Learning Fast and Slow: Towards Inclusive Federated Learning

Muhammad Tahir Munir, Muhammad Mustansar Saeed, Mahad Ali, Zafar Ayyub Qazi, Ihsan Ayyub Qazi, and Agha Ali Raza

Department of Computer Science, Lahore University of Management Sciences
{18030016,18030047,21100119,zafar.qazi,ihsan.qazi,agha.ali.raza}@lums.edu.pk

Abstract. Today's deep learning systems rely on large amounts of useful data to make accurate predictions. Often such data is private and thus not readily available due to rising privacy concerns. Federated learning (FL) tackles this problem by training a shared model locally on devices to aid learning in a privacy-preserving manner. Unfortunately, FL's effectiveness degrades when model training involves clients with heterogeneous devices; a common case especially in developing countries. Slow clients are dropped in FL, which not only limits learning but also systematically excludes slow clients thereby potentially biasing results. We propose Hasaas; a system that tackles this challenge by adapting the model size for slow clients based on their hardware resources. By doing so, Hasaas obviates the need to drop slow clients, which improves model accuracy and fairness. To improve robustness in the presence of statistical heterogeneity, Hasaas uses insights from the Central Limit Theorem to estimate model parameters in every round. Experimental evaluation involving large-scale simulations and a small-scale real testbed shows that Hasaas provides robust performance in terms of test accuracy, fairness, and convergence times compared to state-of-the-art schemes.

Keywords: Federated Learning, Fairness, Robustness, Developing Countries

1 Introduction

Today's deep neural networks (DNNs) power a wide variety of applications ranging from image classification, speech recognition, to fraud detection [21]. DNNs rely on large amounts of data to make accurate predictions and draw useful inferences. However, such data is often *private*[1] and may not be readily available for centralized collection due to rising privacy concerns and growing adoption of data privacy regulations (e.g., Europe's GDPR [36] and California Consumer Privacy Act [7]). A lack of useful data can limit the effectiveness of DNNs [4].

Federated learning (FL) is a distributed machine learning approach that tackles this problem by training a shared model over data that is distributed across

[1] Private data includes any personal, personally identifiable, financial or sensitive user information [14].

multiple edge devices (e.g., mobile phones), which share model parameters with a centralized server to aid learning in a privacy-preserving manner [26,38]. Unfortunately, FL's effectiveness degrades when model training involves heterogeneous client devices [20,5]; a common case especially in developing countries [3,32,2,37]. With FL, slow clients are dropped from the training process to reduce convergence delays. However, such an approach can degrade test accuracy and reduce fairness due to the systematic exclusion of slow clients [20,27].

We propose Hasaas;[2] a system that tackles this challenge by (i) adapting the model size based on client device capabilities (which we call *Differential Model Serving or DMS*), (ii) using a sub-model selection strategy based on post-activation values, and (iii) using insights from the Central Limit Theorem (CLT) to improve model robustness in the presence of statistical heterogeneity [15].

Serving small models to slow clients and large models to fast clients offers two key benefits. First, it reduces the model training time for slow clients, which decreases their chances of being dropped from the training process thereby improving fairness. Second, it can achieve a better tradeoff between model performance and convergence times compared to serving a single model to all clients.[3] However, realizing these benefits requires answering two key questions: *Given a model, how should we select a sub-model to serve to slow clients?* and *how should we aggregate model parameters from slow and fast clients?*

Hasaas selects a sub-model based on post-activation values of neurons. In particular, it bootstraps the FL process by choosing a random sub-model and allows slow clients to train the sub-model for the first r rounds. Then after every r rounds, it chooses a new sub-model based on post-activation values.[4] This allows neurons with small activation values to be excluded from the sub-model, which improves performance. The neurons that have small activation values are considered less important as they contribute less to the model's output and have a smaller impact on weight updates during training.

With FL, a *random* set of clients are picked in *each* round for model training, which changes the proportion of slow and fast clients in each round. This can reduce model accuracy, especially when there is statistical heterogeneity in data across clients. To improve robustness in such scenarios, Hasaas aggregates model parameters in each round using insights from the CLT by learning the *distribution of the sample mean* of the model parameters [15].[5] Hasaas then randomly draws each model parameter from the learned distribution rather than always using the sample mean. This improves performance especially when there is high variance in the model parameters shared by each client.

We carry out extensive evaluation using (i) large-scale simulations involving the LEAF benchmarking framework for learning in FL settings [9] and (ii) small-scale real testbed experiments involving mobile clients with heterogeneous device

[2] In the Urdu language, Hasaas means *sensitive*.
[3] Small models can reduce accuracy, whereas large models lead to slow convergence.
[4] In case of CNN models, it prunes filters too.
[5] Thus, in every training round, we estimate the mean and variance of each model parameter, which together uniquely identifies a Normal distribution.

capabilities. We compare Hasaas's performance with several notable schemes including FedAvg [26], Adaptive Dropout [6], and FedProx [33] and carry out a detailed ablation study to quantify the benefits of each component of Hasaas. Experiments show that Hasaas achieves robust performance in terms of test accuracy, convergence, and fairness across diverse datasets and models.

Taken together, we make the following contributions in this work.

- We design Hasaas; an adaptive model serving framework for FL that adapts model sizes based on client capabilities (§4). It reduces the computational and communication costs in FL by training on a subset of the model's weights and exchanging smaller sub-models between slow clients and the server instead of the full model updates.
- To achieve better generalization, we propose a CLT-based approach, which outperforms other approaches including FedAvg, Adaptive Dropout, and FedProx (§4).
- We carry out extensive evaluation using large-scale simulations and small-scale testbed experiments involving real smartphones over a wide variety of real-world federated datasets (§5). We make our code available on GitHub.[6] for the benefit of the community.

