FEDTRANS: EFFICIENT FEDERATED LEARNING VIA MULTI-MODEL TRANSFORMATION

Yuxuan Zhu [1] **Jiachen Liu** [2] **Mosharaf Chowdhury** [2] **Fan Lai** [1]

ABSTRACT

Federated learning (FL) aims to train machine learning (ML) models across potentially millions of edge client devices. Yet, training and customizing models for FL clients is notoriously challenging due to the heterogeneity of client data, device capabilities, and the massive scale of clients, making individualized model exploration prohibitively expensive. State-of-the-art FL solutions personalize a globally trained model or concurrently train multiple models, but they often incur suboptimal model accuracy and huge training costs.

In this paper, we introduce FedTrans, a multi-model FL training framework that automatically produces and trains high-accuracy, hardware-compatible models for individual clients at scale. FedTrans begins with a basic global model, identifies accuracy bottlenecks in model architectures during training, and then employs model transformation to derive new models for heterogeneous clients on the fly. It judiciously assigns models to individual clients while performing soft aggregation on multi-model updates to minimize total training costs. Our evaluations using realistic settings show that FedTrans improves individual client model accuracy by 14% - 72% while slashing training costs by $1.6\times$ - $20\times$ over state-of-the-art solutions.

1 INTRODUCTION

Federated learning (FL) is an emerging machine learning (ML) paradigm that trains ML models across potentially millions of clients (e.g., smartphones) over hundreds of training rounds (Bonawitz et al., 2019; Huba et al., 2022). In each round, a (logically) centralized coordinator selects a subset of clients and sends the global model to these participating clients (participants). Each participant trains the model on its local data and uploads the model update to the coordinator. Before advancing to the next round, the coordinator aggregates individual client updates to update the global model. As such, federated learning circumvents systemic privacy risks of cloud-centric ML and high costs of data migration to the cloud (McMahan et al., 2017; Lai et al., 2020).

Despite sharing similar goals with traditional cloud ML (e.g., better model accuracy and less resource consumption), FL models are often trained and later deployed on clients with vast device and data heterogeneity (Wang et al., 2023; Lai et al., 2021; Singapuram et al., 2023). This leads to new systems and ML challenges to tailor models for individual clients. First, the heterogeneous capabilities of client devices, such as communication and computation, necessitate FL models with different complexities aligned to clients' hardware for better user experience (e.g., model training and inference latency). Additionally, independently generated data among clients results in heterogeneous data volumes and distributions, making it challenging to train models that fit individual client data at scale.

State-of-the-art FL solutions optimize for better model convergence (Li et al., 2020), accuracy fairness (Li et al., 2019), and faster round execution (Nguyen et al., 2022), while often focusing on the performance of a single (global) model. This may not suit individual client's device capability and data (§2). Although recent work attempted to mitigate data heterogeneity by tuning model weights within client devices (Collins et al., 2021; Ozkara et al., 2021), *they overlook client system heterogeneity*, leading to impractically large models for resource-constrained clients and vice versa. Recent multi-model approaches train models with different weights (Liu et al., 2023) or architectures for various clients (Diao et al., 2020; Hong et al., 2022; Dudziak et al., 2022), but they incur high training costs and/or struggle to identify the right model for each client.

In this paper, we propose FedTrans, a multi-model FL training framework, to automatically and efficiently train customized models for individual clients at scale (§3). FedTrans begins with a small model and progressively expands models in flight to produce well-trained models with different

[1]Computer Science, University of Illinois at Urbana-Champaign, Illinois, USA [2]Computer Science and Engineering, University of Michigan, Michigan, USA. Correspondence to: Yuxuan Zhu <yxx404@illinois.edu>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2024.

complexities, delivering high-accuracy models for diverse hardware-capable clients at a low cost.

FedTrans addresses the following fundamental requirements toward practical FL deployment (§4). First, we must cost-efficiently maximize model accuracy for individual clients subject to their hardware capabilities. Unlike existing methods that rely on pre-determined model architectures (Diao et al., 2020; Hong et al., 2022) or prohibitively expensive model exploration (He et al., 2020), FedTrans identifies accuracy bottlenecks of the training model – model architecture blocks (e.g., convolutional layers) that are incapable of fitting on clients' data – using training feedback (e.g., layer gradients). It then adaptively transforms the model architecture, such as by widening or deepening the bottleneck layers, to maximize accuracy improvement for clients with more resources. Our transformation customizes for different client groups and effectively warms up new model weights, reducing training costs. Second, we must minimize the cost of training multiple models. To this end, FedTrans identifies suitable timing for generating new models to maximize the effectiveness of warmup, and performs aggregation of model weights across models to accelerate training convergence, by exploiting model architectural similarity.

We have integrated FedTrans with FedScale[1] (Lai et al., 2022) and evaluated it across various FL tasks with real-world workloads (§5). Compared to state-of-the-art FL solutions (Diao et al., 2020; Hong et al., 2022; Wang et al., 2024), FedTrans improves model accuracy by 14% - 72% and achieves 1.6× - 20× lower training costs, while reducing manual efforts by automatically spawning models for FL clients at scale.

Overall, we make the following contributions in this paper:

- We propose a systematic multi-model framework to train performant models for individual clients at scale.
- We introduce a novel utility-based model transformation mechanism to generate, train, and assign FL models.
- We evaluate FedTrans in various real-world settings to show its resource savings and accuracy gains.

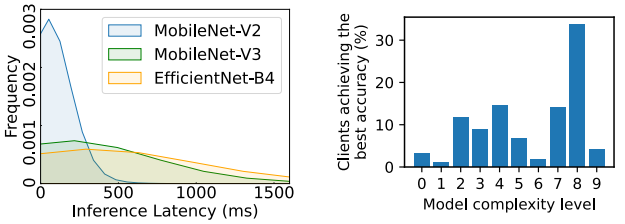
2 Motivation

We start with an introduction to the challenges in FL (§2.1), followed by the limitations of existing solutions (§2.2).

2.1 Challenges of Federated Learning

Unlike traditional cloud ML, FL trains and later deploys models on clients with vast heterogeneities in device capabilities and data. This leads to a pressing need for minimizing costs and maximizing model accuracy for individual clients.

[1]https://github.com/SymbioticLab/FedScale



(a) The heterogeneous device capabilities require models with different complexities.

(b) No single model achieves the best accuracy for the majority of clients.

Figure 1. Client system heterogeneity and accuracy variance call for different models for individual clients.

System heterogeneity requires models with different complexities. FL client devices often have diverse hardware capabilities, requiring models of different levels of complexity for practical training and deployment. We analyze the inference latency of three on-device models across over 700 smartphones using the *AI Benchmark* (Ignatov et al., 2019). Figure 1a shows that end-user devices impose clear model complexity requirements to meet latency constraints. For example, users with resource-constrained devices can hardly endure seconds of inference latency from models like MobileNet-V3 in real deployment (Lv et al., 2022). Hence, different client groups want models of differing complexities. Furthermore, clients with the same latency requirements can have multiple architecture choices, as indicated by the distribution overlap (e.g., MobileNet-V3 and EfficientNet-B4). Consequently, FL developers must carefully design model architectures for individual clients to maximize accuracy, implying significant human effort.

Client accuracy variance implies no one-size-fits-all model. The need for training multiple models is further amplified by the client accuracy variance in single-model FL. To exemplify this challenge, we experiment with seven models of varying complexities selected from the NASBench201 benchmark (Dong & Yang, 2019). Here, model complexity is quantified by the number of multiply-accumulate (MACs) operations, with each increase doubling this count. We train them on the Federated-EMNIST (FEMNIST) dataset across 3400 clients. Figure 1b reports the percentage of clients that achieve the best accuracy on each model. We observe that no single model achieves the best accuracy for the majority of clients. The model that achieves the highest accuracy varies for different clients, highlighting the need for tailoring models to each client's specific requirements.

