.7 ± 0.3

[a]All-in-one methods

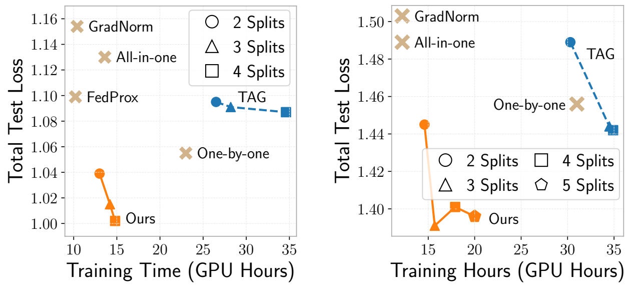
Figure 5: Comparison of test loss, training time, and energy consumption on a set of five FL tasks `sdnkt`, where each character represents an FL task. The figure visualizes the table on test loss and training time. Our proposed MAS achieves the best performance with only slight increases in training time than all-in-one methods, but it requires significant less training time than the other methods.

of trained rounds. We measure the statistical performance of a FL task set using the sum of test losses and measure the training time and energy consumption [1]. Most of the experimental results are from three independent runs. More experimental details are provided in the supplementary.

4.2. Performance Evaluation

We conduct the performance comparison, in terms of total test loss of all tasks, training time, and energy consumption, among the following methods: 1) one-by-one training of FL tasks; 2) all-in-one training of FL tasks using FedAvg; 3) all-in-one training with multi-task optimization (GradNorm [7]) and federated optimization (FedProx [28]); 4) HOA [43] method that groups FL tasks by estimating higher-order of groupings from pair-wise tasks performance and then trains each group from scratch; 5) TAG [12] method that groups FL tasks only based on task affinity and trains each group from scratch; 6) Our proposed MAS.

Figure 5 compares the performance of the above methods on a set of five simultaneous FL tasks `sdnkt`. The methods that achieve lower total test loss and lower energy consumption are better. At the one extreme, all-in-one methods consume the least training time and energy, but their test losses are the highest. Simply applying multi-task learning optimization (GradNorm [7]) or federated optimization (FedProx [28]), can hardly improve performance. At the other extreme, HOA can achieve comparable test losses, but it demands long training time and high energy consumption (around $4-6\times$ of ours) to compute pair-wise tasks for higher-order estimation. Although the one-by-one method and TAG [12] present a good balance between test loss and system metric, MAS is superior in both aspects; it achieves the best test loss with $\sim 2\times$ reduction on training time and $\sim 40\%$ less energy consumption. We evaluate the performance of splitting the all-in-one FL task into 2, 3, and 4 splits in our method, denoted as MAS-2, MAS-3, and MAS-4, respectively. More splits lead to longer training time, but it could help further reduce test losses.



(a) Five tasks: `erckt` (b) Nine tasks: `sdnkterca`

Figure 6: Comparison of test loss and training time on two FL task sets: (a) `erckt` and (b) `sdnkterca`. Our method achieves the best performance with only a slight increase in training time than all-in-one methods and much less training time than the other methods.

In addition, Figure 6 further compares the performance of total test loss and training time on another five-task set `erckt` and a nine-task set `sdnkterca`. They generally achieve similar results as task set `sdnkt`. In set `sdnkterca`, we further find that splitting into more splits may not always improve the total test loss, though they are still better than other methods. The reason could be that it is easier to find tasks that have better synergies when training with more simultaneous tasks. HOA on `erckt` is at least $5.5\times$ slower than our method. We do not report HOA for `sdnkterca` due to computation constraints; HOA com-

