Appendix

A Additional Description and Discussion for DeDES

A.1 Description of Model Filtering

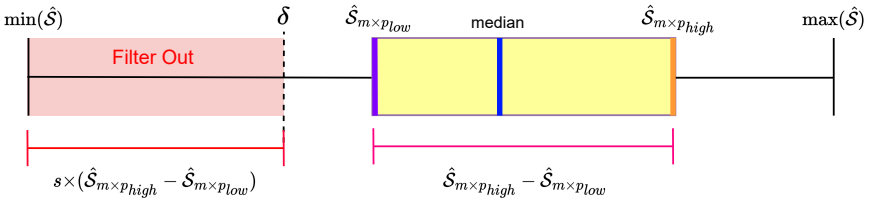

Figure 1: Illustration of the model filtering algorithm.

Fig. 1 is a sample illustration of our model filter algorithm, i.e., algorithm 2 in the main paper. This technique is essentially an innovative adaptation of the box-plot method, employed specifically for identifying outliers that possess scores beneath the threshold value δ.

Given the range parameters p_{low} and p_{high}, in conjunction with a pre-determined scaling factor s, the threshold δ can be computed using the formula $s \times (\hat{\mathcal{S}}_{m\times p_{high}} - \hat{\mathcal{S}}_{m\times p_{low}})$.

In the context of our experimental setup, we opted to set p_{low} at 0.25 and p_{high} at 0.75. Under these settings, $\hat{\mathcal{S}}_{m\times p_{low}}$ and $\hat{\mathcal{S}}_{m\times p_{high}}$ denote the first quartile (25th percentile) and third quartile (75th percentile), respectively, as typically depicted in a traditional box-plot diagram.

It should be emphasized that the primary aim of this procedure is to eliminate models exhibiting exceptionally low-performance scores, as demonstrated by their respective local validation accuracies in our experiments. As evident from Figure 1, any models scoring beneath the threshold δ are effectively filtered out, thereby excluding them from future clustering pursuits.

A.2 Description of Model Representation

Here we discuss the details about the model representation technique. Given a model structure such as Resnet-18, we can choose all or part of the model parameters to represent the model. E.g., as Fig. 2 of the main paper illustrates, we choose to represent models by the parameters of the last layer in every model within the set $\mathcal{M}$. These parameters are preferred as they hold a wealth of individualized information and adequately depict model behavior, particularly for the classifier, and data manifold/space in relation to local training.

If Resnet-18 is used as the model structure for a 10-class dataset, the last layer of its model parameters will be the layer with the name of *linear.weight* and *linear.bias*, which contains 512×10 and 10 parameters, respectively. Then, we will flatten these in total $512 \times 10 + 10 = 51210$ parameters into a one-dimensional vector with the size of 51210.

Following this, the model parameter matrix, denoted by $\mathcal{R}_{\mathcal{M}'}$, is the result of concatenating all these models' one-dimensional vectors and will have the size of $m' \times 51210$, where m' is the number of models in $\mathcal{M}'$. Prior to leveraging this matrix as the model representation for subsequent clustering, we will employ wide-ranging, commonly used preprocessing mechanisms. These may include MIN-MAX scaling, and/or dimensional reduction on $\mathcal{R}_{\mathcal{M}'}$. This optimizes the representation for better clustering results. Further details regarding the suitability and efficacy of diverse preprocessing techniques are elaborated upon in section C.2.

A.3 Discussion on Representative Model Selection

It is widely known that in machine learning and AI, a model's accuracy can often be directly related to the volume of data it has been trained on. In essence, the more diverse and abundant the data available for

Table 1: The impact of Representative Model Selection method for the EMNIST Balanced Dataset, VGG-5 (Spinal FC) structure when m=100, K = 50, where `DeDES_CV`, `DeDES_data` refers to our DeDES method that always selects the model with the highest score/ model with the most number of local data within a cluster, respectively. `DeDES_mixed` is our mixed selection strategy used in the experiments.

Method	`DeDES_mixed`	`DeDES_CV`	`DeDES_data`
homo	**85.19**	84.87	84.81
iid-dq	**87.34**	83.92	**87.34**
noniid-lds	**83.43**	83.1	83.29
noniid-l18	**85.43**	85.14	85.33

training, the higher is the potential for the model to comprehend and discern the intricacies of the underlying data. If a client only has a small amount of data, then the trained model might struggle to identify significant patterns or it might overfit to that specific data, meaning it could perform well on that data but poorly on new, unseen data. Inversely, models trained on large volumes of data have a broader foundation to learn from and are typically better at generalizing their findings to new data, thereby often yielding higher accuracy.

We can also validate this conclusion from our experiment results, as depicted in Fig. 4 and Table 1 of the main paper, it is clear that Data Selection (DS) consistently outperforms other methods in most datasets when it comes to the iid-dq partition. This is particularly true when one or a few parties have significantly larger pools of data compared to other clients. The single TOP 1/2 models as in Fig. 4 (b) have the the largest dataset with samples of every class in the label set, which leads to their strong generalization ability under this partition. However, when data distribution is more uniform or balanced, referred to as the homogenous partition, the Data Selection method doesn't necessarily offer the best model selection strategy. It's simple to comprehend that in scenarios where data is uniformly distributed among different parties, selecting models purely based on the volume of data becomes ineffective. This is due to the lack of a significant data volume disparity between the client with the highest data volume and the rest. Therefore, a modification to the selection strategy becomes necessary, focusing more on models with the highest validation accuracy rather than just data volume.

We refined a strategy to choose the representative model. If the models within a cluster equivalently distribute the training data, our strategy selects the model with the highest local validation accuracy. Conversely, if the distribution is skewed, the model with the maximum training data is chosen. This strategy was enforced by calculating the ratio of the quantity of local training data for the median model in a cluster to that of the model with the most local training data. If this ratio is less than the prescribed selection threshold τ, the cluster is deemed unbalanced. To substantiate the soundness of our representative model selection technique, we performed an ablation study on three different methodologies:

1. Mixed: Both data distribution and model validation accuracies were taken into account in this model selection method.

2. CV: This method consistently chose the model with the highest local validation accuracy.

3. Data: This method invariably selected the model with the most local data within a cluster.

The data in Table 1 reveals that the mixed model selection method exceeds other methods across all four data partitions. While the disparity among the selection strategies is not significantly vast (since the representative model selection is just an aspect of the whole model selection algorithm), it still displays superior performance. Not that for the iid-dq data partition, the accuracy of the mixed selection strategy matches that of the data selection strategy. This is a testament to the fact that given a significantly unbalanced dataset, our mixed strategy will invariably choose the model with the most data, mirroring the data selection strategy.