2.2 Limitations of Existing Solutions

Prior FL work (Li et al., 2020; 2019; Nguyen et al., 2022) has primarily focused on improving the performance of a single global model, with recent strides towards multi-

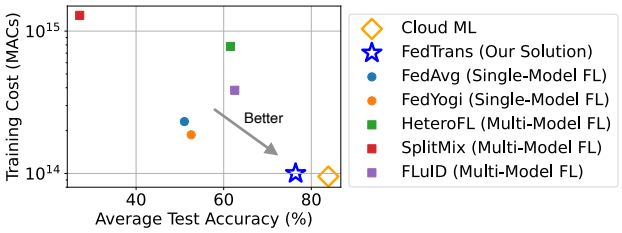


Figure 2. Existing solutions are suboptimal for FL clients.

model federated learning (Dudziak et al., 2022). However, existing solutions exhibit shortcomings in FL training costs and model accuracy.

Huge training costs and laboring effort. Recent FL advances design and train heterogeneous model architectures for different clients. However, they often require manually determining the model set used for training in advance (Hong et al., 2022; Alam et al., 2022; Yao et al., 2021), a labor-intensive but ineffective process without extensive exploration. Moreover, training multiple models concurrently and/or conducting continuous model testing for informed client-model assignment can raise huge costs. Figure 2 shows that even state-of-the-art multi-model solutions (Hong et al., 2022; Diao et al., 2020) and dropout-based solutions (Wang et al., 2024) result in a cost increase of orders of magnitude compared to training a single global model. We use multiply-accumulate operations (MACs) to measure costs (Bannink et al., 2021).

Inferior model accuracy. Training a single global model may reduce costs, but it struggles to accommodate individual client data (Figure 2). Conversely, existing multi-model FL solutions, particularly those using pre-determined models, struggle to pinpoint accuracy bottlenecks in model architectures and/or decide appropriate model assignments to clients at scale. When we consider the cloud ML performance as a hypothetical performance upper bound, where the data is centralized and shuffled to be homogeneous, Figure 2 shows that existing solutions are far from achieving this upper bound.

3 FedTrans Overview

We present FedTrans, a multi-model FL training framework, to efficiently produce and train high-accuracy, hardware-compatible models for individual clients at scale. We next provide an overview of how FedTrans fits in the FL lifecycle.

System Components At its core, FedTrans relies on a model abstraction, *Cell*, to transform the initial global model into multiple models during FL training. The *Cell* is the minimum component of the model architecture (e.g., a convolution layer or a ResNet block), on which FedTrans performs model transformation (e.g., widen or deepen a *Cell*). FedTrans enables online model transformation via three system components:

- *Model Transformer*: It captures accuracy bottlenecks in the model architecture, whereby it performs information-preserving transformations on the model architecture to generate new models and warm up model weights.
- *Client Manager*: It explores the utility of each model to individual clients over different training rounds, considering potential accuracy gains and system constraints. Then, it assigns the right model to each client.
- *Model Aggregator*: It manages the training of multiple models and performs soft aggregation on model weights across models to accelerate their convergence.

FedTrans Lifecycle Figure 3 illustrates the lifecycle of FedTrans. FL developers initialize FL training with a small model. In each training round, ① Model Transformer leverages training feedback from previous rounds (e.g., gradient and loss) to potentially transform (e.g., widen or deepen) a specific *Cell* of the model. If transformation occurs, Model Transformer will generate a new model with the current largest model weights to reduce training costs. ② For each participant, Client Manager assigns an appropriate model among all currently available models, in terms of their utility (e.g., training loss) and system constraints. Then participants download their models and start local training. ③ When receiving client updates, Model Aggregator aggregates model weights in a weight-sharing manner to accelerate convergence.

FedTrans iterates these phases in each round until exhausting the training budget, or the model architecture complexity reaches the maximum supported by any participant and all models converge. Note that conventional single model training is a special case of this lifecycle, where FedTrans does not create any new model.

4 FedTrans Design

In this section, we introduce how Model Transformer generates a suite of performant models at low training costs (§4.1), how Client Manager assigns individual clients with the right model (§4.2), and how Model Aggregator accelerates the co-training of model suites (§4.3).

4.1 Model Transformer

Generating the right model for FL clients is challenging due to the interplay among model accuracy, client system constraints, and training costs. Intuitively, using a large model often implies better accuracy, but it incurs high training costs, and clients may not be able to run such large

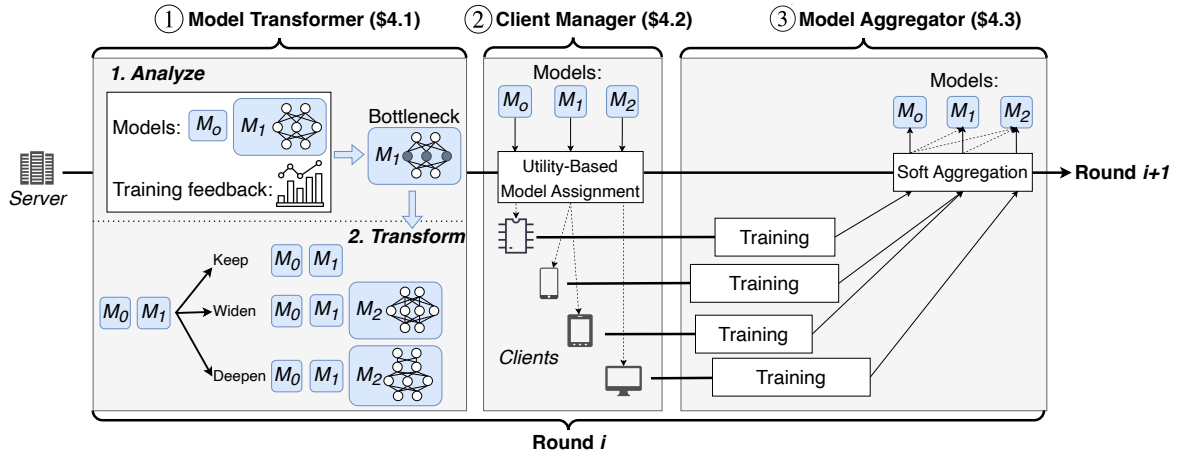


Figure 3. FedTrans architecture and its lifecycle in FL deployment.

models (Wang et al., 2023). It is tempting to model it as a Neural Architecture Search (NAS) problem in traditional cloud ML, such as by training a super-network (Liu et al., 2019). However, doing so in FL is suboptimal and even impractical due to the expensive planet-scale FL training (e.g., training a super-network) and massive data heterogeneity.

To strike a balance between low costs and high model accuracy, Model Transformer generates new models with different system requirements by transforming, thus improving model architectures during FL training. The key insight is that (1) models achieving state-of-the-art performance are often built with well-established blocks (e.g., ResNet or Transformer blocks), so we can reduce the exploration space of model architectures by transforming at the block (*Cell*) level; and (2) model transformation, which changes the width and depth of bottleneck *Cell*s to generate a new model with inherited model weights (Lai et al., 2023), can reduce training costs by warming up new models' training.

However, we need to tackle the following challenges:

- When to perform a model transformation?
- At which *Cell* of the model architecture to transform?
- How to transform (e.g., widen/deepen) the selected *Cell*?

Identifying the right time to transform When to transform introduces a trade-off between the maturity of the current model and the waiting time spent on training new models. Transforming the current model too early yields limited warmup benefits due to the current model's low accuracy, resulting in longer training of the new model. On the other hand, transforming too late, such as when model training converges, results in longer waiting times to start the new model and only marginal warm-up benefits due to the sublinear convergence speed of ML training.

