Federated Learning with Flexible Architectures

Jong-Ik Park and Carlee Joe-Wong ✉[0000−0003−0785−9291]

Carnegie Mellon University, Pittsburgh PA 15213, USA
cjoewong@andrew.cmu.edu
https://www.cmu.edu/

Abstract. Traditional federated learning (FL) methods have limited support for clients with varying computational and communication abilities, leading to inefficiencies and potential inaccuracies in model training. This limitation hinders the widespread adoption of FL in diverse and resource-constrained environments, such as those with client devices ranging from powerful servers to mobile devices. To address this need, this paper introduces Federated Learning with Flexible Architectures (FedFA), an FL training algorithm that allows clients to train models of different widths and depths. Each client can select a network architecture suitable for its resources, with shallower and thinner networks requiring fewer computing resources for training. Unlike prior work in this area, FedFA incorporates the *layer grafting* technique to align clients' local architectures with the largest network architecture in the FL system during model aggregation. Layer grafting ensures that all client contributions are uniformly integrated into the global model, thereby minimizing the risk of any individual client's data skewing the model's parameters disproportionately and introducing security benefits. Moreover, FedFA introduces the *scalable aggregation* method to manage scale variations in weights among different network architectures. Experimentally, FedFA outperforms previous width and depth flexible aggregation strategies. Specifically, FedFA's testing accuracy matches (1.00 times) or is up to 1.16 times higher globally for IID settings, 0.98 to 1.13 times locally, and 0.95 times to 1.20 times higher globally for non-IID settings compared to earlier strategies. Furthermore, FedFA demonstrates increased robustness against performance degradation in backdoor attack scenarios compared to earlier strategies. Earlier strategies exhibit more drops in testing accuracy under attacks—for IID data by 1.01 to 2.11 times globally, and for non-IID data by 0.89 to 3.31 times locally, and 1.11 to 1.74 times globally, compared to FedFA.

Keywords: Federated Learning · Heterogeneous Local Network Architectures · Backdoor Attack

1 Introduction

As the need for advanced decision-making capabilities in scenarios such as medical imaging and military operations grows, modern machine learning and deep learning techniques are increasingly in demand [2,4]. To tackle potential privacy

concerns associated with handling clients' data in these sensitive areas, federated learning (FL) has been developed. FL allows for training a shared machine learning model using data from multiple clients without the need to exchange or reveal their local data directly [23,24,27]. Clients iteratively compute updates on local models and periodically synchronize these updates with an aggregation server to create a global model, which is sent back to the clients for another round of local model updates. However, *traditional FL strategies struggle to integrate heterogeneous clients with varying computational and communication resources* [5,6,19]. For example, devices in a federated network may range from powerful fighter jets to less capable tanks in a military context or large hospitals to smaller clinics in healthcare. Prior works along these lines generally focus on reducing the number of local model updates that weaker clients complete during the training [21,30,32,34], which can slow model convergence. Moreover, each client must store a copy of and compute gradients on the full, possibly very large, model.

In this work, we explicitly account for heterogeneity in clients' compute and communication resources by allowing clients to customize their local model architectures according to their specific resources, ensuring efficient participation by avoiding delays from slower (straggler) clients, which could hinder the FL process and negatively impact global model updates [20, 34]. Thus, our work falls into the same category as width-flexible FL aggregation strategies like HeteroFL [6], depth-flexible FL aggregation strategies such as FlexiFed [41], and strategies flexible in both width and depth like NeFL [16]. These strategies enable FL clients to train network architectures with variable depths and widths.

However, prior works do not consider the fact that client networks' diversity (or heterogeneity) in FL presents **unique security challenges**. Combining models with different network architectures introduces weak points in the aggregation process that are susceptible to attacks [3,23]. These *weak points* refer to weights that are *incompletely aggregated*, since only a subset of clients compute their values due to the differences in network structures. Attackers can exploit these vulnerabilities in commonly used *backdoor attacks* [3, 23], which aim to induce inaccurate predictions on specific data inputs by manipulating model updates from malicious clients. By manipulating weights that are only updated by a few clients, attackers can successfully compromise the model, as depicted in the last two layers of the global model in Figure 1.