Task Set	Splits	Ours	Train from Scratch		Train from Initialization	
			Optimal	Worst	Optimal	Worst
sdnkt	2	**0.578** ± **0.015**	0.622 ± 0.007	0.685 ± 0.010	0.595 ± 0.008	0.595 ± 0.004
	3	**0.555** ± **0.008**	0.585 ± 0.026	0.674 ± 0.022	0.560 ± 0.006	0.578 ± 0.006
erckt	2	**1.039** ± **0.024**	1.070 ± 0.013	1.312 ± 0.065	1.048 ± 0.024	1.068 ± 0.037
	3	**1.015** ± **0.018**	1.058 ± 0.029	1.243 ± 0.099	1.020 ± 0.012	1.052 ± 0.026

Table 1: Performance (total test loss) comparison of our proposed MAS with the optimal and worst splits. Our method achieves the best performance, indicating the effectiveness of the splits obtained from MAS.

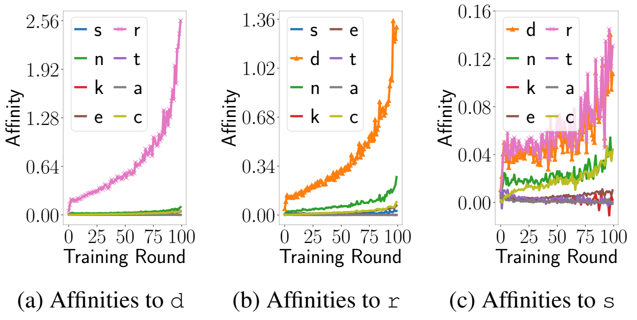


Figure 7: Changes of affinity scores of one FL task to the other on task set sdnkterca. FL task d and r have high inter-task scores. The trends of affinities emerge at the early stage of training.

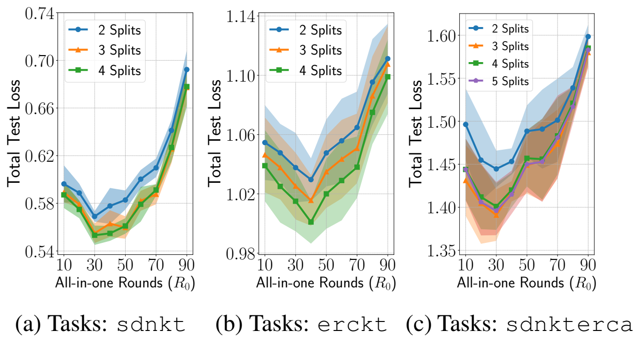


Figure 8: Comparison of training all-in-one tasks for different R_0 rounds. Fixing the total training rounds $R = 100$, MAS achieves the best performance when $R_0 \in \{30, 40\}$.

putes at least 36 pairs of FL tasks (∼720 GPU hours).

4.3. How Effective are the Splits From MAS?

We demonstrate the effectiveness of the splitting method in MAS by comparing it with the performance of splits with possible optimal and worst splits. The optimal and worst splits are obtained with two steps: 1) we measure the performance over all combinations of two splits and three splits of an FL task set by training them from scratch;[4] 2) we select the combination that yields the best performance as the optimal split and the worst performance as the worst split.

Table 1 compares the total test loss of MAS with the optimal and worst splits trained in two ways: 1) training each split from scratch; 2) training each split based on model parameters obtained from all-in-one training, which is adopted in our method but not in TAG [12]. On the one hand, training from initialization outperforms training from scratch in all settings. It suggests that initializing each split with all-in-one training model parameters can significantly improve the performance. On the other hand, our splitting method achieves the best performance in all settings, even though training from initialization reduces the gaps of different splits (the optimal and worst splits). These results indicate the effectiveness of the splits obtained from MAS.

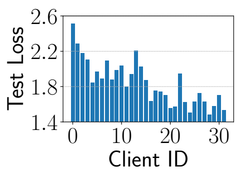
[4]There are fifteen and twenty-five combinations of two and three splits, respectively, for a set of five tasks.

4.4. When to Split the All-in-one FL Task?

We further answer the question that how many R_0 rounds should we train the all-in-one FL task before splitting. It is determined by two factors: 1) the rounds needed to obtain affinity scores for a reasonable splitting; 2) the rounds that yield the best overall performance.