To find the sweet spot, we refer to the popular elbow method that picks the elbow of the curve to maximize benefits (Shi et al., 2021). Here, we define the degree of convergence (DoC) to measure the slope of the moving training loss. Equation 1 defines the DoC at round i by averaging γ consecutive loss (L) slopes, where each loss slope is calculated with a step of δ:

$$\text{DoC} = \frac{1}{\gamma}\sum_{i=1}^{\gamma}\frac{L(i-\delta)-L(i)}{\delta} \tag{1}$$

FedTrans initiates the transformation when DoC is less than a certain threshold, i.e., reaching the elbow of the curve. Intuitively, a larger threshold will make FedTrans transform models more frequently, while a larger step size tolerates more oscillations of the training loss. Theoretically, our DoC design draws from the positive link between loss sharpness and model generalizability (Li et al., 2018; Dziugaite & Roy, 2017). A smooth loss curve indicates robust generalizability and guides optimal model transformation timing. We also empirically evaluate the effectiveness of our DoC design (§5.4).

Picking the right model *Cell* to transform Once starting the transformation, we need to decide to transform which model *Cell* can unblock model accuracy with the limited information in practical deployment, where only the training loss and gradients are available most of the time. In complement to existing privacy-preserving mechanisms, FedTrans solely utilizes aggregate gradients, not the gradients of individual clients.

Model Transformer selects the *Cells* with larger gradient norms to transform. Intuitively, a large gradient implies that *Cell* is under great dynamics and harder to fit, especially given that transformation will not be activated unless the model starts to converge, so we should transform this *Cell* to

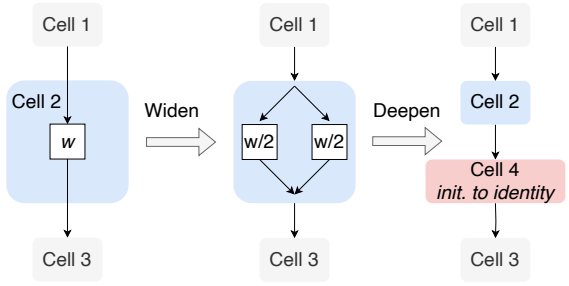


Figure 4. Model transformation on a convolution *Cell*.

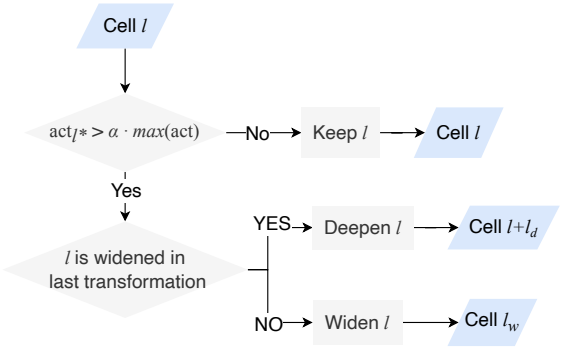


Figure 5. Control flow of the transformation for a *Cell* l.

unblock model convergence (performance). In theory, our design is motivated by transfer learning, where layers that are actively being updated during training contribute more to the bottleneck of model convergence (Long et al., 2015; Wang et al., 2022), indicating the incapability of capturing the data characteristics of certain FL clients.

FedTrans relies on the distribution of *Cell* activeness – measured by the gradient norm $\frac{\|\nabla w_l\|}{\|w_l\|}$ in round update – to decide which *Cells* to transform. We normalize the gradient norm by the weight norm to mitigate the bias in selecting cells due to gradient vanishing. Since a model can have multiple *Cells* blocking the performance, we transform the *Cells* whose activeness exceeds α times of the largest activeness among all *Cells*. By default, α is set to 0.9. We empirically evaluate the effectiveness of α in our ablation study (§5.4).

Widening or Deepening After selecting which *Cells* to transform, we may widen or deepen them. Taking Figure 4 as an example, we can widen a *Cell* by increasing the number of neurons or deepen it by inserting an identity *Cell*, which directly passes the input of its predecessor to the successor. However, doing so is non-trivial. First, we need to decide which operation to perform and enable it with little overhead and loss of its parent model's weights. Also, we should carefully control the degree of transformation to avoid growing too fast and missing the right complexity for clients.

Inspired by the concept of compound scaling in model architecture design (Tan & Le, 2019), which emphasizes the importance of balanced width and depth for efficiency, Model Transformer alternates between widening and deepening the *Cell*. Figure 5 shows the overall procedure to perform architecture transformation, wherein each operation will widen the selected *Cell* by a factor or insert (deepen) a certain number of *Cell* (s). By default, FedTrans widens a *Cell* by two or inserts one *Cell* at a time. After these transformations, FedTrans generates a new training model with the potential for better accuracy upon convergence. We empirically show that our compound scaling design achieves better performance than its counterparts (§5.4).

After widening or deepening the *Cell*, Model Transformer performs function-preserving weight transformation to inherit model weights. When deepening a *Cell*, we initialize the newly inserted cell's weights as an identity tensor. When widening a *Cell*, we randomly select columns from the pre-expanded *Cell*'s weights to fill the widened *Cell* and divide each column by the times it is selected. This transformation, theoretically (Chen et al., 2015; Cai et al., 2019; Lai et al., 2023), maintains the parent model's information, ensuring the same tensor output for many common tensor operators such as fully connected and convolutional layers. As a result, it allows for the transfer of model weights to warm up new models.

4.2 Client Manager

We have discussed how to generate models *in the cloud*, to account for clients with heterogeneous data and system preferences. The following major challenge is to distribute and train these models for individual clients *at the edge*. Algorithm 1 outlines how FedTrans enables this model transformation across numerous clients through the coordination of Client Manager (Line 5) and Model Aggregator (Line 12). Next, we introduce how the Client Manager selects the right model among all generated models for each client.

Utility-Based Model Assignment The design of the model assignment should follow two criteria. First, each client should be assigned a model whose complexity does not exceed the client's hardware capability to respect their system constraints. Here, we consider the models whose number of multiply-accumulate operations (MACs) is fewer than the participant's hardware capability, called compatible models. Note that our solution can easily support other definitions of "compatible models." Second, among compatible models, we should assign the model that is most suitable to that client's data characteristics. Moreover, the design should only leverage the information available in today's FL deployment to respect privacy. For example, we cannot access the client data.

Algorithm 1: Pseudo-code of FedTrans runtime.

Parameter : *Cell* selection threshold α, DoC threshold β
Input : initial model with k_0 cells $M_0 = [l_1^0, \ldots, l_{k_0}^0]$, initial model weights $W_0 = [w_1^0, \ldots, w_{k_0}^0]$, registered client list $\mathcal{C}$, client capacity list T, number of clients per round N
Output : a list of models trained $\mathcal{M}$

$\mathcal{M} \leftarrow [M_0], \mathcal{W} \leftarrow [W_0]$ // init. models & weights
$U_0 \leftarrow [0, \ldots, 0], \mathcal{U} \leftarrow [U_0]$ // init. utilities
for *each FL training round* $i = 0, 1, 2, \ldots$ **do**
  $\widetilde{\mathcal{C}} \leftarrow \text{Select}(\mathcal{C}, N)$
  // Assigning models to clients (§4.2)
  for *each selected client* $c \in \widetilde{\mathcal{C}}$ ***parallelly*** **do**
    $K \leftarrow \|\{M \in \mathcal{M} | \text{MAC}(M) \leq T_c\}\|$
    $n \leftarrow \text{Sample}(\mathcal{U}_c, [0, \ldots, K-1])$
    // Clients return weights, gradients, and loss
    $W_{n,c}, G_{n,c}, L_{n,c} \leftarrow \text{ClientTrain}(\mathcal{W}[n], c)$
  $\mathcal{U} \leftarrow \text{UpdateUtility}(\mathcal{U}, L)$ // Eq. (4)
  // Inter-model weights aggregation (§4.3)
  $\mathcal{W} \leftarrow \text{UpdateWeight}(W)$
  // Model transformation and warmup (§4.1)
  $\text{DoC} \leftarrow \text{UpdateDoC}(L)$ // Eq. (1)
  if $\text{DoC} \leq \beta$ **then**
    $M^* \leftarrow$ copy the parent model's weights
    $U^* \leftarrow$ copy the parent model's utility
    for *each Cell* $l^* \in M^*$ **do**
      $\text{act}_{l^*} \leftarrow \text{UpdateAct}(G, W)$
      $\text{TransformCell}(l^*, \text{act}_{l^*}, \alpha)$
    add new model M^* to $\mathcal{M}$