2 Background and Related Work

Our work focuses on synchronous FL algorithms that proceed in training rounds. These algorithms aim to learn a shared global model with parameters embodied in a real tensor $\boldsymbol{\Gamma}$ from data stored across several distributed clients. In each round $t \geq 0$, the server distributes the current global model $\boldsymbol{\Gamma}_t$ to the set of selected clients S_t with a total of n_t data instances. The selected clients locally execute stochastic gradient descent (SGD) on their data and independently train the model to produce the updated models $\{\boldsymbol{\Gamma}_t^k | k \in S_t\}$. The update of each client k can be expressed as:

$$\boldsymbol{\Gamma}_t^k = \boldsymbol{\Gamma}_t - \alpha H_t^k, \ \ \forall k \in S_t \tag{1}$$

where H_t^k is the gradients tensor for client k in training round t, and α is the learning rate chosen by the server. Each selected client k then sends the update back to the server, where the new global model ($\boldsymbol{\Gamma}_{t+1}$) is constructed by aggregating all client-side updates as follows:

$$\boldsymbol{\Gamma}_{t+1} = \sum_{\forall k \in S_t} \frac{n_k}{n_t} \cdot \boldsymbol{\Gamma}_t^k \tag{2}$$

where n_k is the number of data instances of client c and $n_t = \sum_{\forall k \in S_t} n_k$. Hence, $\boldsymbol{\Gamma}_{t+1}$ can be written as:

$$\boldsymbol{\Gamma}_{t+1} = \boldsymbol{\Gamma}_t - \alpha_t H_t \tag{3}$$

where $H_t = \frac{1}{n_t} \sum_{k \in S_t} n_k H_t^k$.

[6] https://github.com/FederatedResearch/hasaas

Fairness in FL. Due to the heterogeneity in client devices and data in federated networks, it is possible that the performance of a model will vary significantly across the network. This concern, also known as representation disparity, is a major challenge in FL, as it can potentially result in uneven outcomes for the devices. Following Li et al. [22], we provide a formal definition of this fairness in the context of FL below.

Definition. *We say that a model W_1 is more fair than W_2 if the test performance distribution of W_1 across the network is more uniform than that of W_2, i.e., $std\{F_k(W_1)\}_{k\in[K]} < std\{F_k(W_2)\}_{k\in[K]}$ where $F_k(\cdot)$ denotes the test loss on device $k \in [K]$, and $std\{\cdot\}$ denotes the standard deviation.*

We note that there exists a tension between variance and utility in the definition above; in general, the goal is to *lower* the variance while maintaining a reasonable average performance (e.g., average test accuracy). Several prior works have separately considered either fairness or robustness in FL. For instance, fairness strategies include using minimax optimization to focus on the worst-performing devices [18,39] or reweighting the devices to allow for a flexible fairness/accuracy tradeoff (e.g., [23]).

2.1 Related Work

Several recent and ongoing efforts aim to tackle system and statistical heterogeneity in FL [6,8,13,10,24,33,20,19,39,26]. In this section, we discuss works that are most closely related to our study.

System heterogeneity. A number of schemes aim to reduce the impact of client heterogeneity by serving *smaller* models. These schemes differ based on (i) whether they do model pruning on the client-side [19] or the server-side [6], (ii) the criteria used for pruning (e.g., [12,8]), and (iii) whether they serve a single model to all clients or not [19,8,6]. For example, PruneFL [19] performs initial model pruning at a selected client and further adaptive pruning on the server-side. It serves the same pruned model to *all* clients. Moreover, because the initial model pruning is carried out on only one selected client and its data, the pruned model can be biased towards the selected client. HeteroFL [12] trains local models based on clients' device characteristics. It pre-defines subset models with different complexity levels and assigns the same model to clients belonging to the same complexity level. However, the subset models are statically defined, which limits model performance. AFD [6] trains a subset model using either a single-model or multi-model serving approach. With the former approach, the same model is served to each client and monitored for average training loss. A positive score is given for decreasing loss, while a new model is served if loss increases. While AFD outperforms Federated Dropout (FD), which relies on random dropping to select sub-models, such a strategy can trigger frequent model changes, which can negatively affect model performance. We demonstrate this in Section 5.[7]

[7] With multi-model AFD, a different subset model is used by *each* client, all of the same size. However, training with a small fraction of clients in each round – a typical scenario in FL – makes the algorithm behave randomly, just like the FD scheme [6].

Statistical heterogeneity. A number of approaches aim to tackle statistical heterogeneity by either modifying the (i) client selection strategy of FL (e.g., [25]), (ii) FedAvg aggregation method (e.g., [13]) or (iii) objective function to include a regularization term (e.g., [33]). For example, PFedR [25] is a client selection strategy in which the server generates dummy datasets from the inversion of local model updates, identifies clients with large distribution divergences, and aggregates updates from highly relevant clients only. Generating dummy data from client updates raises privacy concerns. Also, in case of secure multi-party aggregation where client updates are meaningless without aggregation, generating dummy data from those updates will not help in identifying client distribution divergences. In such cases, this approach may not yield the desired results. FedDNA [13] is a parameter aggregation method for FL that aggregates gradient and statistical parameters, separately. While the gradient parameters are aggregated using FedAvg, the statistical parameters are aggregated collaboratively to reduce the divergence between the local models and the central model. This technique is only applicable to models with a batch normalization layer.

FedProx [33] uses partial work from resource-constrained devices to tackle system heterogeneity and adds a regularization term in the FedAvg objective function to improve performance under statistical heterogeneity. However, incorporating partial updates from slow clients can reduce test accuracy if the model is not trained enough.[8] In addition, downloading a large model from the server can still be a significant burden for slow clients, particularly those located in regions with limited connectivity.