Affinity Analysis. We analyze changes in affinity scores over the course of training to show that MAS can use early-stage affinity scores for splitting. Figure 7 presents the affinity scores of different FL tasks to one FL task on task set sdnkterca. Figure 7a and 7b indicate that FL task d and FL task r have high inter-task affinity scores; they are split into the same group as a result. In contrast, both d and r have high-affinity score to FL task s in Figure 7c, but not vice versa. These trends emerge in the early stage of training, thus, we employ the affinity scores of the *10-th* round for splitting by default; they are effective in achieving promising results as shown in Figure 5 and 6. We provide more affinity scores of FL tasks in the supplementary.

The Impact of R_0 Rounds. Figure 8 compares the performance of training R_0 for 10 to 90 rounds before splitting. Fixing the total training round $R = 100$, we train each split of FL tasks for $R_1 = R - R_0$ rounds. The results indicate that MAS achieves the best performance when $R_0 = \{30, 40\}$ rounds. Training the all-in-one FL task for enough rounds helps utilize the benefits and synergies of



(a) Test loss distribution of standalone training

Method	Total Test Loss
Standalone	1.842
All-in-one	0.677
MAS-2	0.578

(b) Test loss comparison

Figure 9: Performance of standalone training that conducts training using data in each client independently on task set `sdnkt`: (a) shows test loss distribution of 32 clients. (b) compares test losses of standalone training and FL methods.

training together, but training for too many rounds almost suppresses the benefits of considering differences among FL tasks. We suggest training R_0 for [30, 40] rounds to strike a good balance between these two extremes from the above empirical results and consider other mechanisms to determine R_0 in future works.

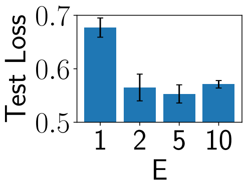
4.5. Additional Analysis

We further analyze the performance of standalone training, the impact of local epoch E, and the impact of the number of selected clients K in FL using all-in-one training. We report the results of FL task set `sdnkt` here and provide more results in the supplementary.

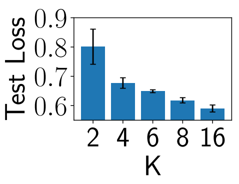
Standalone Training. Standalone training refers to training using data of each client independently. Figure 9a shows the test loss distribution of 32 clients in experiments. The client ID corresponds to the dataset size distribution in Figure 4. These results suggest that clients with larger data sizes may not lead to higher performance. Figure 9b compares test losses of standalone training and FL methods. Either all-in-one or our MAS greatly outperforms standalone training. It suggests the significance of federated learning when data are not shareable among clients.

Impact of Local Epoch E. Local epoch defines the number of epochs each client trains before uploading training updates to the server. Figure 10a compares test losses of local epochs $E = \{1, 2, 5, 10\}$. Larger E could lead to better performance with higher computation (fixed training round $R = 100$), but it is not effective when increasing $E = 5$ to $E = 10$. It suggests the limitation of simply increasing computation with larger E in improving performance. Note that MAS (Table 1) achieves better results than $E = 5$ with $\sim 5\times$ less computation.

Impact of The Number of Selected Clients K. Figure 10b compares test losses of the number of selected clients $K = \{2, 4, 6, 8, 16\}$ in each round. Increasing the number of selected clients improves the performance, but the effect becomes marginal as K increases. Larger K can also be considered as using more computation in each round. Similar to the results of the impact of E, simply increasing computation can only improve performance to a certain extent. It also shows the significance of MAS that increases performance with slightly more computation.



(a) Impact of E



(b) Impact of K

Figure 10: Analysis of the impact of (a) local epoch E and (b) the number of selected clients K on task set `sdnkt`. Larger E and K could reduce losses with more computation, but the benefit decreases as computation increases.