Following these concerns, another critical challenge arises from *scale variations* in client weights due to the heterogeneous nature of network architectures [5,6,39]. When clients possess varying numbers of layers and filters, scale variations arise, potentially causing unfairness in the global model aggregation: data from specific clients whose model weights have a larger scale may be disproportionately emphasized in the global model.

In response to these challenges, this paper proposes a novel strategy, '**Federated Learning with Flexible Architectures**' (FedFA), that retains the benefits of employing heterogeneous model architectures while minimizing the impact of weak point attacks and addressing scale variation in aggregation. Our strategy aggregates model layers uniformly, regardless of the complexities of

individual networks. We thus establish a global model that matches the greatest depth and width found among all local models. This setup allows each client to contribute to the value of each weight in the global model, minimizing the risks associated with specific weak point attacks. Lastly, we propose a fair-scalable aggregation method to ensure fairness across local models and reduce the model bias from scale variations.

In essence, our FedFA framework delivers four significant **contributions**:
1) We introduce a novel *aggregation strategy* that is the first, to the best of our knowledge, to address security challenges in FL on heterogeneous architectures. FedFA uniformly incorporates layers from various client models into a unified global model, exploiting similarities between layers of a neural network induced by the common presence of skip connections.
2) We are the first to effectively address scale variations in a dynamic training environment. We propose the *scalable aggregation method* to compensate for scale variations in the weights of heterogeneous network architectures.
3) FedFA *utilizes NAS (Neural Architecture Search)* [18] to optimize each client's model architecture based on its specific data characteristics, thus elucidating the impact of employing optimal model architectures tailored to local data characteristics on both local and global model performance.
4) In our experiments on Pre-ResNet, MobileNetV2, and EfficientNetV2, *FedFA outperforms previous width- and depth-flexible strategies*. FedFA achieves accuracy improvements by factors of up to 1.16 in IID (independent and identically distributed) data settings and 1.20 in non-IID settings on the global model. In non-IID environments, clients' local accuracies increase by up to 1.13. Furthermore, FedFA demonstrates increased robustness. In contrast, prior strategies experienced accuracy declines under backdoor attacks by up to 2.11 in IID and up to 3.31 globally and 1.74 locally in non-IID settings compared to FedFA. Additionally, our experiments with a Transformer-based language model showed a significant reduction in perplexity, improving by 1.07 to 4.50 times.

We outline the 'Related Work' in Section 2 and motivate FedFA in Section 3, which introduces previous width- and depth-flexible strategies and the model properties that we exploit. Next, 'Flexible Federated Learning' in Section 4 details the design of FedFA, and Section 5 discusses our experimental results. We discuss directions of future work in Section 6 and conclude in Section 7.

2 Related Work

2.1 Heterogeneous Network Aggregation in Federated Learning

FlexiFed [41] is a depth-flexible strategy that aggregates common layers of clients' networks with varying depths, like in VGG-16 and VGG-19, forming global models from different layer clusters. **HeteroFL** [6] is a width-flexible strategy that accommodates clients with varying resources by aggregating networks of different widths. It selectively aggregates weights where available and employs a heuristic to manage weight variability [28]. **NeFL** [16] combines width

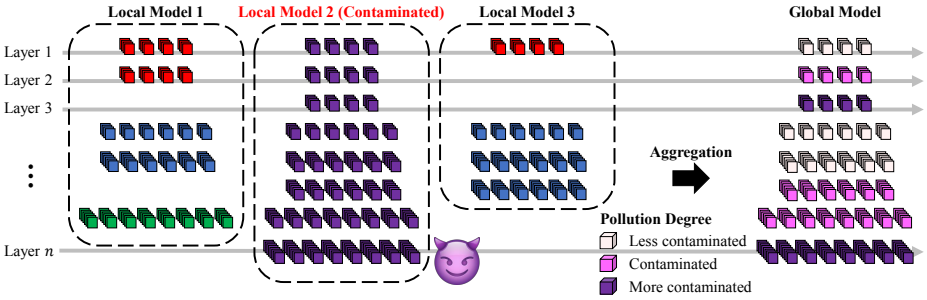

Fig. 1. Aggregating heterogeneous networks introduces vulnerabilities due to incomplete aggregation and increased susceptibility to backdoor attacks for the global model.