Hasaas tackles *both* system and statistical heterogeneity by combining the benefits of differential model serving and CLT-based aggregation. It extends the state-of-the-art in terms of system heterogeneity by employing an approach in which slow clients are served a small model whereas the fast clients continue to receive the global model. This is unlike AFD and PruneFL that serve smaller models to *all* clients, which can degrade model accuracy. Moreover, unlike HeteroFL that uses pre-defined subset models, Hasaas dynamically updates small models based on average post-activation values in the global model. As a result, sub-models that are not performant are discarded as training progresses. We delve deeper into the design of Hasaas in the following section.

3 Problem Motivation

It is common for distributed clients in FL to exhibit considerable heterogeneity in terms of computational resources (e.g., number of CPU cores, RAM size) and network bandwidth [8,33]. This heterogeneity impacts both the model accuracy and the training time of the FL process [9]. Consequently, resource-constrained edge devices (e.g., entry-level smartphones), which are prevalent in developing countries, are either unable to train models due to their limited compute and

[8] Moreover, if the slow clients are unable to run the large model due to resource constraints, they cannot participate in the training process.

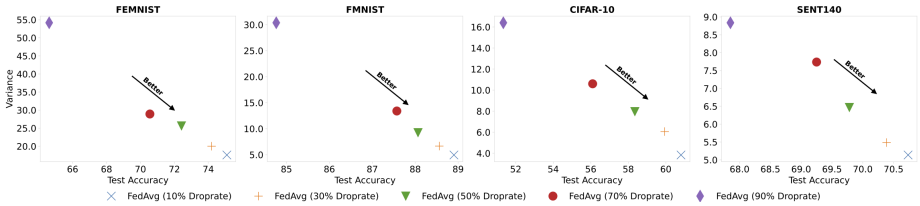

Fig. 1: Impact of systems heterogeneity on test accuracy and fairness in FedAvg for different client drop rates and datasets. As the fraction of slow clients increases, the test accuracy, and fairness decrease.

memory resources or take a prohibitively long time in training [26,32,28]. According to a study involving one of the largest online social networks, 57% of smartphones in developing regions, had 1 GB or less RAM [29].[9] Such entry-level smartphones frequently operate under low memory regimes, which is known to degrade performance [32,37,2]. Unfortunately, slow clients (or stragglers) are dropped in FL schemes (e.g., FedAvg) for efficiency reasons because waiting for slow clients to report their updates can increase convergence times. However, dropping slow clients can (i) degrade test accuracy and (ii) lead to unfairness.

System heterogeneity degrades robustness and fairness. To evaluate the impact of slow clients in FedAvg on model accuracy and fairness, we simulate different levels of system heterogeneity using LEAF [9]. In particular, we vary the client drop rate (CDR)[10] from 10% to 90% and carry out evaluation on multiple real datasets including FEMNIST, FMNIST, CIFAR-10, and Sent140. Figure 1 shows that with FedAvg, system heterogeneity negatively impacts both model robustness as well as fairness.[11] In particular, test accuracy decreases as CDR increases across *all* datasets whereas the variance of the test loss (across clients), which captures model fairness, generally increases with CDR.

Homogeneous model serving (HMS) either slows convergence or degrades model performance. To quantify the impact of serving the same model to all clients we train a CNN model over the FEMNIST dataset [9] and measure the test accuracy and the time to complete 100 training rounds on two real smartphones, i.e., Nokia 1 (Quadcore, 1 GB RAM) and Nexus 6P (Octacore, 3 GB RAM). These devices represent slow and fast clients, respectively.[12] We find that with FedAvg, serving the same model to both slow and fast clients leads to slow convergence. In our testbed, it took 15.9 hours with FedAvg to complete 100 train rounds. To address this system heterogeneity, one can serve

[9] In 2018, ∼300 million Android phones shipped globally had 1 GB or less RAM [1].
[10] CDR is the fraction of slow clients in the system. With FedAvg, such clients are dropped from the training process.
[11] Similar to Li et al. [22], we capture model fairness using the variance of test loss across clients. Thus, the more uniform the loss distribution is, the fairer the model.
[12] The memory specifications of these devices represent a wide range of smartphones. In developing regions, phones with 1 GB or less RAM had a market share of 57% compared to 20% in developed regions. Phones $\geq$ 3 GB RAM had less than 25% market share in developing regions and over 50% share in developed countries [29].

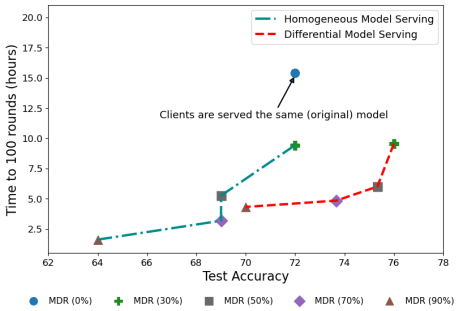

Fig. 2: Time to complete 100 rounds of training on a real testbed as a function of test accuracy for different model drop rates (MDR).

a smaller model (e.g., a subset of the original model) to all clients [8]. Figure 2 shows that indeed serving smaller models reduces the time to complete 100 rounds. However, it also reduces test accuracy. For example, increasing the model drop rate (MDR) (i.e., the pruning percentage) from 30% to 50% reduces the test accuracy by 4.2%. Another approach is to serve a smaller model to only slow clients while the faster clients continue to train on the large model (we refer to this approach as *Differential Model Serving*, which is part of Hasaas). We find that such an approach considerably improves test accuracy while also achieving significant reductions in convergence time. For example, when the MDR is 50%, DMS improves test accuracy by 9.4% over HMS with similar convergence times. However, we see diminishing returns beyond 50% MDR because in this regime the fast client becomes the bottleneck as opposed to the slow client.