Here, we leverage the training loss of models to approximate their data-aware affinity to individual clients. Intuitively, a small training loss means potentially better accuracy of that model on client data. However, to avoid prematurely overfitting to a specific model, we maintain a loss-based utility list for each registered participant c with K compatible models: $U^c = [U_0^c, \ldots, U_{K-1}^c]$. When the client participates in FL training, the coordinator probabilistically samples a model following the distribution of model loss, as specified in Eq. 2. Here, we take the exponential format of training loss for its smoothness and normalize the probability by the sum of all models' training loss on that client (Eq. 3):

$$n = \text{Sample}(p^c, [0, \ldots, K-1]) \tag{2}$$

$$p_k^c = \frac{\exp[U_k^c]}{\sum_{k=0}^{K-1} \exp[U_k^c]}, k = 0, \ldots, K-1 \tag{3}$$

The utilities of each client on each model are updated using the model performance (e.g. training loss). This soft model assignment scheme encourages a client to explore other models when its training performance is bad (i.e., high training loss), while encouraging a client to stick to the current model when its training performance is good.

Joint Utility Learning However, new models can be generated over time, and not all models are trained on that client all the time. Updating the utility U_k^c of each model can be sporadic, and the previous utility can become stale. To accelerate the exploration of better model assignment, the Client Manager jointly updates the utility of compatible models based on their architectural similarity. This is because similar models tend to exhibit similar model accuracy. As such, after the completion of that client, the Client Manager updates the utility lists of all the compatible models as follows:

$$U_k^c = U_k^c - L_k^c \cdot \text{sim}(M_k, M^*) \tag{4}$$

where M_k is the k-th compatible model, M^* is the model assigned to client c in the last round, L_k^c is the standardized training loss of client c in the last round, and $sim(\cdot, \cdot) \in [0, 1]$ calculates the similarity between two models. We take the subtractive format as a high training loss means a lower utility. As such, the model with similar architectures would borrow more utility information from the current model.

Here, we measure the similarity of two model architectures in terms of the *Cell*-wise number of parameters that we can transform, which aligns with our design of model transformation. For each *Cell* l, we measure its similarity score, $mc(l)$, between the new model and the model it transformed from (i.e., parent model). Specifically, (a) if l is inherited from M without change, $mc = 1$ (i.e.a full matching degree); (b) if l is widened from the *Cell* l' of M, $mc = \#\text{param}(l')/\#\text{param}(l)$ (i.e., the portion of inherited model weights); (c) if l is inserted to M in the deepen operation, then $mc = 0$ as it does not inherit any model weights; Otherwise, (d) $mc = -1$ because it loses the weights of its parent model. Finally, we get the model similarity, $\text{sim}(M_k, M^*)$, by cumulating the similarity of all *Cells*.

4.3 Model Aggregator

After assigning models to clients, we concurrently train (co-train) multiple models. Minimizing their total training costs becomes the crux. Moreover, compared to conventional single-model training, each model trains on fewer clients due to the unbalanced model assignment and system compatibility, which can slow down training convergence.

Historically, sharing weights among similar model architectures has accelerated model convergence (Diao et al., 2020; Mei et al., 2022; Pham et al., 2018). While this implies

Breakdown	Dataset	Avg. Accu. (%)
FedTrans	FEMNIST	75.5
FedTrans (l2s)	FEMNIST	60.9
FedTrans	Cifar10	77.5
FedTrans (l2s)	Cifar10	54.1

Table 1. Model accuracy with or without weights sharing from large models to small models

an opportunity to leverage the similarity of models, aggregating multi-model weights is non-trivial. This is because not all models contribute equally to individual model aggregation. First, FedTrans generates large models when the small model is converging, meaning that the small model is under fine-tuning while the large model is still under-trained. Using large models to update small models can lead to big noise in the convergence of small models, impeding their convergence. As shown in Table 1, we compare the accuracy performance of FedTrans on FEMNIST and Cifar10 datasets, with and without enabling weights sharing from large models to small models (l2s). Disabling l2s significantly improves final model accuracy.

Second, when leveraging the weights of different models to update a model, we must consider their architectural similarity and real-time convergence.

By taking these two insights into account, we develop a soft model aggregation mechanism that performs a *weighted* average over models' weights while considering model architectural similarity:

$$w_j = \frac{\sum_{i=1}^{j} \eta^{\mathbb{1}(i \neq j) \times t} \text{sim}(M_i, M_j) \cdot w_i}{\sum_{i=1}^{j} \text{sim}(M_i, M_j)} \quad (5)$$

$\eta^{\mathbb{1}(k \neq j) \times t}$ is the decaying factor in round t, while w_j and w_i denote the weights of model M_j and M_i, respectively. We crop w_i if necessary to fit the shape of w_j as in HeteroFL (Diao et al., 2020).

When updating model weights using Eq. (5) in each round, it combines updates from both that model's clients and similar models based on their architectural similarity. Moreover, as the model converges over rounds, η progressively reduces the impact of other models to mitigate noise toward its training convergence. We empirically show the effectiveness of our soft aggregation design using realistic datasets (§5.3).

5 Evaluation

We evaluate FedTrans on four CV and NLP datasets across four models. We summarize the results as follows:

- FedTrans improves average model accuracy by 13.78% - 72.15% against existing multi-model FL frameworks, while reducing training costs by 1.6× - 20.0× (§5.2).
- FedTrans reduces manual efforts by automatically finding better models for clients, wherein each component is effective for the overall performance (§5.3).
- FedTrans improves performance over a wide range of settings and outperforms its design counterparts (§5.4).

5.1 Methodology

Experimental Setup We conducted experiments using FedScale (Lai et al., 2022), a widely used FL benchmarking platform, on a cluster of 15 NVIDIA V100 GPUs. The platform produces realistic FL client system speed and client data. For each training round, we select 100 clients, each using a batch size of 10 and performing 20 local steps. More details are available in Appendix A.1. The initial model's complexity corresponds to the client with the lowest computation and communication capacities, while the maximum model's complexity aligns with the client possessing the highest resource capacities. We sample client hardware capacities from FedScale, which includes the traces of 500k real-world mobile devices. The disparity between the most capable and least capable devices exceeds 29×.

Datasets and Models We first experiment with *CIFAR-10* image classification tasks and follow existing works (Diao et al., 2020) to partition them into non-IID datasets with 100 clients. Then, we run three real-world FL datasets of different scales and use their realistic partitions, which are widely used in FL benchmarking (Liu et al., 2023; Lai et al., 2022; Li et al., 2020):

- *Speech Command*: a small-scale Google speech dataset of 2,618 clients (Warden, 2018). We use ResNet18 as the initial model to recognize 35-category commands.
- *F-EMNIST*: a middle-scale image classification dataset of 3,400 clients (Cohen et al., 2017). We choose the smallest model in NAS- Bench201 as the initial model.
- *OpenImage*: a large-scale image classification dataset of 14,477 clients (ope), with 1.5 million images spanning 600 categories. We start with ResNet18.

More detailed setups are available in Appendix §A.1.