and depth flexibility, using skip connections to omit certain blocks and structured pruning for width control, similar to HeteroFL [6]. These strategies result in *incomplete aggregation*, which poses security risks to the global model (see Figure 1). Here, *incomplete aggregation* refers to the process where weights in layers or filters in the global model at a specific position are updated with contributions from only a subset of the participating local models rather than all.

Unlike HeteroFL, FlexiFed and NeFL do not consider scale variations between clients' model weights, leading to potential unfairness in model aggregation. Moreover, HeteroFL's scaling factors might be less relevant in architectures with batch normalization layers [15], which stabilize learning and reduce the need for additional scaling. For more on HeteroFL's scaling, refer to Appendix G. Furthermore, several other width- and depth-flexible strategies like Sub-FedAvg [39] and TailorFL [5] predominantly rely on online filter pruning, which can lead to significant computational overhead, contradicting their aim for computational efficiency. Therefore, in this study, we benchmarked our proposed FedFA strategy against HeteroFL, FlexiFed, and NeFL (see Section 5).

2.2 Split Learning

The **split learning** FL framework also allows clients to maintain a variable number of neural network layers, which connect to common layers stored at the aggregation server [7,31]. Clients can choose the number of local layers according to their computing resources and data characteristics [35]. However, split learning requires intensive client-server communication, as clients cannot compute local model updates without communicating with the layers at the server [10,29,38].

2.3 Skip Connections

Modern network architectures have highlighted the significance of **skip connections** (or residual connections) in neural networks, a feature we utilize in our layer grafting method to mitigate the security risks of incomplete aggregation

(e.g., ResNets [13], MobileNets [14], and EfficientNets [36]). Skip connections allow gradients to bypass specific layers, mitigating vanishing gradients in deep learning [22]. This functionality preserves training stability and enhances pattern recognition efficiency, making these networks suitable for resource-constrained devices [13,26], and making layers similar [9,16,40] (See Appendix B).

3 Motivation: Challenges in Heterogeneous Aggregation

3.1 Security Concerns of Heterogeneous Network Aggregation

Aggregating models from diverse network architectures presents security challenges, as malicious actors can exploit these strategies to carry out sophisticated and covert attacks, implanting subtle yet harmful alterations within model updates [3,37]. These modifications, often in the form of triggers or slight changes, are designed to exploit the aggregation process covertly [1] and steer a model to degrade its accuracy or to embed hidden vulnerabilities, which become more pronounced over time [23,33]. A particular point of vulnerability with heterogeneous client architectures is the layers that are not fully aggregated (i.e., *incomplete aggregation*) in the global model (which has the largest width and depth across all local models) due to limited contributions from few clients, as depicted in Figure 1.

A common attack embeds a backdoor into a malicious client's local model in a heterogeneous FL setting. This hidden function or behavior, designed to remain dormant, activates only under specific conditions. Once integrated into the global model, these backdoors can trigger significant security breaches, such as targeted misclassifications [1,3]. Mathematically, a backdoor attack computes a malicious model update as follows:

$$\Delta M^t_{\text{malicious}} \leftarrow \Delta M^t_c + \lambda \cdot \Delta M_{\text{backdoor}} \tag{1}$$

Here, ΔM^t_c is the original update from client c at global iteration t, and $\Delta M_{\text{backdoor}}$ represents the backdoor modification. λ determines the intensity of the backdoor effect. The contribution of $\lambda \cdot \Delta M_{\text{backdoor}}$ to the aggregation process of all clients' model updates dictates the extent of damage to the global model. Specifically, weights of the global model that undergo incomplete aggregation are more susceptible to being compromised, as can be seen from the aggregation:

$$\Delta M^t_G = \frac{1}{N}\left(\sum_{c=1}^{N-1} \Delta M^t_c + \Delta M^t_{\text{malicious}}\right) \not\approx \frac{1}{N-1}\sum_{c=1}^{N-1} \Delta M^t_c$$