In summary, serving a single model to all clients presents a tradeoff between convergence time and model accuracy. DMS can address this tradeoff by serving models of different sizes to slow and fast clients.

4 Design

Hasaas tackles both system heterogeneity and the statistical challenges with heterogeneous data using the following design features:

1. *Differential model serving.* Clients are served models with different sizes based on their capabilities.
2. *Sub-model selection using activations.* Small sub-models are selected for slow clients based on post-activation values.
3. *Model generalization using insights from the Central Limit Theorem.* To improve performance robustness, especially over non-IID datasets, we use a CLT-based approach to choose model parameters.

A. Differential Model Serving. DMS has several benefits compared to serving the *same* model to all clients participating in FL. First, due to the large heterogeneity in mobile device characteristics, training a single model over all clients

can lead to widely different training times, which can result in frequent dropping of slow clients from FL,[13] potentially leading to unfairness across clients [8,5,22]. Second, it is difficult to choose a single model that allows all clients to participate in FL training while achieving high accuracy; small models can degrade model accuracy, whereas large models lead to the dropping of slow clients. DMS addresses these challenges by allowing model sizes to be *adapted* based on device characteristics such as the number of CPU cores, memory size, and GPU characteristics (if present). Thus, slow clients are served smaller models than faster clients, which can improve fairness by reducing the dropping of clients from the FL process.

B. Sub-model Selection. Given a model size, a key design question in Hasaas is, "*which sub-model should we serve to slow clients?*" There are many possible sub-models one can pick. In a feed-forward neural network, suppose we allow dropping of neurons from all layers including the input and output layers, then the number of distinct sub-models are lower bounded by $\binom{n_1}{\lfloor f.n_1 \rfloor}\binom{n_2}{\lfloor f.n_2 \rfloor}..\binom{n_l}{\lfloor f.n_l \rfloor}$, where n_i is the number of neurons in layer i, $(1-f)$ is the pruning fraction (i.e., fraction of neurons dropped from each layer), and l is the total number of layers. Finding the best sub-model from among such a large set of possible sub-models is challenging. We address this challenge with two strategies. First, we bootstrap the FL process by choosing a random sub-model and allowing slow clients to train over the sub-model for the first r rounds. This allows us to assess the parts of the sub-model that contribute the most to the learning task. Second, after every r rounds, we choose a new sub-model based on the post-activation values of neurons (in case of CNN models, filters too) [35]. This allows neurons with small activation values to be excluded from the sub-model. This approach can enable better model selection than randomly picking models in each round or choosing a different random model in each round [8].

C. Model Generalization. Training a model in FL is challenging due to the statistical variations in data distributed across clients, which impacts both model accuracy as well as model convergence [8,33]. This is exacerbated by the fact that in each round, FL picks k *random* clients for training from a pool of N clients. As a result, choosing the sample mean of the weights of each model parameter across clients in the *current round* may not be representative of clients picked in the next round, especially when k is much smaller than N, which is a typical case in FL [26].[14] To generalize model training, we use insights from CLT, which posits that the distribution of the sample mean of IID random variables converges to a Normal distribution.[15] We consider the setting, where each client i draws independent samples from a distribution D with finite mean μ and finite variance σ^2. Let $X_j^i \sim D$ be the random variable denoting weight of the jth model parameter for client i. Then FL aims to learn the average $\bar{X}_j = \sum_{i=1}^{N} p_i X_j^i$

[13] Some clients may not be able to run large models at all due to memory constraints.

[14] While the sample mean is an unbiased estimator of the population mean, the variance of the sample mean depends on the sample size (i.e., the number of clients).

[15] If each client's model weights follow a different distribution, one can use generalizations of CLT, such as the Lyapunov CLT and Lindeberg CLT [15].

across all clients, where p_i is the proportion of samples trained by client i. CLT posits that $\bar{X}_j$ converges in distribution to the Normal distribution with mean μ and finite variance σ^2/N. Thus, larger the variance, the more imprecise is our estimate of $\bar{X}_j$. To achieve better generalization, we randomly draw samples from the learnt sample mean distribution and use them for the next round, where a new set of random clients are selected for model training.

By drawing random samples from the sample mean distribution rather than just using the sample mean in each round, we ensure that clients with large values for the model parameters do not skew the learning process. To estimate the distribution of the sample mean, we use parameters shared by clients in each round *independently*. As a result, there is no pooling of parameter values across rounds due to dependencies introduced by SGD. In our evaluation, we show that this strategy improves accuracy, robustness, and convergence speed compared to just using the sample mean.

Algorithm 1: Hasaas

Input: Model Dropout Rate (k%), Pruning Round (r)
Server executes:
Initialize: Global model W_0, mask $M \leftarrow 0$;
for *each round* $t = 1, 2, \ldots, T$ **do**
  if $t > 1$ **then**
    Select sub-model w_t from W_t based on mask M
  else
    $w_t \leftarrow$ Random selection k% ;
    $M \leftarrow$ Indexes of sub-model w_t;
  end
  $C_t \leftarrow$ (select n clients randomly) ; ▷ `n ≤ N`
  Send W_t or w_t to C_t based on their device characteristics
  for *each client* $c \in C_t$ **do**
    if c *is slow* **then**
      Train sub-model w_t**:**
      $activations_t^c, w_{t+1}^c = \ell(w_t, c)$;
      $W_{t+1}^c = Broadcast(w_{t+1}^c, M)$;
    else
      Train large model W_t**:**
      $activations_t^c, W_{t+1}^c = \ell(W_t, c)$;
    end
  end
  $activations_t = \frac{1}{n} \sum_{c \in C_t} activations_t^c$;
  $\mu = \sum_{c \in C_t} \frac{s_c}{S_t} W_{t+1}^c$; ▷ S_t: `Total samples`
  $\sigma = \sqrt{\frac{\sum_{c \in C_t} s_c (W_{t+1}^c - \mu)^2}{S_t - 1}}$;
  $\sigma = \frac{\sigma}{\sqrt{t}}$;
  $W_{t+1} = \mathcal{N}(\mu, \sigma^2)$;
  if $(t \mod r) == 0$ **then**
    Update M using activations of dense layer and ℓ1-Norm of CNN filters
  end
end