Parameters We set default values for key FedTrans parameters: layer activeness threshold, α, is set to 0.9; the number of consecutive slopes to compute DoC, γ, is 10; and the DoC threshold for β transformation is 0.003. We show the robustness of FedTrans benefits across these parameters in our ablation study (§5.4). Other FL hyperparameters are available in Appendix A.1.

	Method	Accu. (%)		IQR (%)	Cost (PMACs)		Storage (MB)	Network (MB)
CIFAR-10	FedTrans	78.29		5.25	0.86		6.8	1368.4
	FLuID	59.81	↑ 18.48	3.00	1.50	↓ 1.7×	71.7	1433.4
	HeteroFL	64.51	↑ 13.78	7.00	4.31	↓ 5.0×	45.7	1828.4
	SplitMix	51.36	↑ 26.93	6.23	4.49	↓ 5.2×	35.4	7089.3
FEMNIST	FedTrans	76.42		13.23	0.10		0.04	8.9
	FLuID	62.52	↑ 13.90	13.63	0.38	↓ 3.8×	0.3	64.6
	HeteroFL	61.54	↑ 14.88	13.50	0.78	↓ 7.8×	1.0	209.0
	SplitMix	27.13	↑ 49.29	10.60	1.29	↓ 12.9×	1.3	9392.5
Speech	FedTrans	91.75		4.75	0.10		0.3	56.4
	FLuID	76.50	↑ 15.25	43.65	0.26	↓ 2.6×	0.6	121.5
	HeteroFL	73.69	↑ 18.06	40.00	1.49	↓ 14.9×	1.0	197.5
	SplitMix	19.60	↑ 72.15	8.89	0.60	↓ 6.0×	1.3	5736.8
OpenImage	FedTrans	61.86		28.56	47.39		10.6	2118.9
	FLuID	32.87	↑ 28.99	51.61	76.88	↓ 1.6×	35.5	4926.92
	HeteroFL	24.53	↑ 37.33	36.00	799.52	↓ 16.9×	92.0	6187.7
	SplitMix	35.44	↑ 26.24	21.15	950.38	↓ 20.1×	38.7	2468.0

Table 2. Detailed performance comparison.

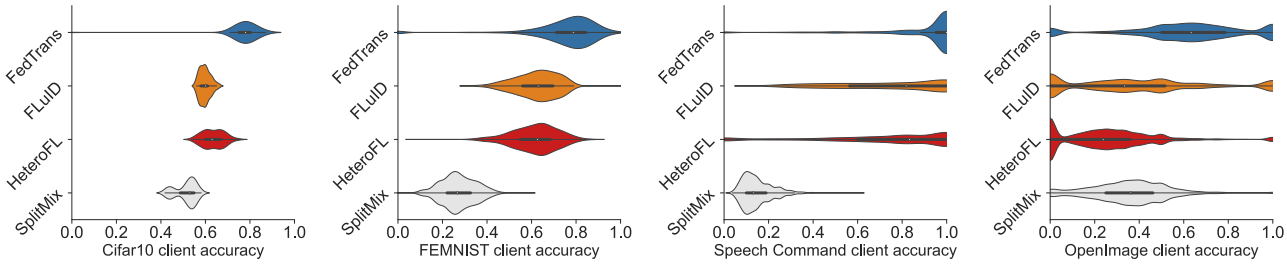


Figure 6. FedTrans improves individual client model accuracy over state-of-the-art multi-model federated learning on four datasets.

Baselines We compare FedTrans with state-of-the-art solutions for multi-model federated learning: HeteroFL (Diao et al., 2020) and SplitMix (Hong et al., 2022), and a dropout-based FL solution, FLuID (Wang et al., 2024).

Metrics We care about *model accuracy* and the *total training costs* to achieve them. We evaluate each client only on its compatible models and assign it the model with the highest utility. We report the average accuracy of all clients. We measure training costs using the widely used total number of MAC operations performed by all clients (Han et al., 2015; Bannink et al., 2021; Rapp et al., 2022).

For each experiment, we report the mean value over 3 runs.

5.2 End-to-End Performance

Table 2 summarizes the key accuracy performance and training costs on all datasets.

FedTrans improves model accuracy over state-of-the-art solutions. FedTrans improves the average model accuracy over HeteroFL by 13% or more (Table 2). The box plots in Figure 6 zoom into the accuracy distribution of individual clients. FedTrans clearly improves model accuracy for individual clients and achieves the best average accuracy on all four datasets. Note that HeteroFL benefits the largest model among its trained models the most, yet results in lower accuracy for smaller models. As such, HeteroFL has extremely low accuracy for clients with low hardware capacity.

FedTrans reduces training costs Table 2 also reports that FedTrans reduces the total training costs by more than 4×. We also notice that FedTrans has the lowest storage footprint and the network transmission volume. Figure 7 further validates FedTrans's benefits on cost-to-accuracy performance: FedTrans incurs the lowest MAC costs toward achieving the same model accuracy. This is because FedTrans begins with smaller models and judiciously introduces additional ones.

FedTrans improves existing optimizations for FL training. We implemented FedTrans as a complementary component to the popular optimization algorithms `FedProx`

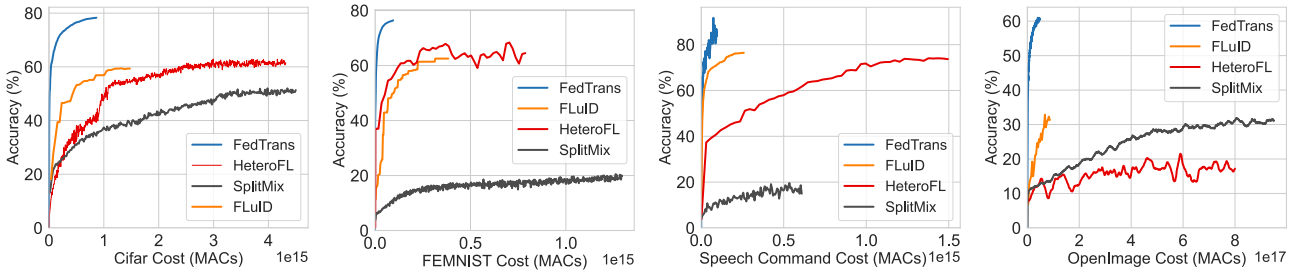


Figure 7. FedTrans reduces training costs over state-of-the-art multi-model federated learning on four datasets.

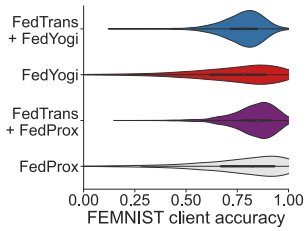


Figure 8. FedTrans complements existing FL optimizations and improves their performance.

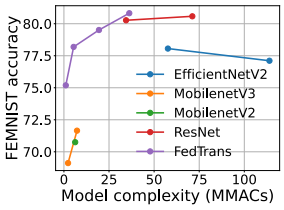


Figure 9. FedTrans finds better model architectures for individual clients (FEMNIST dataset).

Breakdown	Accu. (%)	Costs (MACs)
FedTrans	**76.42**	**9.68** $\times 10^{14}$
FedTrans- l	73.44	9.17 $\times 10^{14}$
FedTrans- ls	65.16	9.75 $\times 10^{14}$
FedTrans- lsw	64.40	24.73 $\times 10^{14}$
FedTrans- lswd	55.90	3.37 $\times 10^{14}$

Table 3. Performance breakdown. 'l' means layer selection, 's' means soft aggregation, 'w' means warm up, and 'd' means decayed weight sharing.

and `FedYogi`. We run FedTrans with `FedProx` or `FedYogi` on FEMNIST dataset with the NASBench201 base model as the initial model. We run `FedProx` and `FedYogi` solely with the middle-sized model generated by FedTrans. We report the average test accuracy and training cost as the metrics. Figure 8 shows that FedTrans can improve `FedProx` and `FedYogi`, achieving higher average accuracy with the same training cost.