For the global model update, ΔM^t_G, in the presence of many clients N, the influence of an attack by malicious clients could be diluted. However, this dilution is limited to the weights that are updated by most or all clients. Furthermore, by selecting the largest network architecture, attackers can amplify the effect of their attacks, in contrast to local clients who select network architectures based on their resource capabilities or the characteristics of their local data.

3.2 Scale Variations in Heterogeneous Networks

During the training process, the gradients of weights can vary significantly across different network architectures, influenced by the number of weights present in each network before applying the loss function. As a result, the magnitudes of weight changes during each step of gradient descent (i.e., step sizes) can differ due to the variations in gradients during optimization. This leads to scale variations across the heterogeneous networks [11] (refer to Appendix F for more details). In FL environments, such variations significantly impact the performance and accuracy of the aggregated global model. For instance, consider two client models within an FL system, Model A and Model B; each has a distinct architecture. The magnitude of their updates in a given round of FL, ΔM_A and ΔM_B, can differ significantly based on their respective models' complexities. When these models are aggregated using an unweighted averaging method, as is typical in federated learning, we have:

$$\Delta M = \frac{1}{2}(\Delta M_A + \Delta M_B)$$

This may lead to an imbalance, causing the aggregated model update, ΔM, to be skewed towards the model with a larger magnitude of weights.

4 Flexible Federated Learning

This section details our methodology for implementing Federated Learning with Flexible Architectures (FedFA). We first introduce an overview of our FedFA procedure and then focus on the procedure for the layer grafting method, which ensures security by enforcing complete aggregation. Additionally, we introduce scaling factors for normalizing model weights, a critical component for achieving fair aggregation within the FedFA framework.

4.1 FedFA Procedure

The FedFA algorithm, as presented in Algorithm 1, incorporates the layer grafting and the scalable aggregation methods into the FL paradigm. Initially, the server proposes a variety of network architectures (line 1). Clients then choose their architectures, e.g., using Neural Architecture Search (NAS) methods (line 2), followed by the server setting up the global model with the maximum possible number of weights (line 3). The algorithm begins its iterative process by selecting a random subset of clients to update the model in each round (lines 5-9). Subsequently, the layer grafting method (explained in Section 4.2) will adjust each client's updated local model to align with the global model's architecture (line 11).

For scalable aggregation, the server first finds the weights below the 95th percentile in each layer of each local model from each participating client (line 12). With these extracted weights, which exclude outliers, the server calculates