4.1 Algorithm

At the start of the FL process, the server initializes a global model W_0 and a mask M to keep track of the smaller model sent to the slow clients; see Algorithm

1. The server randomly picks a smaller model w_0 (which we call a *sub-model*) from the global model W_0. It does so randomly at the start as it does not have any prior information to effectively choose a sub-model. The server selects n clients randomly and sends the sub-model w_0 or the large model W_0 to selected clients based on their device characteristics. Let C_t denote the set of all clients selected in round t. Each client trains its model and sends the model updates as well as activations of dense layers to the server. The server aggregates these updates and activations. In weighted aggregation, s_c and S_t represent the number of training samples of client c and the number of training samples in round t, respectively. Based on average activations, and $\ell 1$-norm of the CNN filters, the server picks the optimal sub-model for slow clients after every r rounds. The only two parameters of Hasaas are k (MDR), representing the percentage of neurons/filters to be dropped from the dense and convolution layers and the Mask Update Round (MUR) r, at which M is updated.

Aggregation. Hasaas uses insights from CLT to aggregate clients' model parameters. Server calculates the weighted mean μ (weighted by number of data samples) and standard deviation σ of each parameter across clients. It then uses μ and σ to randomly sample parameters from the Normal distribution to send back to the clients. As training progresses and the model becomes stable, large changes in model parameters can adversely impact performance. As a result, we continue decreasing σ proportional to $1/\sqrt{t}$ (current round), which limits model deviations from the stable parameters and helps with generalization.

5 Evaluation

We now present empirical results for Hasaas using large-scale simulations and small-scale testbed experiments under federated settings. We demonstrate the effectiveness of Hasaas in the presence of both system and statistical heterogeneity and study its convergence, robustness, and fairness properties. All code and scripts for generating the paper results are available here.

A. Experimental Details. We evaluate Hasaas on multiple models, tasks, and real-world federated datasets. We implement Hasaas in LEAF [9] – a benchmarking framework for FL to simulate our federated setup – and evaluate its performance on CNN and LSTM models, and five real-world datasets. Specifically, we use CIFAR-10, Federated extended MNIST (FEMNIST), FEMNIST (skewed), Fashion-MNIST (FMNIST) for CNN and Sent140 for the LSTM model. We compare Hasaas performance with FedAvg [26], FedProx [33], and Single-Model Adaptive Federated Dropout [6].

Real Data. The datasets we use are curated from prior work in FL [33,22,8] and recent FL benchmarks in LEAF [9]. FMNIST, FEMNIST, and Sent140 are non-IID datasets. To study Hasaas under an IID dataset, we curated CIFAR-10 in an IID fashion, where each example has the same probability to belong to any device. We then study a more complex 62-class FEMNIST dataset [11,8]. Details of datasets, models, and workloads are provided in Appendix A.2 on GitHub.

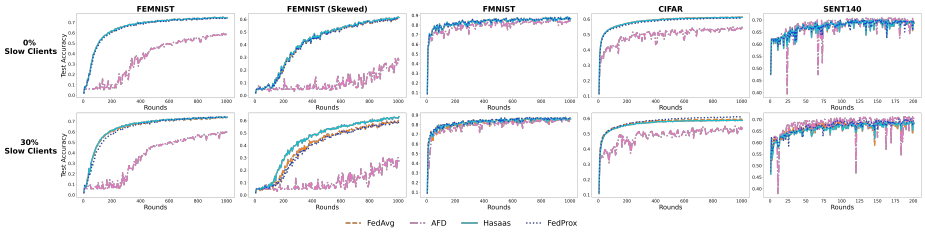

Fig. 3: Test accuracy as a function of round number for 0% and 30% system heterogeneity. 0% client drop rate indicates no system heterogeneity. If there is no system heterogeneity, Hasaas provides better or similar performance to FedAvg and FedProx.

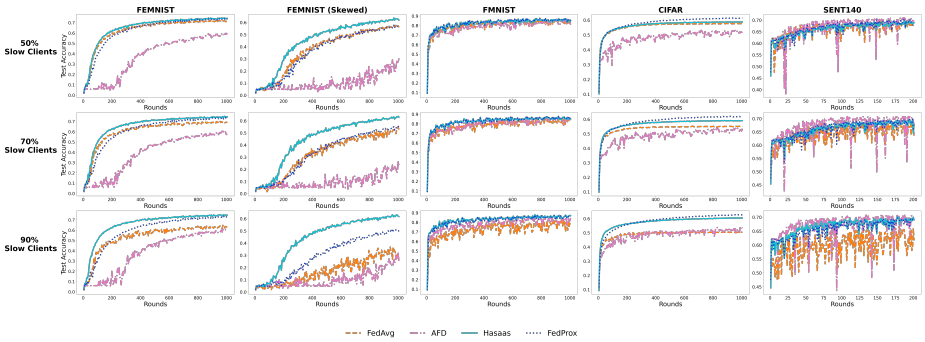

Fig. 4: Test accuracy as a function of round number for 50%, 70% and 90% system heterogeneity, where system heterogeneity referes to the percentage of slow clients. Hasaas results in significant convergence improvements relative to other schemes. As statistical heterogeneity increases Hasaas provides robust performance. We also report train loss in Appendix A.5.