A.4 Description of Adaptation to Heterogeneous models

In this section, we will provide more details about how to apply `DeDES` to heterogeneous models. As mentioned in the main paper, our method transforms these models into a unified model parameter matrix $\mathcal{R}_{\mathcal{M}'}$.

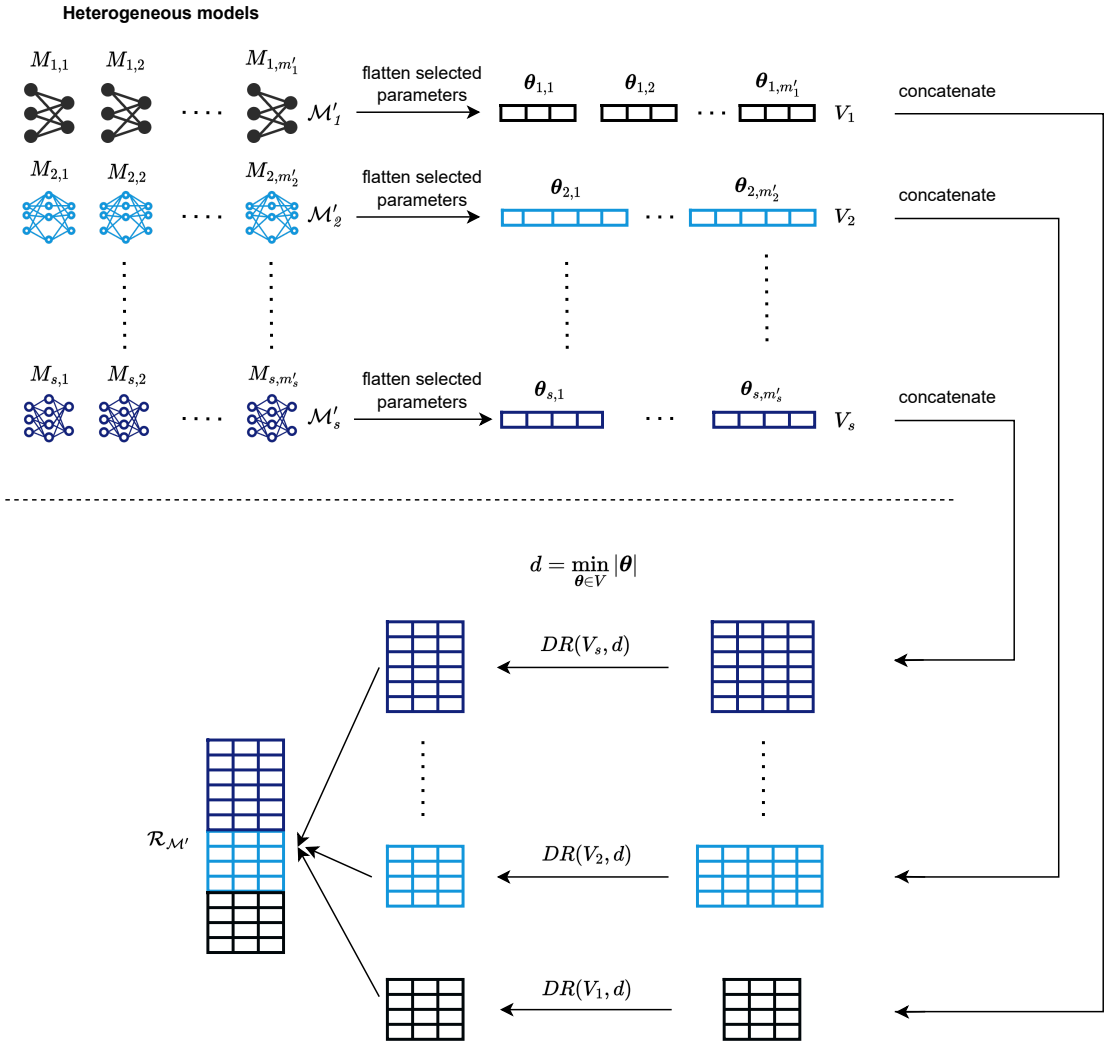

Figure 2: Process diagram for handling heterogeneous models

Fig. 2 shows the whole handling process, as described as follows:

Assuming that there are s types of model structures and each contains any number of models, the model set $\mathcal{M}'$ for all those models is,

$$\mathcal{M}' = \{M_1, M_2, \ldots, M_{m'}\} \overset{\text{group by } s}{=\!=\!=} \bigcup_{i=1}^{s} \mathcal{M}'_i \text{ with } \mathcal{M}'_i = \{M_{i,1}, \ldots, M_{1,m'_i}\},$$

where m'_i denotes the number of models in $\mathcal{M}'_i$ that share the i-th model structure; and the total number of models in $\mathcal{M}'$ is $m' = \sum_{i=1}^{s} m'_i$.

We extract part of the model parameters and flatten them to be a one-dimensional vector as the representation of the model. For the model $M_i \in \mathcal{M}'$, let $\boldsymbol{\theta}_i$ denote the flattened selected parameters (e.g., the parameters of the last layer) of M_i. Let $V_i = \{\boldsymbol{\theta}_{i,1}, \ldots, \boldsymbol{\theta}_{i,m'_i}\}$ be the vector set for the model subset $\mathcal{M}'_i$. Thus, the vector set V for all models $\mathcal{M}'$ will be $V = \bigcup_{i=1}^{s} V_i = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \ldots, \boldsymbol{\theta}_{m'}\}$.

Since the model structures vary for different $\mathcal{M}_i$, the sizes of the vectors in V are also different, which makes it hard to directly apply the clustering methods. To solve such an issue, we reduce the dimension of all

vectors to a unified dimension d. Given a Dimension Reduction (DR) algorithm, we transform the size of all vectors $\boldsymbol{\theta}_i$ of V into $d = \min_{\boldsymbol{\theta} \in V} |\boldsymbol{\theta}|$ by applying DR to each sub-vector set V_i, separately. The *model parameter matrix* $\mathcal{R}_{\mathcal{M}'}$ is obtained by combining all these matrices together,

$$\mathcal{R}_{\mathcal{M}'} = [DR(V_1, d), DR(V_2, d), \cdots, DR(V_s, d)]^T. \tag{1}$$

$\mathcal{R}_{\mathcal{M}'} \in \mathbb{R}^{m' \times d}$ with each sub-matrix $DR(V_i, d) \in \mathbb{R}^{m'_i \times d}$. We can then perform further clustering on $\mathcal{R}_{\mathcal{M}'}$, since all heterogeneous models are transformed to the same dimensions.