FedTrans models outperforms state-of-the-art models for clients. We compare FedTrans-generated models with state-of-the-art models on the FEMNIST dataset: EfficientNet-V2, MobileNet-V2, MobileNet-V3, ResNet-18, and ResNet-34. We sampled four of our transformed architectures from FedTrans. Note that all FedTrans models are transformed from the base model of NASBench201, and we measured their average accuracy. Figure 9 shows that FedTrans-generated models achieve a better tradeoff between MACs and accuracy over today's advanced models, owing to our heterogeneity-aware model assignment.

5.3 Performance Breakdown

We next analyze the impact of each component of FedTrans on the final performance, in terms of average accuracy and training costs. We have three major components in our system: transformation-based model expansion, gradient-based layer selection, and soft aggregation across models.

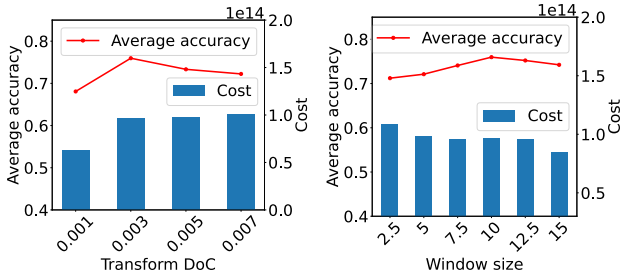
As shown in Table 3, all components contribute to the overall performance, indicating the effectiveness and necessity of all FedTrans components. By replacing the gradient-based layer selection with the random layer selection, the accuracy witnessed an around 3% drop since random selection does not always pick the best layer to expand. If we further disable the weight sharing among different models, the accuracy drops by 8%. By removing the warmup transformation and re-initializing the weights of larger models, the accuracy drops, and training cost increases by 1.6×. Finally, model accuracy drops by 8% after removing the decay factor in soft aggregation.

5.4 Ablation Study

We next analyze the impact of parameters and data heterogeneity on FedTrans performance. We present the results of the FEMNIST dataset with similar trends observed on other datasets.

Threshold of DoC **to transform** (β). Figure 10a shows that as β increases, a model is more easily considered to be ready for transformation, leading to more models transformed in the whole training process and thus higher training costs. Moreover, the accuracy is higher as β increases initially because FedTrans has more models capturing the data characteristics of clients comprehensively. However, the accuracy significantly drops when β is too high because the data samples per model are too few.

Number of consecutive slops to calculate DoC (γ). Figure 10b shows that increasing γ generally increases the difficulty of reaching a certain DoC because we are taking more con-



(a) Degree of convergence (β). (b) #slops for DoC (γ).

Figure 10. FedTrans achieves robust performance improvement by picking the right time to transform.

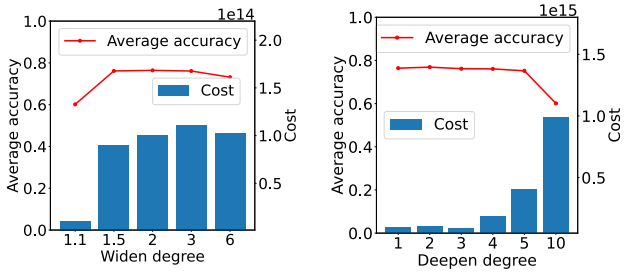


Figure 11. FedTrans is robust to different transformation degrees.

secutive loss slopes into the average. Therefore, increasing γ significantly decreases the training cost before the model is transformed fewer times. On the other hand, increasing γ can help improve the final accuracy as more data samples are used to train one model, increasing the generality of the model. However, if γ becomes too large, FedTrans has too few models to capture the heterogeneity of clients' data.

Impact of Widening and Deepening Degrees Figure 11 shows that the average test accuracy and training cost are robust to a wide range of widening or deepening degrees. Intuitively, when the widening or deepening degree is slightly larger, FedTrans trains fewer large models but earlier, which does not change the training cost. Meanwhile, although larger degrees decrease the total number of models transformed, each model is more aggressively optimized and gains more capability.

Method	Accu. (%)	Cost (MACs)
FedTrans + FedAvg	76.5	3.8×10^{11}
FedAvg	71.5	1.09×10^{13}

Table 4. FedTrans improves for ViT models (FEMNIST dataset).

FedTrans optimizes for a wide range of models. Prior work is mostly limited to convolution networks (Diao et al., 2020; Hong et al., 2022), while FedTrans is generalizable to many model architectures, including ViT (Dosovitskiy et al., 2020). Table 4 shows that FedTrans outperforms existing FL optimization on ViT models with a 5% improvement of accuracy and saving costs by orders of magnitude.

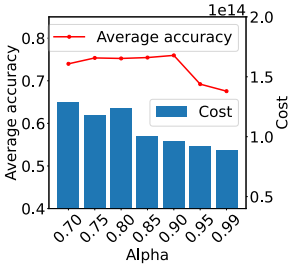


Figure 12. FedTrans improves performance by picking the right *Cell* to transform.

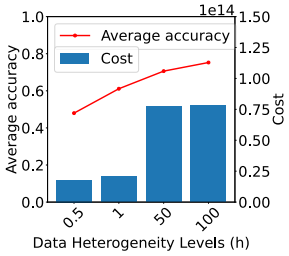


Figure 13. Performance of FedTrans under different levels of data heterogeneities.

Threshold of layer activeness to transform (α). Figure 12 shows that as α increases, fewer layers are selected to expand, making the expanded model smaller and thus decreasing the training cost. On the other hand, the accuracy drops after α reaches 0.9, indicating that too few layers are selected to expand the capability of the model sufficiently.

Impact of Data heterogeneity Figure 13 shows the change in average accuracy and training cost while we tune the synthetic data heterogeneity of the FEMNIST dataset. Similar to the prior work (Diao et al., 2020; Alam et al., 2022), we synthesize different data heterogeneity levels by controlling the label distribution with a Dirichlet distribution and parameter h. The lower h, the higher the heterogeneity.

Under low data heterogeneity, FedTrans converges with better accuracy using more rounds. Therefore, the training costs are seemingly high (Fig. 13). Meanwhile, the performance of FedTrans diminishes under high data heterogeneity, which underscores the need for future research to enhance the training algorithms of multi-model FL in highly heterogeneous settings.

6 Related Work

System optimizations for FL training Several studies propose to optimize the FL training from a system scheduling perspective. (Lai et al., 2021) prioritizes clients with good data quality and hardware capability to improve efficiency, while (Li et al., 2022a) further investigates the data and system heterogeneity of selected clients to optimize the profiling. Besides, clustering-based optimization is proposed to mitigate the data heterogeneity by only aggregating the local models of clients with similar data distribution (Liu et al., 2023; Duan et al., 2021), while (Nguyen et al., 2022) proposes a scalable asynchronous training scheduling algorithm to address straggler issues in synchronous FL.

Multi-model FL Training customized models for clients at scale is challenging due to the data and system heterogeneity, as well as execution costs. Weight-sharing mechanisms have been proposed to train multiple models simultaneously to save training costs, but their exploration of model architectures is either static (Diao et al., 2020) or tailors only the width of the model (Li et al., 2021a; Horvath et al., 2021; Mei et al., 2022), resulting in suboptimal performance. Secondly, previous work on applying NAS algorithms to FL settings either uses a super-model reduction method, which introduces an expensive search phase (He et al., 2020), or randomly samples architectures, which are usually of low quality (Dudziak et al., 2022). Recent work on personalized FL uses multi-task learning (Marfoq et al., 2021; Li et al., 2021b), meta learning (Acar et al., 2021; Fallah et al., 2020), and other ML-based techniques (Collins et al., 2021; Ozkara et al., 2021) to optimize the accuracy performance, while neglecting the system heterogeneity.