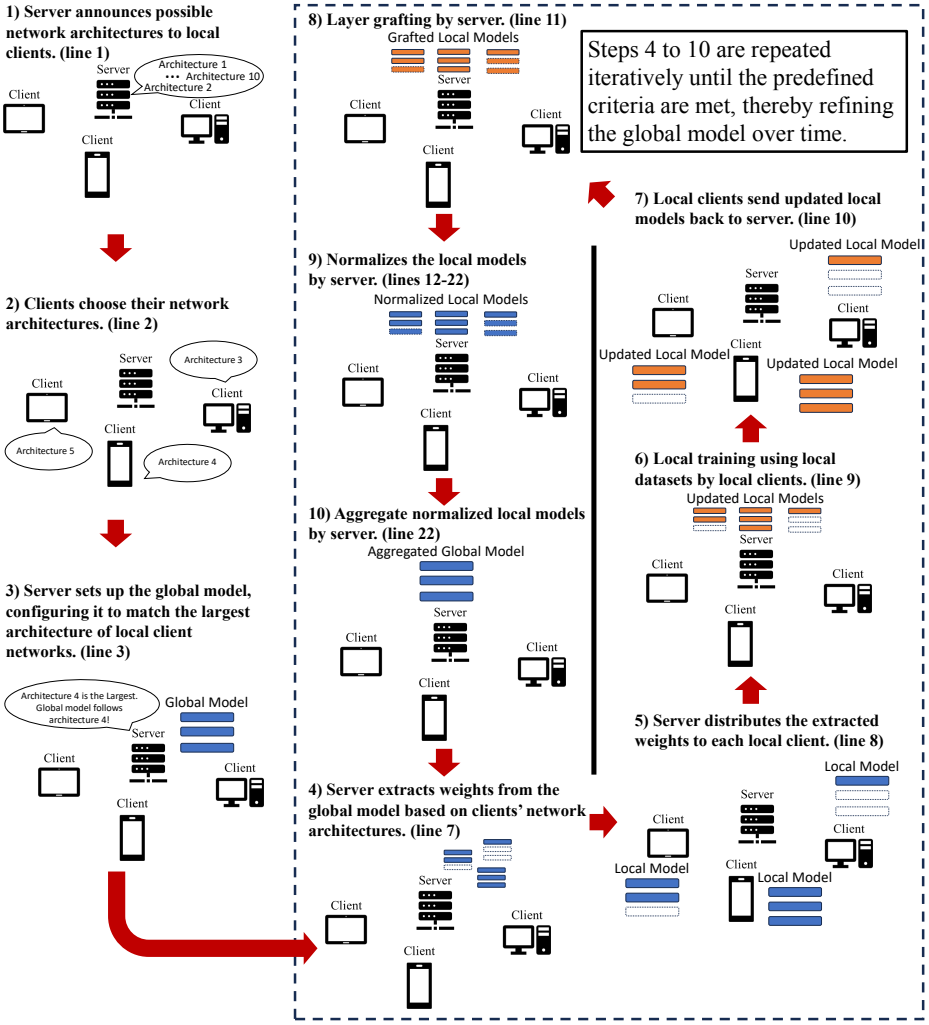

Fig. 2. The FedFA workflow: Server announces network architectures, clients select and send preferences, server configures the global model, clients perform local training, updates are sent to the server, where they are grafted, normalized, and aggregated, iterating until convergence criteria are achieved. Each step in the workflow is mapped to specific lines in Algorithm 1 and depicted in the corresponding steps of the figure.

the scaling factor, $\alpha_c^{(l)}$, for each layer (line 18). Normalization of each local model and aggregation of these normalized local models then ensures that the updates from all participating clients are aggregated in a balanced manner, preserving uniformity and scale consistency throughout the network (lines 16-22) (more details are in Section 4.3).

Algorithm 1 FedFA with the layer grafting and the scalable aggregation methods. The algorithm operates over T rounds with a client set $\mathcal{C}$. Here, the clients' participating rate is C. Each client $c \in \mathcal{C}$ selects a model architecture from a predefined set $\mathcal{A}$. The server determines the maximal architecture width $N^{(l)}_{width,\max}$ and depth $N^{(s)}_{depth,\max}$. The global model, M^t_G, is updated at the server through the aggregation of client updates.