Hyperparameters. We evaluate each dataset using three learning rates: {0.01, 0.001, 0.0003}. While smaller learning rates mean the model takes longer to converge, the behavior of all techniques remains the same relative to each other. As suggested in an earlier work [33], we use a learning rate of 0.001 and 0.01 for the CNN and LSTM models, respectively. We set the number of selected devices per round to 10. Unless specified otherwise, we set $MDR = 50\%$ and $r = 10$, which results in 50% of the filters and neurons being dropped from the convolution and dense layers, and leads to 50% fewer cells in the LSTM layers for the slow client. For a fair comparison, we fix the randomly selected devices, the slow clients, and mini-batch orders across all runs and report the average results of 5 runs.

B. System & Statistical Heterogeneity.

System heterogeneity. For studying the impact of heterogeneity, we vary the percentage of slow devices (0%, 30%, 50%, 70%, and 90%). We emulate devices as slow if they cannot train the model for E epochs due to their system constraints. Settings where 0% devices are slow correspond to environments *without* system heterogeneity, whereas 90% of the slow devices correspond to highly heterogeneous environments. FedAvg simply drops slow clients upon reaching

the global clock cycle but Hasaas incorporates the updates from these devices as they train a subset model and are able to send updates on time. AFD also incorporates slow-device updates as it trains a smaller model on all devices. FedProx incorporates partial updates from the slow devices.

Figures 3 and 4 show that Hasaas achieves robust performance for different levels of system heterogeneity compared to FedAvg, FedProx and AFD. As system heterogeneity increases, FedAvg's performance degrades significantly. FedProx performs better than FedAvg because it incorporates partial updates from slow clients and modifies the objective function to include a regularization term to avoid over-fitting on clients' data. However, incorporating partial updates can have a negative impact if the model is not trained for enough epochs. Thus, as the number of slow clients increases, FedProx achieves lower test accuracy relative to Hasaas especially with non-IID datasets (e.g., FEMNIST).

Statistical heterogeneity. We use datasets with varying degrees of IID-ness to evaluate Hasaas under statistical heterogeneity. We use two versions of FEMNIST dataset; one non-IID version is generated using LEAF, which is employed by Ditto [22]. We generate a skewed non-IID version of the FEMNIST dataset in which each client contains data with only 5 classes of the FEMNIST dataset. This approach has been used in prior works to generate skewed non-IID datasets. Figures 3 and 4 show the test accuracy of all approaches on different datasets. These results indicate that as the degree of non-IID-ness increases, Hasaas provides better generalizability as evidenced by the high test accuracy. For each dataset, Hasaas provides either faster convergence compared to other schemes while also achieving better or comparable test accuracy.

In case of the IID CIFAR dataset, Hasaas achieves fast convergence to a test accuracy of 50% than FedProx but results in 2% lower test accuracy after 1000 rounds. This occurs because in the presence of system heterogeneity, while Hasaas serves a sub-model to slow clients FedProx continues to serve the same large model to all clients. The IID nature of the data leads to a lower variance in the sample mean of the model parameters, resulting in less benefits of the CLT approach. In case of SENT140, Hasaas performs comparable to other approaches, except FedAvg, which experiences large fluctuations as system heterogeneity increases. In case of high system (90%) and statistical heterogeneity (FEMNIST skewed), Hasaas provides 27% test accuracy improvement over FedAvg, 14% over FedProx, and 34% over AFD; see Figure 4.

Fairness. Due to statistical heterogeneity in federated settings, the performance of a model may vary significantly across different devices, resulting in *representation disparity* [17]. In Hasaas, we serve a subset model to slow clients, which potentially has a larger risk of representation disparity. We empirically show that in addition to improving accuracy, Hasaas also offers improved fairness. Hasaas picks the best subset model after every r (a tunable parameter) rounds for slow clients. Variance of test loss across clients can be seen in Figure 5 and Appendix A.5. Interestingly, Hasaas provides better average test accuracy as well as achieves lower variance across clients compared to FedAvg and Hasaas without CLT. Figure 5 compares the robustness and fairness of Hasaas,

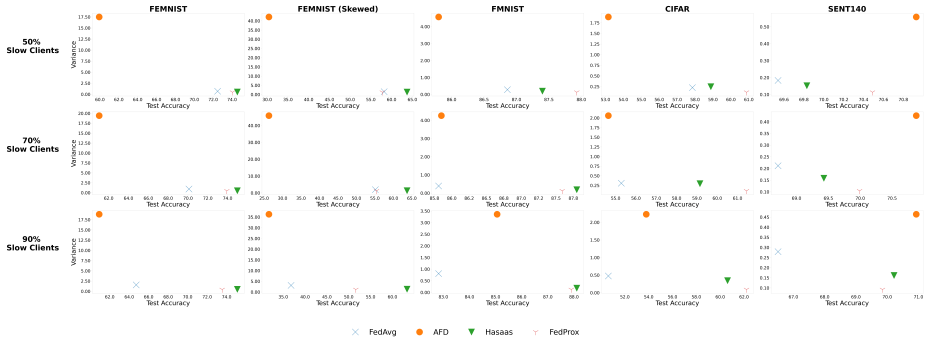

Fig. 5: Variance of test loss as a function of average test accuracy.

FedProx, AFD and FedAvg. Results show that Hasaas provides robust and fair performance as it trains on all clients and uses CLT for improved generalization.

C. Ablation Study.[16]

Benefits of activation-based model pruning. We compared our activation-based sub-model selection strategy with the random sub-model selection strategy, which selects a new model in each round. We find that our activation-based approach consistently outperforms random selection, yielding a test accuracy improvement ranging from 3.7% to 6.9% for CDRs of 30% and 90%, respectively.