Model Transformation Prior work proposes techniques of neural network morphism (Chen et al., 2015; Wei et al., 2016; Cai et al., 2019) to transform a neural network to a larger one, preserving the complete functionality. Such techniques have been applied to reinforcement learning (Czarnecki et al., 2018), transfer learning (Li et al., 2022b), and neural architecture search algorithms (Cai et al., 2018; Chen et al., 2020) to save the costs of training new and larger models. Recent work in cloud computing leverages the model transformation techniques to accelerate training on the cloud without sacrificing the accuracy (Lai et al., 2023). FedTrans further applies model transformation to distributed neural network training in a heterogeneous and federated setting.

7 Conclusion

This paper introduces FedTrans, a novel multi-model FL training framework. FedTrans employs *Cell*-wise model transformation to efficiently generate models for clients with heterogenous data and system capabilities. It uses an adaptive model assignment mechanism to explore the right model for individual clients during training, and performs inter-model aggregation to minimize the costs of training multiple models. Our evaluations using real-world datasets show that FedTrans significantly improves model accuracy while reducing training costs.

Acknowledgement

We thank the anonymous reviewers, our shepherd Kevin Hsieh, and SymbioticLab members for their invaluable comments and suggestions, which greatly improved the quality of the paper. We are grateful to the CloudLab team for providing computing resources for FedTrans experiments. Additionally, we thank Xingran Shen and Xiang Sheng for their insight in developing FedTrans design. This work was supported in part by NSF grant CNS-2106184 and a grant from Cisco.

References

Google Open Images Dataset. https://storage.googleapis.com/openimages/web/index.html.

Acar, D. A. E., Zhao, Y., Zhu, R., Matas, R., Mattina, M., Whatmough, P., and Saligrama, V. Debiasing model updates for improving personalized federated training. In *ICML*, 2021.

Alam, S., Liu, L., Yan, M., and Zhang, M. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. In *NeurIPS*, 2022.

Bannink, T., Hillier, A., Geiger, L., de Bruin, T., Overweel, L., Neeven, J., and Helwegen, K. Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks. In *MLSys*, 2021.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. Efficient architecture search by network transformation. In *AAAI*, 2018.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

Chen, T., Goodfellow, I., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. *arxiv.org/abs/1511.05641*, 2015.

Chen, X., Wang, R., Cheng, M., Tang, X., and Hsieh, C.-J. Drnas: Dirichlet neural architecture search. In *ICLR*, 2020.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *arxiv.org/abs/1702.05373*, 2017.

Collins, L., Hassani, H., Mokhtari, A., and Shakkottai, S. Exploiting shared representations for personalized federated learning. In *ICML*, 2021.

Czarnecki, W., Jayakumar, S., Jaderberg, M., Hasenclever, L., Teh, Y. W., Heess, N., Osindero, S., and Pascanu, R. Mix & match agent curricula for reinforcement learning. In *ICML*, 2018.

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

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICML*, 2019.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2020.

Duan, M., Liu, D., Ji, X., Wu, Y., Liang, L., Chen, X., Tan, Y., and Ren, A. Flexible clustered federated learning for client-level data distribution shift. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

Dudziak, L., Laskaridis, S., and Fernandez-Marques, J. Fedoras: Federated architecture search under system heterogeneity. *arxiv.org/abs/2206.11239*, 2022.

Dziugaite, G. K. and Roy, D. M. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arxiv.org/abs/1703.11008*, 2017.

Fallah, A., Mokhtari, A., and Ozdaglar, A. Personalized federated learning: A meta-learning approach. *arxiv.org/abs/2002.07948*, 2020.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

He, C., Annavaram, M., and Avestimehr, S. Towards non-iid and invisible data with fednas: federated deep learning via neural architecture search. *arxiv.org/abs/2004.08546*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Hong, J., Wang, H., Wang, Z., and Zhou, J. Efficient split-mix federated learning for on-demand and in-situ customization. In *ICLR*, 2022.

Horvath, S., Laskaridis, S., Almeida, M., Leontiadis, I., Venieris, S., and Lane, N. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. In *NeurIPS*, 2021.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *ICCV*, 2019.

Huba, D., Nguyen, J., Malik, K., Zhu, R., Rabbat, M., Yousefpour, A., Wu, C.-J., Zhan, H., Ustinov, P., Srinivas, H., et al. Papaya: Practical, private, and scalable federated learning. In *MLSys*, 2022.

Ignatov, A., Timofte, R., Kulik, A., Yang, S., Wang, K., Baum, F., Wu, M., Xu, L., and Van Gool, L. Ai benchmark: All about deep learning on smartphones in 2019. In *ICCV Workshop*, 2019.

Lai, F., You, J., Zhu, X., Madhyastha, H. V., and Chowdhury, M. Sol: Fast distributed computation over slow networks. In *NSDI*, 2020.

Lai, F., Zhu, X., Madhyastha, H. V., and Chowdhury, M. Oort: Efficient federated learning via guided participant selection. In *OSDI*, 2021.

Lai, F., Dai, Y., Singapuram, S., Liu, J., Zhu, X., Madhyastha, H., and Chowdhury, M. Fedscale: Benchmarking model and system performance of federated learning at scale. In *ICML*, 2022.

Lai, F., Dai, Y., Madhyastha, H. V., and Chowdhury, M. Modelkeeper: Accelerating dnn training via automated training warmup. In *NSDI*, 2023.

Li, A., Sun, J., Li, P., Pu, Y., Li, H., and Chen, Y. Hermes: an efficient federated learning framework for heterogeneous mobile clients. In *MobiCom*, 2021a.

Li, C., Zeng, X., Zhang, M., and Cao, Z. Pyramidfl: A fine-grained client selection framework for efficient federated learning. In *MobiCom*, 2022a.

Li, C., Zhuang, B., Wang, G., Liang, X., Chang, X., and Yang, Y. Automated progressive learning for efficient training of vision transformers. In *CVPR*, 2022b.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

Li, T., Sanjabi, M., Beirami, A., and Smith, V. Fair resource allocation in federated learning. In *ICLR*, 2019.

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

Li, T., Hu, S., Beirami, A., and Smith, V. Ditto: Fair and robust federated learning through personalization. In *ICML*, 2021b.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *ICLR*, 2019.

Liu, J., Lai, F., Dai, Y., Akella, A., Madhyastha, H. V., and Chowdhury, M. Auxo: Efficient federated learning via scalable client clustering. In *SoCC*, 2023.

Long, M., Cao, Y., Wang, J., and Jordan, M. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

Lv, C., Niu, C., Gu, R., Jiang, X., Wang, Z., Liu, B., Wu, Z., Yao, Q., Huang, C., Huang, P., Huang, T., Shu, H., Song, J., Zou, B., Lan, P., Xu, G., Wu, F., Tang, S., Wu, F., and Chen, G. Walle: An End-to-End, General-Purpose, and Large-Scale production system for Device-Cloud collaborative machine learning. In *OSDI*, 2022.

Marfoq, O., Neglia, G., Bellet, A., Kameni, L., and Vidal, R. Federated multi-task learning under a mixture of distributions. In *NeurIPS*, 2021.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Mei, Y., Guo, P., Zhou, M., and Patel, V. Resource-adaptive federated learning with all-in-one neural composition. In *NeurIPS*, 2022.

Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., and Huba, D. Federated learning with buffered asynchronous aggregation. In *AISTATS*, 2022.

Ozkara, K., Singh, N., Data, D., and Diggavi, S. Quped: Quantized personalization via distillation with applications to federated learning. In *NeurIPS*, 2021.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

Rapp, M., Khalili, R., Pfeiffer, K., and Henkel, J. Distreal: Distributed resource-aware learning in heterogeneous systems. In *AAAI*, 2022.