Require: Local datasets $\mathcal{D} = \{D_c | c \in \mathcal{C}\}$.
Ensure: Updated global model M^T_G.
1: Server proposes architecture set $\mathcal{A}$.
2: Clients select network architectures from $\mathcal{A}$ using NAS methods and report their architectures (width $N^{(l)}_{width,c}$ and depth $N^{(s)}_{depth,c}$) to the server.
3: Initialize the global model, M^0_G, with $N^{(l)}_{width,\max}$, $N^{(s)}_{depth,\max}$. {Server}
4: **for** $t = 0$ to T **do**
5: Select a subset $\mathcal{C}_{sel}$ of $m = C \times |\mathcal{C}|$ clients. {Server}
6: **for** all clients c in $\mathcal{C}_{sel}$ **do**
7: Extract M^t_c from M^t_G according to $N^{(l)}_{width,c}$ and $N^{(s)}_{depth,c}$. {Server}
8: Distribute M^t_G to the client c. {Server}
9: $M^{t+1}_c \leftarrow \text{LocalUpdate}(M^t_c, D_c)$ {Client c}
10: Send M^{t+1}_c to the server. {Client c}
11: Apply the layer grafting to M^{t+1}_c. {Server, Algorithm 2}
12: $M_{95\%,c} \leftarrow$ Under 95th percentile values of M^{t+1}_c for each layer {Server}
13: **end for**
14: **for all** layers l in M^t_G **do**
15: $M^{'(l)}_G, \gamma^{(l)} \leftarrow \text{Zeros}(M^{t,(l)}_G)$ {Server}
16: **for** all clients c in $\mathcal{C}_{sel}$ **do**
17: $C_I, C_o \leftarrow$ Input and output channel sizes of $M^{(l)}_c$ {Server}
18: $\alpha^{(l)}_c \leftarrow \frac{\frac{1}{m}\sum_{\kappa \in \mathcal{C}_{sel}} ||M^{(l)}_{95\%,\kappa}||}{||M^{(l)}_{95\%,c}||}$ {Server}
19: Add $N_{D_c}\alpha^{(l)}_c M^{(l)}_c$ to $M^{'(l)}_G[:C_o, :C_I]$. {Server}
20: Add N_{D_c} to all elements of $\gamma^{(l)}[:C_o, :C_I]$. {Server}
21: **end for**
22: $M^{(l)}_G \leftarrow M^{'(l)}_G / \gamma^{(l)}$ {Server}
23: **end for**
24: $M^{t+1}_G \leftarrow M^t_G$ {Server}
25: **end for**

26: **Function** LocalUpdate(M, D)
27: Update model M using local dataset D.
28: **Return** Updated model M

Since local models might differ in width from the global model, contiguous structured pruning [6,16] is used (line 19). This process accumulates the weights of local models only at the same position, considering their input and output channel sizes ($[:C_O, :C_I]$). Here, $M^{'(l)}_G$ and $\gamma^{(l)}$ are temporary placeholders with the same architecture as the global models, initialized with zeros for all

elements in every round (line 14). $M_G^{'(l)}$ is used for accumulating local updates (line 19), and $\gamma^{(l)}$ is for the weighted average of these local updates (line 20). $\gamma^{(l)}$ considers the number of data samples for each client of line 1, N_{D_c}, aligning with the original FedAvg algorithm [24]. If the server cannot even access the number of data samples, we take $N_{D_c} = 1$. After accumulation, the server can obtain the updated global model (line 24) by element-wise dividing $M_G^{'(l)}$ by $\gamma^{(l)}$ for every layer l (line 22). The algorithm continues through these rounds until predefined criteria are met. The overall FedFA process is visually summarized in Figure 2. We also show the effectiveness of heterogeneous network aggregation strategies in Appendix D and the convergence analysis of FedFA in Appendix E.

4.2 Layer Grafting Method for Ensuring Security

In FL, client models can vary in architecture due to differences in computational resources and data characteristics. This heterogeneity can lead to adversarial attacks during the aggregation of local models, potentially compromising the security of the global model. To address these issues, we introduce the layer grafting method (line 11 in Algorithm 1), which ensures uniformity in model architectures while accommodating client-specific characteristics.

Algorithm 2 The Layer Grafting method. This algorithm standardizes the depth of each section $M_c^{(s)}$ in local model M_c to the maximum depth $N_{depth,\max}^{(s)}$ across all clients c. $N_{depth,c}^{(s)}$ denotes the current depth of section $M_c^{(s)}$, while ΔD represents the depth difference that needs to be augmented. The last residual block in a section is denoted by $R_{last}^{(s)}$, and $\oplus$ symbolizes the grafting operation of this block to the section. The process iteratively augments each section's depth $N_{depth,c}^{(s)}$ by grafting $R_{last}^{(s)}$ until the target depth $N_{depth,\max}^{(s)}$ is matched.