For some dimension reduction algorithms such as PCA, it is required to set the maximum dimensionality that can be reduced to the minimum value between the two dimensions of the original matrix. In this case, we set the dimensionality d to $\min(\min_{\boldsymbol{\theta} \in V} |\boldsymbol{\theta}|, \min_{1 \leq i \leq s} m'_i)$.

Here we show an example:

Assuming that three distinct model structures are given ($s = 3$), each structure comprises 600, 700, and 800 models, respectively. We will have in total $m' = 600 + 700 + 800 = 2100$ models. The last layer is selected to serve as the model's representation, with the representation sizes for the three model structures being 300, 400, and 500, respectively. Then, we can obtain three vector sets: $V_1 = \{\boldsymbol{\theta}_{1,1}, \boldsymbol{\theta}_{1,2}, \ldots, \boldsymbol{\theta}_{1,600}\}$, $V_2 = \{\boldsymbol{\theta}_{2,1}, \boldsymbol{\theta}_{2,2}, \ldots, \boldsymbol{\theta}_{2,700}\}$, $V_3 = \{\boldsymbol{\theta}_{3,1}, \boldsymbol{\theta}_{3,2}, \ldots, \boldsymbol{\theta}_{3,800}\}$, and the lengths of the vectors within the V_1, V_2, and V_3 sets are respectively 300, 400, and 500. Then the vector set V for all models $\mathcal{M}'$ will be:

$$V = \bigcup_{i=1}^{3} V_i = \{\overbrace{\boldsymbol{\theta}_{1,1}, \boldsymbol{\theta}_{1,2}, \ldots, \boldsymbol{\theta}_{1,600}}^{V_1}, \underbrace{\boldsymbol{\theta}_{2,1}, \boldsymbol{\theta}_{2,2}, \ldots, \boldsymbol{\theta}_{2,700}}_{V_2}, \overbrace{\boldsymbol{\theta}_{3,1}, \boldsymbol{\theta}_{3,2}, \ldots, \boldsymbol{\theta}_{3,800}}^{V_3}\}. \tag{2}$$

Then, the unified dimension d will be:

$$d = \min_{\boldsymbol{\theta} \in V} |\boldsymbol{\theta}| = \min\{\overbrace{300, 300, \ldots, 300}^{|V_1|=600}, \underbrace{400, 400, \ldots, 400}_{|V_2|=700}, \overbrace{500, 500, \ldots, 500}^{|V_3|=800}\} = 800 \tag{3}$$

Thus, we can get the model parameter matrix $\mathcal{R}_{\mathcal{M}'}$ by:

$$\mathcal{R}_{\mathcal{M}'} = [DR(V_1, 300), DR(V_2, 300), \ldots, DR(V_s, 300)]^T \tag{4}$$

with $\mathcal{R}_{\mathcal{M}'}$ size of 2100×300. And this model parameter matrix can be then used by `DeDES` to conduct further clustering.

A.5 Discussion on Privacy Analysis

We recognize the potential privacy leakages and are dedicated to mitigating risks in our proposed method. The generator of the GAN is prevented from directly accessing the real data, and our one-shot setting with only a single communication round helps to minimize the exposure of sensitive information. Prior to model submission to the server, we can employ various mechanisms (Kolluri et al., 2021; Liu et al., 2021) to reinforce model privacy against GAN attacks (Hitaj et al., 2017), ensuring enhanced security and data confidentiality.

To thwart model inference attacks, which aim to recover training data from the models, and membership inference attacks that can determine if a particular record was part of the model's training dataset or not, we can choose to incorporate several strategic responses. For instance, the application of differential privacy (Xiong et al., 2021) introduces random noise to the model's outputs, obscuring the model's predictions. Additionally, including the regularization terms (Niknam et al., 2020), such as L1 and L2 regularization, to the model's loss function is also a way to discourage the model from overly fitting the training data, thus safeguarding the training data's privacy.

Table 2: Physical running time (second) for different model selection methods, KMeans is used as the clustering method for DeDES.

Dataset	Model	Partition	m	K	DeDES	AS
EMNIST Letters	VGG-5 (Spinal FC)	noniid-l8	400	200	33.73	2.12×10^{-5}
FEMNIST	Resnet-18	noniid-lds	3597	100	70.2	2.13×10^{-5}

In terms of model extraction attacks, where the adversary attempts to replicate the targeted machine learning model without direct access to its parameters or training data, a similar approach of differential privacy (Yan et al., 2021) and regularization (Li et al., 2023b) could be applied during model training. Furthermore, we can also try to integrate techniques such as watermarking (Tekgul et al., 2021), which embeds identifiable information into the model, to detect any unauthorized replication attempts. Other defense policies can also be applied to defend against attack, such as the defense scheme based on the physical unclonable function (PUF) (Li et al., 2023a), etc.

In summation, by incorporating these diverse measures into our model selection method, we aim to strengthen the protections for model privacy and enhance the security of the model ensemble team.

A.6 Discussion on Computational/Time Complexity

In this section, we explore the circumstances wherein a model selection approach such as DeDES is more efficient than the All Selection (AS) method. As we know, for the ensemble selection problem in ensemble learning, the time cost will have two parts: model selection and ensemble learning.

1. The time complexity involved in selecting models from the complete set $\mathcal{M}$ is denoted as $O(selection)$. As illustrated in Lemma 1 of the primary paper, DeDES's overall time complexity is defined as $O(m \log m + md + m) + O(clustering)$, where $O(clustering)$ represents the time complexity of the applied model clustering methodology. When KMeans is utilized in conjunction with DeDES, the total time complexity equals $O(m \log m + md + mKd + m)$. On the other hand, the AS method's time complexity is O(1), as it entails selecting all models for the ensemble team, with no additional calculations required.

2. The time complexity for ensemble learning (weighted voting) determined by the ensemble team $\mathcal{M}_K^*$ is denoted as $O(ensemble)$. For the purposes of our hypothesization, we'll assume that all model structures within $\mathcal{M}_K^*$ are identical. Consequently, given this homogeneity in structure and runtime environment, the inference time for a single sample is consistent across all models, denoted as $O(C)$. Assuming that we possess n_t samples suitable for testing by $\mathcal{M}_K^*$, DeDES's total ensemble learning time complexity will be: $O(KnC)$. Given that the weighted averaging time $O(1)$ is relatively negligible, the AS method's ensemble learning time complexity will be: $O(mnC)$.

To summarize, the overall time complexity for DeDES and AS will be:

1. $O(\texttt{DeDES}) = O(selection) + O(ensemble) = O(m \log m + md + m) + O(clustering) + O(Kn_tC)$.