Method	K	Total Test Loss
All-in-one	8	0.618
MAS-2	8	0.512

Table 2: Comparison of total test loss using $K = 8$ selected clients on FL task set `sdnkt`. MAS achieves even better performance on $K = 8$.

The majority of experiments in this study are conducted with $K = 4$. We next analyze the impact of K in MAS with results of two splits on task set `sdnkt` in Table 2. The results indicate that MAS is also effective with $K = 8$, which outperforms $K = 4$ and all-in-one training.

5. Conclusions

In this work, we propose MAS, the first FL system to effectively coordinate and train multiple simultaneous FL tasks under resource constraints. In particular, we introduce task merging and task splitting to consider both synergies and differences among multiple FL tasks. Extensive empirical studies demonstrate that our method is effective in elevating performance and significantly reduce the training time and energy consumption by more than 40%. We believe that it is important to study training multiple simultaneous FL tasks and apply it in many real-world applications. We hope this research will inspire the community to further work on algorithm and system optimizations of training multiple simultaneous FL tasks. Future work involves designing better scheduling mechanisms to coordinate these tasks. Client selection strategies can also be considered to optimize resource and training allocation.

Acknowledgements This study is supported by RIE2020 Industry Alignment Fund-Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s) and Sony AI.

References

[1] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems, July 2020. arXiv:2007.03051.

[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[3] Hakan Bilen and Andrea Vedaldi. Integrated perception with recurrent multi-task neural networks. *Advances in neural information processing systems*, 29, 2016.

[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems*, 1:374–388, 2019.

[5] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[6] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pages 125–136, 2020.

[7] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, 2018.

[8] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[9] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and the 7th international joint conference on natural language processing (volume 2: short papers)*, pages 845–850, 2015.

[10] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[11] Ziqing Fan, Yanfeng Wang, Jiangchao Yao, Lingjuan Lyu, Ya Zhang, and Qi Tian. Fedskip: Combatting statistical heterogeneity with federated skip aggregation. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 131–140. IEEE, 2022.

[12] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[13] FLAIROP. Federated learning for robot picking, 2022.

[14] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *Advances in Neural Information Processing Systems*, 33:19586–19597, 2020.

[15] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, pages 3854–3863. PMLR, 2020.

[16] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[17] Cheol-Ho Hong, Ivor Spence, and Dimitrios S Nikolopoulos. Gpu virtualization and scheduling methods: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 50(3):1–37, 2017.

[18] Siyu Huang, Xi Li, Zhi-Qi Cheng, Zhongfei Zhang, and Alexander Hauptmann. Gnas: A greedy neural architecture search method for multi-attribute learning. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 2049–2057, 2018.

[19] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[20] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[22] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011.

[23] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[24] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[25] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[26] Abhishek Kumar and Hal Daumé. Learning task grouping and overlap in multi-task learning. In *Proceedings of the*

29th International Coference on International Conference on Machine Learning, page 1723–1730, 2012.

[27] Jingtao Li, Lingjuan Lyu, Daisuke Iso, Chaitali Chakrabarti, and Michael Spranger. Mocosfl: enabling cross-client collaborative self-supervised learning. In *The Eleventh International Conference on Learning Representations*, 2022.

[28] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2:429–450, 2020.

[29] Wenqi Li, Fausto Milletarì, Daguang Xu, Nicola Rieke, Jonny Hancox, Wentao Zhu, Maximilian Baust, Yan Cheng, Sébastien Ourselin, M Jorge Cardoso, et al. Privacy-preserving federated brain tumour segmentation. In *International Workshop on Machine Learning in Medical Imaging*, pages 133–141. Springer, 2019.

[30] Jie Liu, Jiawen Liu, Wan Du, and Dong Li. Performance analysis and characterization of training deep learning models on mobile device. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 506–515. IEEE, 2019.