Benefits of model generalization module. We also evaluated Hasaas' model generalization module in our ablation study. We find that incorporating the generalization module provided up to 6.7% improvement in test accuracy, compared to the model without the module.

Choice of normalizing σ by $1/\sqrt{t}$. We conducted an empirical evaluation of various normalizing factors for σ in order to identify the optimal approach for improving the performance of a model using random sampling from a normal distribution. Our findings indicate that using a large normalizing factor results in a reduction in the improvement provided by this sampling technique, as the value of σ becomes smaller and the sampled weights tend to remain close to the mean. This effectively reduces the effectiveness of the technique to that of FedAvg. On the other hand, failing to normalize σ leads to large model parameters, which can cause the model to become unstable, which is illustrated in Appendix. Based on our empirical evaluation, we suggest using a normalizing factor that keeps σ moderate and increases as the round progresses and the model weights become more stable, leading to better accuracy. In our experiments, we found that $1/\sqrt{t}$ was a particularly effective normalizing factor for σ.

Using CLT with FedProx & FedAvg. We performed experiments by applying the model generalization module of Hasaas to FedAvg and FedProx on the FEMNIST skewed data. We observe that the differences between these schemes (i.e., FedAvg and FedProx) with and without CLT are small and not significant. With vanilla FedAvg, CLT does not provide any significant improvement be-

[16] Due to space limitations, figures related to ablation experiments are available in the Appendix A.1 on our GitHub.

cause slow clients are dropped in FedAvg. The same trend holds with FedProx, which also does not serve small models to slow clients but instead incorporates partial work and adds a regularization term to the loss function.

D. Real Testbed Experiments. We implement Hasaas on a real FL testbed. We use PySyft [31], an open-source framework for FL, to train models on mobile devices. Mobile devices connect with the server to download models and train them using KotlinSyft [30]. The server communicates with the clients using Google Firebase Services [16]. We perform evaluations on real smartphones, i.e., Nexus 6P (3 GB RAM, Octacore) and Nokia 1 (1 GB RAM, Quadcore) as fast and slow client, respectively. We employed real datasets, namely FEMNIST and a CNN model from the LEAF benchmark, to investigate the impact of model size on model training times. Further details of the model can be found in the Appendix A.2. We present the training time for various model drop rates on the slow device in Hasaas in Appendix A.3. The large model is the unpruned model served to the slow client and a 30% MDR implies a 30% pruned model. Our results indicate that increasing the MDR decreases the training time. Specifically, a 50% MDR leads to a 66.7% reduction in training time due to reduced FLOPs, as shown in Appendix A.3. A MDR of 50% results in 3.8× fewer FLOPs and a training time reduction of roughly 2.9×. We examine the impact of network heterogeneity on convergence time for different MDRs in Appendix A.4.

6 Limitations and Future Work

Differential model serving. We only evaluated Hasaas using a 2-model approach (i.e., fast clients train over the global model whereas slow clients train over a sub-model). In the future, it would be useful to examine the effectiveness of customizing model sizes for *each* client based on their characteristics.

Impact on mobile user experience. By including slow clients in the FL training process, it is possible that these clients may be further slowed down thereby degrading mobile users' experience of other applications (e.g., mobile browsers). This could be explored in future works.

Hasaas and multi-task learning. In multi-task learning [34], the goal is to train *personalized* models for each device independent of sizes whereas Hasaas focuses on reducing the overhead of model training for improving inclusiveness in the presence of client heterogeneity.

7 Conclusion

We presented the design and evaluation of Hasaas, an inclusive framework for federated learning that achieves improved learning and fairness properties in the presence of client heterogeneity. Hasaas's differential model serving ensures that slow clients are not dropped from the training process and achieve training times similar to fast clients whenever possible. Our evaluation involving large-scale simulations and a small-scale real testbed of mobile clients shows that Hasaas achieves robust performance across a variety of real-world federated datasets.

References

1. Build for Android (Go edition): optimize your app for global markets (Google I/O '18), https://bit.ly/2UKLQDl
2. Abdullah, M., Qazi, Z.A., Qazi, I.A.: Causal impact of android go on mobile web performance. In: Proceedings of the 22nd ACM Internet Measurement Conference. p. 113–129. IMC '22, Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3517745.3561456
3. Ahmad, S., Haamid, A.L., Qazi, Z.A., Zhou, Z., Benson, T., Qazi, I.A.: A view from the other side: Understanding mobile phone characteristics in the developing world. In: Proceedings of the 2016 Internet Measurement Conference. p. 319–325. IMC '16 (2016), https://doi.org/10.1145/2987443.2987470
4. Bonawitz, K., Kairouz, P., McMahan, B., Ramage, D.: Federated learning and privacy: Building privacy-preserving systems for machine learning and data science on decentralized data. Queue **19**(5), 87–114 (nov 2021), https://doi.org/10.1145/3494834.3500240
5. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konecny, J., Mazzocchi, S., McMahan, H.B., Van Overveldt, T., Petrou, D., Ramage, D., Roselander, J.: Towards federated learning at scale: System design (2019), http://arxiv.org/abs/1902.01046, cite arxiv:1902.01046
6. Bouacida, N., Hou, J., Zang, H., Liu, X.: Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In: IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 1–6 (2021). https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484526
7. BUKATY, P.: The California Consumer Privacy Act (CCPA): An implementation guide. IT Governance Publishing (2019), http://www.jstor.org/stable/j.ctvjghvnn
8. Caldas, S., Konečný, J., McMahan, H.B., Talwalkar, A.: Expanding the reach of federated learning by reducing client resource requirements. CoRR **abs/1812.07210** (2018), http://arxiv.org/abs/1812.07210
9. Caldas, S., Wu, P., Li, T., Konečný, J., McMahan, H.B., Smith, V., Talwalkar, A.: LEAF: A benchmark for federated settings. CoRR **abs/1812.01097** (2018), http://arxiv.org/abs/1812.01097
10. Chou, L., Liu, Z., Wang, Z., Shrivastav, A.: Efficient and less centralized federated learning. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2021), https://2021.ecmlpkdd.org/wp-content/uploads/2021/07/sub_932.pdf
11. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: an extension of MNIST to handwritten letters. CoRR **abs/1702.05373** (2017), http://arxiv.org/abs/1702.05373
12. Diao, E., Ding, J., Tarokh, V.: Heterofl: Computation and communication efficient federated learning for heterogeneous clients. arXiv preprint arXiv:2010.01264 (2020)
13. Duan, J.H., Li, W., Lu, S.: Feddna: Federated learning with decoupled normalization-layer aggregation for non-iid data. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2021), https://2021.ecmlpkdd.org/wp-content/uploads/2021/07/sub_539.pdf

14. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) Theory of Cryptography. pp. 265–284. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
15. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley (January 1968), http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0471257087
16. Google: Firebase services. https://firebase.google.com
17. Hashimoto, T.B., Srivastava, M., Namkoong, H., Liang, P.: Fairness without demographics in repeated loss minimization (2018)
18. Hu, Z., Shaloudegi, K., Zhang, G., Yu, Y.: Fedmgda+: Federated learning meets multi-objective optimization. CoRR **abs/2006.11489** (2020), https://arxiv.org/abs/2006.11489
19. Jiang, Y., Wang, S., Valls, V., Ko, B.J., Lee, W.H., Leung, K.K., Tassiulas, L.: Model pruning enables efficient federated learning on edge devices. IEEE Transactions on Neural Networks and Learning Systems pp. 1–13 (2022). https://doi.org/10.1109/TNNLS.2022.3166101
20. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., et al.: Advances and open problems in federated learning. CoRR **abs/1912.04977** (2019), http://arxiv.org/abs/1912.04977
21. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539, https://doi.org/10.1038/nature14539
22. Li, T., Hu, S., Beirami, A., Smith, V.: Ditto: Fair and robust federated learning through personalization. CoRR **abs/2012.04221** (2020), https://arxiv.org/abs/2012.04221
23. Li, T., Sanjabi, M., Smith, V.: Fair resource allocation in federated learning. CoRR **abs/1905.10497** (2019), http://arxiv.org/abs/1905.10497
24. Li, X.C., Zhan, D.C., Shao, Y., Li, B., Song, S.: Fedphp: Federated personalization with inherited private models. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2021), https://2021.ecmlpkdd.org/wp-content/uploads/2021/07/sub_654.pdf
25. Ma, Z., L, Y., Li, W., Cui, S.: Beyond random selection: A perspective from model inversion in personalized federated learning. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2022), https://2022.ecmlpkdd.org/wp-content/uploads/2022/09/sub_810.pdf
26. McMahan, B., Moore, E., Ramage, D., Hampson, S., Arcas, B.A.y.: Communication-Efficient Learning of Deep Networks from Decentralized Data. In: Singh, A., Zhu, J. (eds.) Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (20–22 Apr 2017), https://proceedings.mlr.press/v54/mcmahan17a.html
27. Mohri, M., Sivek, G., Suresh, A.T.: Agnostic federated learning. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 4615–4625. PMLR (09–15 Jun 2019), https://proceedings.mlr.press/v97/mohri19a.html
28. Naseer, U., Benson, T.A., Netravali, R.: Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In: Proceedings of

the 22nd International Workshop on Mobile Computing Systems and Applications. p. 71–77. HotMobile '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3446382.3448652, https://doi.org/10.1145/3446382.3448652
29. Naseer, U., Benson, T.A., Netravali, R.: Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In: Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications. p. 71–77. HotMobile '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3446382.3448652, https://doi.org/10.1145/3446382.3448652
30. OpenMined: Kotlinsyft. https://github.com/OpenMined/KotlinSyft/
31. OpenMined: Pysyft. https://github.com/OpenMined/PySyft/
32. Qazi, I.A., Qazi, Z.A., Benson, T.A., Murtaza, G., Latif, E., Manan, A., Tariq, A.: Mobile web browsing under memory pressure. SIGCOMM Comput. Commun. Rev. **50**(4), 35–48 (Oct 2020). https://doi.org/10.1145/3431832.3431837, https://doi.org/10.1145/3431832.3431837
33. Sahu, A.K., Li, T., Sanjabi, M., Zaheer, M., Talwalkar, A.S., Smith, V.: Federated optimization in heterogeneous networks. arXiv: Learning (2020)
34. Smith, V., Chiang, C.K., Sanjabi, M., Talwalkar, A.: Federated multi-task learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 4427–4437. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
35. Tan, C.M.J., Motani, M.: DropNet: Reducing neural network complexity via iterative pruning. In: III, H.D., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 9356–9366. PMLR (13–18 Jul 2020), https://proceedings.mlr.press/v119/tan20a.html
36. Voigt, P., Bussche, A.v.d.: The EU General Data Protection Regulation (GDPR): A Practical Guide. Springer Publishing Company, Incorporated, 1st edn. (2017)
37. Waheed, T., Akhtar, Z., Qazi, I.A., Qazi, Z.A.: Coal not diamonds: How memory pressure falters mobile video qoe. In: ACM CoNEXT (2022)
38. Wang, J., Charles, Z., Xu, Z., Joshi, G., et al.: A field guide to federated optimization. CoRR **abs/2107.06917** (2021), https://arxiv.org/abs/2107.06917
39. Xu, C., Qu, Y., Xiang, Y., Gao, L.: Asynchronous federated learning on heterogeneous devices: A survey (2021)