2. $O(AS) = O(selection) + O(ensemble) = O(1) + O(mn_tC) \approx O(mn_tC)$.

In order for DeDES to be considered more efficient than AS, the time complexity of AS would need to be greater than that of DeDES. In other words, $O(mn_tC) > O(m \log m + md + m) + O(clustering) + O(Kn_tC)$.

Assuming the clustering method used is KMeans, the inequality can be further refined as: $O(mn_tC) > O(m \log m + md + mKd + m + Kn_tC)$.

I.e., for a chosen K, it requires the number of test samples, n_t, to be greater than $\frac{m \log m + md + mKd + m}{(m-K)C}$. This condition is feasibly attainable in practical situations, due to the frequent necessity to test a significant number of samples. Consequently, n_t can effortlessly transcend a nominal value.

To defend our argument, consider the following example. Table. 2 and Table. 3 display the physical running time comparison for various model selection methods and the inference time for a single test sample on two different datasets, respectively. These were tested on our Linux server, the details of which are provided in

Table 3: Physical running time (second) for the inference of one sample.

Dataset	Model	Partition	Inference Time
EMNIST Letters	VGG-5 (Spinal FC)	noniid-l8	2.2216×10^{-4}
FEMNIST	Resnet-18	noniid-lds	1.7291×10^{-4}

Table 4: Physical running time (second) comparison of `DeDES` and AS for the whole ensemble selection process, including model selection and inference of ensemble learning where KMeans is used as the clustering method for `DeDES` on $n_t = 100,000$ test samples.

Dataset	Model	Partition	m	K	`DeDES`	AS
EMNIST Letters	VGG-5 (Spinal FC)	noniid-l8	400	200	4479.93	8886.40
FEMNIST	Resnet-18	noniid-lds	3597	100	1799.3	62195.73

section C.4. Upon inspection, it's evident that our `DeDES` method takes more time in model selection than the AS method. It's essential, however, to note that this model selection process will be carried out **only once**. Thus, despite the seemingly lengthier process, the overall time cost is manageable given that we only have to wait for approximately 1 minute to obtain the selection results for both datasets.

Table 3 shows that processing a test sample through the VGG-5 (Spinal FC) model takes 2.2216×10^{-4} second, denoted as $C = 2.2216 \times 10^{-4}$. In the case of the EMNIST Letters dataset, we selected $K = 200$ from a pool of $m = 400$ models for our ensemble. Therefore, gauging by our earlier assertion, `DeDES` can be considered more efficient than the AS method when: $Time(All\ Selection) + Time(AS\ Ensemble) > Time(\texttt{DeDES}) + Time(\texttt{DeDES}\ Ensemble)$, i.e,

$$\underbrace{2.12 \times 10^{-5}}_{All\ Selection} + n_t \times \underbrace{400}_{m} \times \underbrace{2.2216 \times 10^{-4}}_{C} > \overbrace{36.73}^{\texttt{DeDES}} + n_t \times \overbrace{200}^{K} \times \overbrace{2.2216 \times 10^{-4}}^{C} \tag{5}$$

Consequently, we get that $n_t > 826.66$, suggesting that when we have over 827 test samples at our disposal to carry out ensemble learning, the time efficiency of the `DeDES` surpasses that of the All Selection method. This implication becomes particularly encouraging when dealing with the FEMNIST dataset with $K = 100$ among $m = 3597$ models, whereby n_t merely requires to exceed 116.09 for us to reap both superior performance (performance results can be seen from Table 1. in the main paper) and efficiency through the adoption of the `DeDES` method in lieu of AS.

In practice, we often encounter thousands of test samples in need of inference from the ensemble team. Hence, it is no doubt that applying a model selection method is beneficial for ensemble learning. Table. 4 presents the comprehensive running time for both methods when we have $n_t = 100{,}000$ test samples available for voting. The clear observation is that the AS method consumes considerable time due to its model testing and execution of ensemble learning, however, our `DeDES` provides tangible savings of approximately 49.59% of the time than AS for the EMNIST Letters dataset, and a remarkable **97.1% of the time** than AS for the FEMNIST dataset with no compromise on performance. This empirically attests the superior efficiency and effectiveness of our proposed method.

Computation Complexity and Communication Cost: Since our approach adheres to the one-shot federated learning scheme, which inherently requires every client to communicate with the server **only once** - when they transmit their respective models. Consequently, the sole communication cost incurred by every client is the transmission of its meticulously trained model to the central server. To illustrate, a client, such as a smartphone, may train a Resnet model with a 44-MB size, and dispatch it to our server via the internet. This task is not a challenge for modern devices. We would like to emphasize that our usual protocol does not impose any fixed constraints on the computational resources designated for the central server. This is based on our fundamental belief that the central server possesses an exceptional computational capability. Hence, in discussing communication costs and computational resources, our method exhibits flexibility and

Table 5: Details of experiment configurations. Note that the **bold** model structure is the default structure used in our experiment for the corresponding dataset.

Dataset	C	Size ($\sum_i n_i$)	k in noniid-lk	Model	m
EMNIST Digits	10	280,000	3	**VGG-5 (Spinal FC)**, ResNet-50	100, 200, 400
EMNIST Letters	26	145,600	8		
EMNIST Balanced	47	131,600	18		
SVHN	10	99,289	3	**ResNet-18**	200
FEMNIST	62	805,263	-	**ResNet-18**	**3597**
CIFAR10	10	60,000	4	**ResNet-50**, DenseNet-121	50, 100, 200
CIFAR100	100	60,000	45	**ResNet-50**, Deep Layer Aggregation	5, 10, 20

scalability. It can be comfortably expanded to accommodate a large-scale federated learning system without compromising efficiency.

B Justification on `DeDES`

Here we provide a justification for why `DeDES` is a good approximation for solving Problem 1.

Theorem 1 presented by Luis A Ortega et al. (Ortega et al., 2022) states that the expected loss of an ensemble *decreases* as the diversity measure among model ensembles *increases*. Theorem 3, also presented by Ortega et al. (Ortega et al., 2022), bolsters this evidence, showcasing that the diversity measure proliferates with a decrease in correlation among ensembles.

To sum up, a lower correlation among model ensembles encourages lower expected loss. This rationalization is reinforced within algorithm 1 in the main paper, which within Lines 5 through 13, clusters models and subsequently selects one from each cluster, ensuring that the models in the generated model ensemble are highly uncorrelated among each other thus strengthening the diversity of the ensemble team, and setting the stage for an optimal approximation to Problem 1.