Shi, C., Wei, B., Wei, S., Wang, W., Liu, H., and Liu, J. A quantitative discriminant method of elbow point for the optimal number of clusters in clustering algorithm. *EURASIP Journal on Wireless Communications and Networking*, 2021.

Singapuram, S. S. V., Hu, C., Lai, F., Zhang, C., and Chowdhury, M. Flamingo: A user-centric system for fast and energy-efficient dnn training on smartphones. In *DistributedML*, 2023.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Wang, E., Kannan, A., Liang, Y., Chen, B., and Chowdhury, M. Flint: A platform for federated learning integration. In *MLSys*, 2023.

Wang, I., Nair, P., and Mahajan, D. Fluid: Mitigating stragglers in federated learning using invariant dropout. *Advances in Neural Information Processing Systems*, 36, 2024.

Wang, Y., Sun, D., Chen, K., Lai, F., and Chowdhury, M. Efficient dnn training with knowledge-guided layer freezing. *arxiv.org/abs/2201.06227*, 2022.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arxiv.org/abs/1804.03209*, 2018.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *ICML*, 2016.

Yao, D., Pan, W., O'Neill, M. J., Dai, Y., Wan, Y., Jin, H., and Sun, L. Fedhm: Efficient federated learning for heterogeneous models via low-rank factorization. *arxiv.org/abs/2111.14655*, 2021.

A APPENDIX

A.1 Experiment settings

Comparing with baselines We choose MobileNetV3-small ((Howard et al., 2019)) for Cifar-10, base model of NASBench201 ((Dong & Yang, 2019)) for FEMNIST, and modified smaller ResNet18 ((He et al., 2016)) for Speech Command and OpenImage as initial models. The detailed architecture of the base model of NasBench201 is shown in Figure 14. The detailed architecture of the modified small ResNet18 is shown in Figure 15.

To fairly evaluate the performance across different methods, HeteroFL, SplitMix, and FLuID should use the same architecture as FedTrans. However, HeteroFL, Splitmix, and FLuID shrink models, which means they take a large model and adopt some algorithm to reduce, compress, or prune it to form multiple small models. Therefore, we give the largest model transformed by FedTrans as the input large model to HeteroFL, SplitMix, and FLuID. Since HeteroFL and SplitMix do not support convolutional layer with groups, we convert the grouped convolution layer to non-grouped one, which potentially increases the complexity of the layer.

The hyperparameter setting for FedTrans is shown in Table 7. The training is considered complete when either the maximum number of training rounds is reached or the validation accuracy converges, which is defined as the accuracy not improving by more than 1% over 10 consecutive rounds. The hyperparameter settings for HeteroFL, SplitMix, and FLuID are the same as those in their paper.

Quality of transformed models To evaluate the quality of transformed models (Fig. 9), we fine-tune each transformed model on all the clients. We use the default FedAvg (McMahan et al., 2017) setting for this evaluation part, which means we remove the hardware capacity constraints and disable the transformation, adaptive model assignment, and soft aggregation.

B COMPUTATION AND COMMUNICATION OVERHEADS ANALYSIS

Due to the challenge of data heterogeneity and the nature of distributed computing, FL training itself is expensive. Therefore, FedTrans introduces minimal computation and communication overhead compared with standard FedAvg.

Clients The local training on the client is the same as FedAvg, with no computation overhead. After the local training, clients are required to upload the model weights, model gradient, and training loss back to the coordinator. However, the updated model weights can be easily derived from the model gradient and the model weights of the last round. Therefore, only the training loss is considered as communication overhead for clients. Overall, on the side of clients, there is no computational overhead and negligible (i.e.. a floating number) communication overhead.

Overhead	Estimated value
client's computation	0
client's communication	rpc
coordinator's computation	$r(mn+1)c + \|W\|c$
coordinator's communication	0

Table 5. Computation and communication overheads analysis for m registered clients, p participated clients, n models, r rounds, where c is a small constant and $|W|$ is the average size of the model weights.

Method	Avg. (s)	Std. (s)
FedTrans + FedAvg	134.5	237.1
FedAvg	226.3	325.6

Table 6. Round completion time comparison.

Coordinator After receiving the updates from clients, the coordinator is scheduled to do four steps of computation, which are (1) updating utilities, (2) updating local weights, (3) updating the degree of convergence (DoC), and (4) model transformation. Among these steps, updating utilities, updating the degree of convergence, and model transformation are computational overhead. Given m clients and n models, the coordinator needs to do $m \times n$ times of utility updating operations. For each utility update, the coordinator needs to calculate the standardized loss and the subtraction, which are considered to have constant complexity. Updating DoC calculates the average of loss slopes, which is considered to have constant complexity. We consider the model transformation happens at constant times. For each model transformation, the coordinator calculates the layer activeness and applies the widening and/or deepening operations, whose complexity is considered to be proportional to the size of model weights. As for communication, FedTrans does not introduce any overhead on the side of the coordinator. Overall, the computational and communication overhead analysis is summarized in Table 5.

C FEDTRANS MITIGATES THE STRAGGLER ISSUE.

In synchronous federated learning, slow clients could slow down the training process if clients are given the same workload, which is referred to as the straggler issue. FedTrans can mitigate the straggler issue as we assume each client has a hard requirement for the model complexity (MACs). As shown in Table 6, FedTrans improves FedAvg both in the average and the std of the round completion time among clients on FEMNIST dataset compared with FedAvg.

Hyperparameters	Cifar-10	FEMNIST	Speech Command	OpenImage
# of participants per round	10	100	100	100
maximum number of training rounds	1000	2000	1500	2000
step size to calculate the loss slope (δ)	20	30	100	50
local training steps		20		
batch size		10		
learning rate		0.05		
decay factor		0.98		
# of consecutive gradient to calculate activeness (T)		5		

Table 7. Hyperparameters

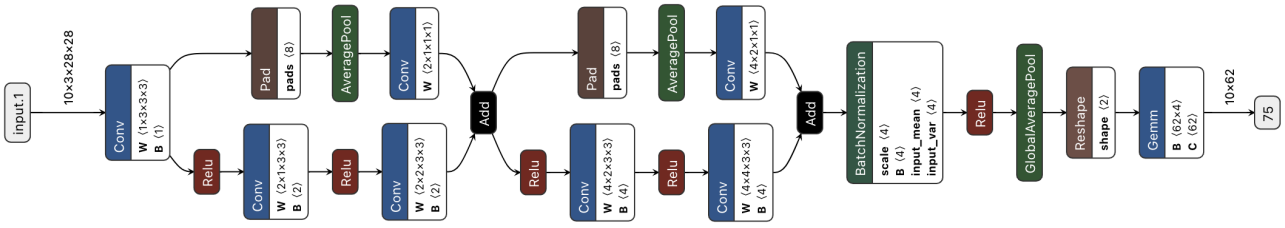


Figure 14. Base model of NASBench201

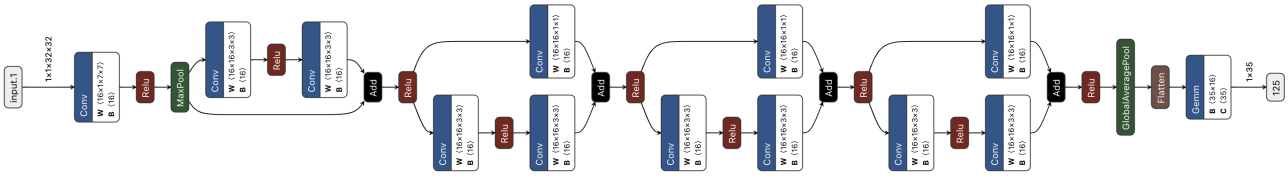


Figure 15. Modified smaller ResNet18