[31] Ruixuan Liu, Fangzhao Wu, Chuhan Wu, Yanlin Wang, Lingjuan Lyu, Hong Chen, and Xing Xie. No one left behind: Inclusive federated learning over heterogeneous devices. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3398–3406, 2022.

[32] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5334–5343, 2017.

[33] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems*, 34, 2021.

[34] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[35] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[36] Vladimir Nekrasov, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7101–7107. IEEE, 2019.

[37] Xiaomin Ouyang, Zhiyuan Xie, Jiayu Zhou, Jianwei Huang, and Guoliang Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 54–66, 2021.

[38] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[39] Jason Posner, Lewis Tseng, Moayad Aloqaily, and Yaser Jararweh. Federated learning in vehicular networks: opportunities and solutions. *IEEE Network*, 35(2):152–159, 2021.

[40] Apple Machine Learning Research. A multi-task neural architecture for on-device scene analysis, 2022.

[41] Micah J Sheller, G Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In *International MICCAI Brainlesion Workshop*, pages 92–104. Springer, 2018.

[42] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

[43] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[44] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020.

[45] Yue Tan, Chen Chen, Weiming Zhuang, Xin Dong, Lingjuan Lyu, and Guodong Long. Taming heterogeneity to deal with test-time shift in federated learning. In *International Workshop on Federated Learning for Distributed Data Mining*, 2023.

[46] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, 8, 1995.

[47] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920*, 2019.

[48] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020.

[49] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020.

[50] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Yongfeng Huang, and Xing Xie. Communication-efficient federated learning via knowledge distillation. *Nature communications*, 13(1):2032, 2022.

[51] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *Proceedings of the Web Conference 2021*, pages 935–946, 2021.

[52] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[53] yuyang deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning, 2021.

[54] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018.

[55] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. End-to-end federated learning for autonomous driving vehicles. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[56] Jie Zhang, Chen Chen, Bo Li, Lingjuan Lyu, Shuang Wu, Shouhong Ding, Chunhua Shen, and Chao Wu. Dense: Data-free one-shot federated learning. *Advances in Neural Information Processing Systems*, 35:21414–21428, 2022.

[57] Jie Zhang, Chen Chen, Weiming Zhuang, and Lingjuan Lv. Addressing catastrophic forgetting in federated class-continual learning. *arXiv preprint arXiv:2303.06937*, 2023.

[58] Jie Zhang, Song Guo, Xiaosong Ma, Haozhao Wang, Wenchao Xu, and Feijie Wu. Parameterized knowledge transfer for personalized federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[59] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[60] Xiangyun Zhao, Haoxiang Li, Xiaohui Shen, Xiaodan Liang, and Ying Wu. A modulation module for multi-task learning with applications in image retrieval. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 401–416, 2018.

[61] Ligeng Zhu, Hongzhou Lin, Yao Lu, Yujun Lin, and Song Han. Delayed gradient averaging: Tolerate the communication latency for federated learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[62] Weiming Zhuang, Xin Gan, Yonggang Wen, and Shuai Zhang. Easyfl: A low-code federated learning platform for dummies. *IEEE Internet of Things Journal*, 2022.

[63] Weiming Zhuang, Xin Gan, Yonggang Wen, and Shuai Zhang. Optimizing performance of federated person re-identification: Benchmarking and analysis. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 2022.

[64] Weiming Zhuang and Lingjuan Lyu. Is normalization indispensable for multi-domain federated learning? *arXiv preprint arXiv:2306.05879*, 2023.

[65] Weiming Zhuang, Yonggang Wen, and Shuai Zhang. Divergence-aware federated self-supervised learning. In *International Conference on Learning Representations*, 2022.

[66] Weiming Zhuang, Yonggang Wen, Xuesen Zhang, Xin Gan, Daiying Yin, Dongzhan Zhou, Shuai Zhang, and Shuai Yi. Performance optimization of federated person re-identification via benchmark analysis. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 955–963, 2020.