Notably, Theorem 1 and 3 are primarily designed for a scenario where data is distributed independently and identically (i.i.d.). Our empirical findings, however, suggest that they perform impressively even in a non-i.i.d data setting, provided that the ensemble group is carefully selected for high diversity. Consequently, the relationship between the diversity measure and the expected loss in a non-i.i.d. environment presents a potentially profound area of future theoretical exploration that is worthy of further investigation.

C Experiment Setup

C.1 Dataset and Model Structure Configurations

As shown in Table. 5, we run our method on 7 datasets with 5 different model structures to validate the effectiveness of our method. The EMNIST dataset has three different types of split: Digits, Letters, and Balanced which have 10, 26, and 47 classes respectively.

The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset, they are two widely used datasets for machine learning and computer vision research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes, making a total of 6,000 images per class. The CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each.

The Street View House Numbers (SVHN) dataset is a public dataset that contains real-world, full-color images of house numbers extracted from Google Street View images. We use this dataset to validate that our method is generalizable and can be applied to real-world applications.

Meanwhile, to validate that our method can be applied to large-scale federated learning systems, we use the FEMNIST dataset, which is a benchmark dataset that contains in total 3597 clients, whose data distribution is non-i.i.d among these clients.

All model structures we used are CNN models, such as Resnet-18 which is a popular convolutional neural network model in the realm of deep learning. We use 5 different model structures to show that our method is generalizable for different model structures.

C.2 Component Configurations

In this subsection, we describe the default configurations of our `DeDES` framework for the experiments in the main paper.

For local model training, we utilize the *SGD* optimizer to get 200 models for every party through 200 epochs of training with learning rate started at 0.1 and decrease at later epochs, i.e., we save all the models from the 200 training rounds. After the training is finished, for every party, we select the model with the highest local validation accuracy (among these 200 models) as the final well-trained model and then upload it to the server of model market. Meanwhile, we record the test results on the whole test set for this final model.

In our experiments, we use the local validations of the parties as our model score $\mathcal{S}$, which are uploaded from the clients; we select parameters of the final model layer (last layer) as the model representation; we utilize the MINMAX scaler to preprocess the *Model Representation Matrix*, and the *Gaussian Normalization* scaler to preprocess the *Label Distribution* ground-truth input data; we do not utilize any dimension reduction strategy for the model representation matrix because the last layer of model parameters are already few in number.

Hierarchical clustering is used on the *homo* and *iid-dq* partitioned dataset; *KMeans* clustering is used on the *noniid-lds* and *noniid-lk* partitioned dataset.

Our proposed model representative selection approach is utilized to determine the representative model within each cluster; as we have stated, we apply the weighted voting strategy based on the size of local clients' datasets to perform ensemble learning; we use the test accuracy on the whole test set D^{test} as the evaluation metric for all the ensemble selection methods.

C.3 Method for Tuning Hyperparameters

As described in the main paper, we have five hyperparameters to tune: K, τ, p_{low}, p_{high}, and s. How to select these hyperparameters is an important problem. First, we discuss how to select K, K is the number of models we want to select to form the ensemble team. In practice, it is the final users's decision to determine K, depending on their requirements. For example, after evaluating their computational resources, a user may decide to establish an ensemble by selecting 40 models from a pool of 100. In classical ensemble selection literature, this K will be pre-defined by the author (Fan et al., 2002). In section 5.4 of our main paper, we studied how K will affect the performance of ensemble learning and we picked K around half of the total number of models m in our experiments to show the good performance of our method.

For other hyperparameters, like τ that is used to decide how to select the model within a cluster, and p_{low}, p_{high}, s are used to decide how many models to be filtered out at the first step of `DeDES`, we adopted a variant of the widely used Grid Search (Nagaraj et al., 2022) method to tune them. In the grid search approach, a method is tested for every combination of specified hyperparameters and the best-performed combination on the validation set will be picked as the final hyperparameters. First, we show the search ranges for these parameters, then, we will illustrate how to design a good grid search method for `DeDES` under FL setting.

The search ranges for different hyperparameters are:

- τ: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], and 0.9 is chosen in our experiments. The logic behind this is as follows: if the model with median local training data in a cluster has less than 90 % the data of the model with most local training data, the cluster is identified as unbalanced. In this case, the model with the most training data is selected. If not, the model with the highest score is chosen.

- p_{low}: [0.05, 0.10, 0.15, 0.25, 0.30, 0.35]. We finally choose p_{low} to be the 0.25 (25-percentile), just the same as the classical box-plot method since it is widely proven effective.

- p_{high}: [0.6, 0.65, 0.70, 0.75, 0.80, 0.85]. We finally choose p_{high} to be the 0.75 (75-percentile), just the same as the classical box-plot method.

- s: [0, 0.5, 1.0, 2.0, 3.0], and 1.0 is ultimately selected. How many models will be filtered out before conducting further clustering will depend on the model score distribution of the clients.

Then we discuss how to apply grid search for `DeDES`. Based on the federated learning setting, the central server cannot get the validation set from local clients, so we cannot apply grid search to `DeDES` directly. However, we can still get the data distribution statistics from these clients, i.e., we know the quantity of training/validation data each individual client possesses. With this information, we developed a customized version of grid search specifically designed to fine-tune the hyperparameters within a federated learning framework. This is attained by dispatching the chosen model parameters back to their originating clients and computing a weighted average of their local ensemble validation accuracies. Consequently, even the constraints of a federated learning environment cannot inhibit our method's proficiency and adaptability.

Let's walk through the fine-tuning of hyperparameters using `DeDES`:

Consider a scenario where we need to select $K = 3$ models from $m = 100$ clients. The central model server will collect 100 models M_1, M_2, $\cdots$, M_{100}, each corresponding to a client, from 1 to 100. We employ a systematic grid search strategy to identify the optimal hyperparameter combinations out of a set of C=20 possibilities. For each i-th hyperparameter combination, ranging from 1 to 20, we accomplish the following:

1. Implement `DeDES` on the central server to select the best $K = 3$ models in reference to the i-th hyperparameter combination. In this context, let's assume our selection led us to M_5, M_{36}, and M_{74}.

2. Transmit these 3 models back to their corresponding clients, omitting their own model. Thus, client 5 will receive M_{36}, M_{74}; and likewise, client 36 will get M_5, M_{74} and client 74 will acquire M_5, M_{36}.

3. Client 5, 36, and 74 will then execute ensemble learning (through voting) based on the selected ensemble team $\mathcal{M}_3^* = \{M_5, M_{36}, M_{74}\}$, using their respective local validation sets $D_5^{val}, D_{36}^{val}, D_{74}^{val}$. Following this, they will send their local ensemble validation accuracies Acc_5^{val}, Acc_{36}^{val} and Acc_{74}^{val} back to the central server.

4. The central model server will then derive the final validation accuracy Acc_i^{hyper} for the i-th hyperparameter combination by implementing a weighted average calculation:

$$Acc_i^{hyper} = \sum_{j=1}^{K=3} \frac{n_j^{val} \times Acc_j^{val}}{\sum_{k=1}^{K} n_k^{val}} = \frac{n_5^{val} \times Acc_5^{val} + n_{36}^{val} \times Acc_{36}^{val} + n_{74}^{val} \times Acc_{74}^{val}}{n_5^{val} + n_{36}^{val} + n_{74}^{val}} \tag{6}$$

Repeat the above process until we get all test all validation accuracies Acc_i^{hyper} for i from 1 to 20. Then, we will identify and select the combination of hyperparameters that yields the highest validation accuracy:

$$index = \underset{i \in \{1, \cdots, 20\}}{\arg\max} \; Acc_i^{hyper} \tag{7}$$

We accordingly select the $index$-th hyperparameter combination as our optimal set of hyperparameters.

Table 6: Test accuracy (%) comparison for different dataset on different data partitions and model structures. The best and next best methods are **bolded** and underlined, respectively. If our `DeDES` method is better than the *LD* ground-truth method, the value of *LD* method will be marked in skyblue. The FedAvg and MeanAvg algorithms cannot be utilized on heterogeneous models. Thus, they did not yield results under heterogeneous settings such as EMNIST digits (50% VGG-5 Spinal FC and 50% Resnet-50).

Dataset	Partition	m	K	DeDES	CV	DS	RS	FedAvg	MeanAvg	AS	LD	Oracle
EMNIST Digits (100% Resnet-50)	homo	400	150	96.33	**96.65**	96.23	96.31	10.25	10.22	96.46	96.70	99.71
	iid-dq	400	150	**98.13**	98.08	98.07	97.94	10.63	10.64	98.01	98.13	99.70
	noniid-lds	400	150	**96.52**	86.58	95.48	96.04	10.24	10.19	96.50	94.80	99.70
	noniid-l3	400	150	**96.64**	89.21	58.59	94.72	9.74	9.61	96.81	96.83	99.67
EMNIST Letters (100% VGG-5 Spinal FC)	homo	200	120	88.64	**88.88**	88.82	88.68	3.72	3.71	88.77	88.77	95.12
	iid-dq	200	120	92.32	91.97	**92.33**	92.13	3.84	3.82	92.19	92.33	95.12
	noniid-lds	200	120	**87.93**	86.52	83.45	82.25	4.03	4.02	87.74	85.01	94.90
	noniid-l8	200	120	**89.10**	84.40	86.98	85.95	3.85	3.84	87.93	87.54	95.06
EMNIST Letters (100% Resnet-50)	homo	200	120	78.01	**78.77**	77.13	77.08	3.86	3.89	77.91	77.41	94.76
	iid-dq	200	120	88.88	88.88	**88.89**	88.45	3.82	3.80	88.88	88.85	95.13
	noniid-lds	200	120	**79.78**	79.53	77.27	78.65	4.23	4.28	80.55	78.37	94.86
	noniid-l8	200	120	**81.10**	80.54	78.86	80.28	3.74	3.69	82.79	82.33	95.08
EMNIST Balanced (100% Resnet-50)	homo	100	50	80.12	**80.33**	79.38	79.20	2.15	2.18	80.11	78.93	89.44
	iid-dq	100	50	85.68	85.68	**85.71**	84.54	2.11	2.14	85.68	85.76	89.12
	noniid-lds	100	50	**76.34**	71.64	74.16	75.00	2.23	2.21	77.56	70.75	89.52
	noniid-l18	100	50	**77.99**	77.71	77.98	77.40	2.13	2.13	80.28	78.58	89.39
CIFAR10 (100% Densenet)	homo	200	100	**46.84**	46.47	45.37	45.68	10.49	10.08	46.30	46.34	90.57
	iid-dq	200	100	52.38	**53.55**	53.47	51.76	10.45	10.56	53.01	54.03	90.38
	noniid-lds	200	100	**44.90**	40.89	40.54	41.49	10.38	10.52	43.91	41.61	91.01
	noniid-l4	200	100	**47.04**	46.08	40.92	45.43	9.71	9.47	48.51	47.45	90.79
CIFAR100 (100% Deep Layer Aggregation)	homo	20	12	**23.01**	22.47	22.27	22.63	0.95	0.94	24.80	22.12	52.63
	iid-dq	20	12	**39.18**	**39.18**	**39.18**	37.04	1.01	0.95	39.18	39.18	55.61
	noniid-lds	20	12	**22.29**	21.37	21.88	21.15	0.94	0.94	25.11	21.42	55.94
	noniid-l45	20	12	**24.41**	24.07	23.08	23.59	0.87	0.85	27.42	23.19	54.90
EMNIST Digits (50% VGG-5 Spinal FC, 50% Resnet-50)	homo	400	250	97.46	**98.04**	97.41	97.10	-	-	97.93	97.67	99.74
	iid-dq	400	250	98.63	98.61	**98.64**	97.58	-	-	98.61	98.89	99.71
	noniid-lds	400	250	**97.31**	94.51	94.25	91.21	-	-	97.25	95.86	99.72
	noniid-l3	400	250	**97.64**	96.47	88.16	96.38	-	-	97.65	97.55	99.61
EMNIST Letters (50% VGG-5 Spinal FC, 50% Resnet-50)	homo	400	200	81.90	**83.49**	54.66	79.82	-	-	83.05	82.58	95.12
	iid-dq	400	200	**90.00**	89.38	89.50	88.39	-	-	89.04	88.89	95.12
	noniid-lds	400	200	**82.91**	82.87	81.27	81.78	-	-	82.97	82.9	94.9
	noniid-l8	400	200	80.69	**81.19**	64.56	65.58	-	-	82.90	81.58	95.06

C.4 Environment

All our experiments are running on a single machine with 1TB RAM and 256 cores AMD EPYC 7742 64-Core Processor @ 3.4GHz CPU. The GPU we used is NVIDIA A100 SXM4 with 40GB memory. The environment settings are: Python 3.9.12, PyTorch 1.12.1 with CUDA 11.6 on Ubuntu 20.04.4 LTS.

All the experimental results are the average over three trials.

D Additional Experiments

In this section, we will show more experimental results as a supplement to the main paper. Due to the relatively large amount of experimental data, for each conclusion/finding, we take one of the experimental cases as presentation (one dataset, one m and one K), while the conclusion remains similar to other datasets/m/K.

D.1 Performance Analysis

Table 6 shows the performance of different methods when we apply them on the 5 datasets with 4 partitions, but different model structures than the main paper. K is selected about half of m. Note that when

Method	Rank	Accuracy (%)
DeDES	**214/1024**	**98.34**
AS	372/1024	97.09
DS	608/1024	89.63
LD	675/1024	87.86
CV	933/1024	74.73
RS	952/1024	72.45

Table 7: Complete inspection on ensemble teams for *EMNIST Digits* dataset with $m = 10, K = 5$, *noniid-lds* partition.

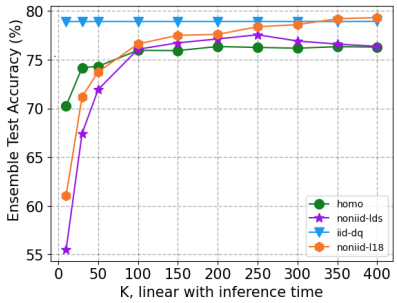

Figure 3: The relationship of K and Ensemble Test Accuracy of DeDES for the *EMNIST Balanced* Dataset when m=400.

Table 8: Test accuracy comparison among model ensemble, DENSE and FedAvg for the *CIFAR10* Dataset, Resnet-50 structure when m=5.

Method	Model Ensemble	DENSE	FedAvg
homo	**62.15**	58.84	15.93
iid-dq	**53.33**	13.25	13.42
noniid-lds	**46.63**	44.27	15.99
noniid-l4	**33.63**	31.66	19.9

$m = 200, K = 120$ on the EMNIST Letters dataset when the model structure is Resnet-50, the test accuracy of DeDES and AS, CV are the same, which means they both selected the same ensemble team. Compared to other baseline methods, DeDES can still achieve good performance (at least the second best, close to the best method of All Selection, which is very time-consuming).

Under the heterogeneous models setting, DeDES is still able to achieve a relatively better performance, though it may not perform as well as the homogeneous one which results from the significant variation in the structure of the models and making models of the same type to the same group when performing clustering. Therefore, to ensure that the clustering algorithm can find diverse models in the same structure type, a relatively large value of K is required.

Table 7 enumerated the accuracy of all 1024 teams and the ranking of ensemble teams selected by different methods for another data partition (*noniid-lds*) of the EMNIST balanced dataset. We can see that the ensemble team selected by DeDES is ranked higher than other baseline methods, which validates the efficacy of our method. Note the value of K here is 5 while the size of the best ensemble team among all 1024 teams is 4, therefore, how to select an appropriate K remains an open problem.

D.2 Impact on Efficiency

Fig. 3 gives another plot of the relationship between K and test accuracy for the EMNIST Balanced dataset. The conclusion remains the same as the main paper that we don't have to select all models to form an ensemble team for most of the cases, which saves the inference time and also keeps good performance.

D.3 Compare DENSE with ensemble

To show the necessity of model ensemble rather than knowledge distillation (KD), we give another comparison between DENSE and model ensemble with $m = 5$ for CIFAR10 dataset, Resnet-50 model structure on Table 8. As we can see, no matter under what data partitions, it is discernible that the performance of the model ensemble consistently surpasses that of DENSE, which employs KD, even with a minimal party size of just 5 clients. Thus, to achieve desirable performance under the one-shot federated learning setup, it becomes essential to leverage model ensemble.

Table 9: Performance comparison of choosing different hyperparameter combinations for the SVHN dataset, Resnet-18 structure when $m = 200$, $K = 80$. Here we provide 7 hyperparameter combinations to show the effects of these hyperparameters.

ID	τ	p_{low}	p_{high}	s	homo	iid-dq	noniid-lds	noniid-l3
1	0.9	0.25	0.75	1	**38.90**	**65.83**	**32.56**	**31.45**
2	0.9	0.25	0.75	0	**38.90**	65.46	32.21	30.92
3	0.9	0.25	0.75	2	**38.90**	65.82	32.50	31.45
4	0.1	0.25	0.75	1	**38.90**	65.03	32.41	31.07
5	0.5	0.25	0.75	1	**38.90**	65.35	32.51	31.11
6	0.9	0.35	0.85	1	**38.90**	65.82	32.54	31.45
7	0.9	0.05	0.6	1	**38.90**	65.81	32.53	31.44

E Additional Ablation Studies

In this section, we will give more ablation studies about the effect of each component of DeDES including model representation, dimension reduction, etc.

E.1 Comparison with different hyperparameter combinations

Table 9 presents the comparative analysis of seven distinct hyperparameter combinations. Initially, one can observe that the fluctuation in hyperparameters does not bring a substantial change in performance, thereby solidifying the stability of our `DeDES` approach. Simultaneously, Table 9 reaffirms that regardless of the hyperparameters selected, the homo data partition will consistently deliver identical performance. This consistency can be attributed to the homo data partition generating models with nearly the same model parameters, thereby ensuring that no matter how many models we filter out or what model will we choose within a cluster, the eventual ensemble performance remains largely similar.

Observing other data partitions reveals a similar influence of varying hyperparameters. For instance, the necessity for a model filtering algorithm becomes apparent when comparing combinations 1, 2 and 3, with $s = 0$ implying that no models are filtered out, resulting in a performance decrement. When comparing combinations 1, 4, and 5, it is implied that there's a requirement for an effective threshold τ to determine the degree of data imbalance among all clients. A τ value of 0.9 signifies data-sensitivity and a tendency to select a model with a significant amount of local training data. Nevertheless, it is not advisable to set τ=1, as it implies always select the model with the highest quantity of data within a cluster, which could potentially overshadow models with superior model scores, leading to a dip in performance, echoing the findings in section A.3.

An examination of combinations 1, 6, and 7 indicates that the value of the threshold pair (p_{low}, p_{high}) doesn't significantly impact performance, with the commonly utilized 25-percentile and 75-percentile exhibiting the best performance among all combinations.

E.2 Performance comparison on different model representation

As shown in Fig. 4, for the VGG-5 (Spinal FC) model, we pick its first batch of input layers (layer_1), first and fourth batch of fc_spinal_layers (fs_layer1 and fs_layer4), and final/last fully-connected layers (last_layer) as the model representations.The rnd_layer denotes the random selection of 10% of all layers as our model representation. For the iid partitions (homo and iid-dq), there is almost no performance difference observed, irrespective of the layer chosen. However, for the non-iid partitions (noniid-lds and noniid-l18), a performance gap is evident when using different layers as representations. In Fig. 4, it is evident that using parameters from the models' later layers as representation is moderately better than using parameters from their front layers. However, this is a crude observation and does not apply to all situations. Therefore, the selection of an optimal model representation to achieve better performance, particularly for the non-iid data partition, remains an open problem.

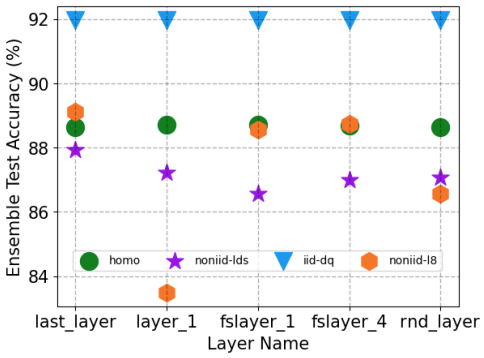

Figure 4: Comparison on the Ensemble Test Accuracy when applying different model representations on `DeDES` for the *EMNIST Letters* Dataset, VGG-5 (Spinal FC) structure when m=200, $K = 120$.

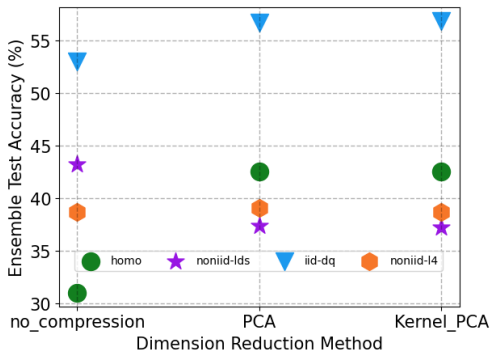

Figure 5: Comparison on the Ensemble Test Accuracy when applying different dimension reduction methods with the last layer as model representation on `DeDES` for the *CIFAR-10* Dataset, Resnet-50 structure when m=50, $K = 30$.

E.3 Importance of dimension reduction methods

As shown in Fig. 5, we compare three DR methods: PCA, Kernel-PCA, and *no-compression* which means we don't compress the model representation (here is the parameters of model's last layer). For PCA and Kernel-PCA, we reduced the model representation to $m - |\mathcal{O}|$ dimensions. We can see that for most of the partitions, the Kernel-PCA is better than other methods such as *PCA* and *no-compression*. This is because the Kernel-PCA can convert non-linear separable data to a new low-dimensional subspace suitable for alignment for linear classification, thus is suitable for the non-linearly separable deep learning models. But we can also see that for the noniid-lds partition, we don't have to do the dimension reduction to get better ensemble learning results. Therefore, similar to the ablation study of model representations, how to design a more effective dimension reduction method is still an open problem.

E.4 Comparative analysis of clustering methods

Our analysis, as reflected in Table 5 of the main paper, shows that for iid data, hierarchical clustering outperformed all other methods with regard to performance. Nevertheless, the variance in accuracy between the methods is minuscule in an iid framework, a point we addressed earlier in section 5.3 of the main paper. The rationale behind these results could be attributed to the fact that under iid conditions, the model parameters across clients tend to be similar and positioned closely in terms of the feature space. In such a situation, the data points representing these model parameters might be challenging to separate distinctly. Hence, as KMeans is predominantly proficient in dealing with spherical and well-spaced data, data points in proximity might not exhibit a spherical form or sufficient separation. On the contrary, hierarchical clustering is capable of repeatedly identifying two models that are nearest to each other, regardless of the shape of the data it is working on.

In contrast, under non-iid circumstances, KMeans is expected to outshine other algorithms, particularly under severely non-iid conditions. This behavior is accounted to the fact that, as compared to iid environments, the model parameters are relatively sparse under non-iid settings. Consequently, models trained on similar datasets will likely cluster in a spherical form, enhancing the KMeans clustering method's efficiency over other clustering algorithms.

Moreover, our observations highlighted that Spectral Clustering consistently delivered the lowest performance. This is because Spectral Clustering relies on a presumption that the data encapsulates a low-dimensional structure, which can be exposed via spectral decomposition. In circumstances where the data deviates from this assumption, the performance of Spectral Clustering deteriorates. Hence, for our model selection scenario, Spectral Clustering does not emerge as an ideal choice.

Continuing further in our experimental trajectory, we extensively tested the DBSCAN clustering algorithm. However, DBSCAN failed to segregate the models effectively. This can be attributed to the fact that DBSCAN is fundamentally a density-based clustering method and handles our model parameters, which represent high-dimensional data. As such, the points become considerably sparse, which results in difficulty in defining density.

References

Wei Fan, Fang Chu, Haixun Wang, and Philip S Yu. Pruning and dynamic scheduling of cost-sensitive ensembles. In *AAAI/IAAI*, pp. 146–151, 2002.

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 603–618, 2017.

Aashish Kolluri, Teodora Baluta, and Prateek Saxena. Private hierarchical clustering in federated networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2342–2360, 2021.

Dawei Li, Di Liu, Ying Guo, Yangkun Ren, Jieyu Su, and Jianwei Liu. Defending against model extraction attacks with physical unclonable function. *Information Sciences*, 628:196–207, 2023a.

Jingtao Li, Adnan Siraj Rakin, Xing Chen, Li Yang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Model extraction attacks on split federated learning. *arXiv preprint arXiv:2303.08581*, 2023b.

Ruixuan Liu, Yang Cao, Hong Chen, Ruoyang Guo, and Masatoshi Yoshikawa. Flame: Differentially private federated learning in the shuffle model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8688–8696, 2021.

P Nagaraj, V Muneeswaran, G Deshik, et al. Ensemble machine learning (grid search & random forest) based enhanced medical expert recommendation system for diabetes mellitus prediction. In *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 757–765. IEEE, 2022.

Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. Federated learning for wireless communications: Motivation, opportunities, and challenges. *IEEE Communications Magazine*, 58(6):46–51, 2020.

Luis A Ortega, Rafael Cabañas, and Andres Masegosa. Diversity and generalization in neural network ensembles. In *International Conference on Artificial Intelligence and Statistics*, pp. 11720–11743. PMLR, 2022.

Buse GA Tekgul, Yuxi Xia, Samuel Marchal, and N Asokan. Waffle: Watermarking in federated learning. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pp. 310–320. IEEE, 2021.

Zuobin Xiong, Zhipeng Cai, Daniel Takabi, and Wei Li. Privacy threat and defense for federated learning with non-iid data in aiot. *IEEE Transactions on Industrial Informatics*, 18(2):1310–1321, 2021.

Haonan Yan, Xiaoguang Li, Hui Li, Jiamin Li, Wenhai Sun, and Fenghua Li. Monitoring-based differential privacy mechanism against query flooding-based model extraction attack. *IEEE Transactions on Dependable and Secure Computing*, 19(4):2680–2694, 2021.