Require: Local model M_c for client c, maximum depth $N_{depth,\max}^{(s)}$ for each section $M_c^{(s)}$ across all clients.
Ensure: Updated model M_c with each section $M_c^{(s)}$ augmented to depth $N_{depth,\max}^{(s)}$.

for each section $M_c^{(s)}$ in model M_c **do**
  $N_{depth,c}^{(s)} \leftarrow$ current depth of section $M_c^{(s)}$
  $\Delta D \leftarrow N_{depth,\max}^{(s)} - N_{depth,c}^{(s)}$
  if $\Delta D > 0$ **then**
    $R_{last}^{(s)} \leftarrow$ last residual block in section $M_c^{(s)}$
    for $d = 1$ to ΔD **do**
      $M_c^{(s)} \leftarrow M_c^{(s)} \oplus R_{last}^{(s)}$
    end for
  end if
end for

The layer grafting method, as described in Algorithm 2 and illustrated in Figure 6 in the Appendix, aims to standardize the depth of each section across

the FL client models. In this context, a 'section' is a part of the model where residual blocks share the same sequence of filter numbers in layers. A single model may comprise multiple such sections, each containing several residual blocks.

This addition is iteratively performed until the section reaches the specified maximum depth (lines 4-9 of the Algorithm 2). This systematic addition of residual blocks guarantees a consistent depth across all client models, thereby preserving architectural coherence within the FL network. Further details and the rationale behind layer grafting, particularly regarding the similarity of layers within residual blocks, are elaborated in Appendix B.

To see how layer grafting mitigates the potential risk of backdoor or poisoning attacks in aggregations across heterogeneous client architectures [6, 16, 41], we examine the aggregation for the global FL model with layer grafting, assuming the commonly used averaging method [24]:

$$\begin{aligned} M_G^t &= \frac{1}{N}\left(\sum_{c=1}^{N-1}(M_G^{t-1}+\Delta M_c^t)+(M_G^{t-1}+\Delta M_{\text{malicious}}^t)\right) \\ &\approx \frac{1}{N-1}\sum_{c=1}^{N-1}\left(M_G^{t-1}+\Delta M_c^t\right) \end{aligned}$$

In this simple aggregation formula [24], M_G^t and M_G^{t-1} represent the global models at iterations t and $t-1$, respectively. N denotes the total number of clients, and ΔM_c^t are the updates from each individual client c. This equation illustrates how the influence of malicious updates is diluted in a complete aggregation, especially for a large number (N) of clients. This mitigation is effective if the updated weights, $M_G^{t-1}+\Delta M_c^t$ and $M_G^{t-1}+\Delta M_{\text{malicious}}^t$ are on the same scale.

4.3 Scalable Aggregation: Normalization of Local Model Weights

In addressing scale variations stemming from heterogeneous network architectures in FL, we incorporate a crucial normalization step in the model aggregation process (lines 18-19 in Algorithm 1). This involves applying scaling factors to the weight updates from each client model to ensure balanced contributions in the aggregated model.

The scaling factor, denoted as $\alpha_c^{(l)}$ for layer l of each local model, is calculated in response to the diverse scales of weight updates from different network architectures. These factors are determined based on the L2 norm to prevent larger updates from disproportionately influencing the global model.

The formula for the scaled weights in the FedFA framework is:

$$\alpha_c^{(l)} M_c^{(l)} = \frac{\frac{1}{m}\sum_{\kappa\in\mathcal{C}_{sel}}||M_{95\%,\kappa}^{(l)}||}{||M_{95\%,c}^{(l)}||} M_c^{(l)} \text{ where } c \in \mathcal{C}_{sel}$$

Here, $M_c^{(l)}$ represents the weights from the local model of client c for layer l. In one aggregation round, m denotes the number of participating clients, $\mathcal{C}_{sel}$.

The L2 norm $||M^{(l)}_{95\%,\kappa}||$ refers to the weights within the inner part of the 95th percentile from client κ for the same layer l. Similarly, $||M^{(l)}_{95\%,c}||$ signifies the weights under the 95th percentile in layer l for client c. We utilize the 95th percentile as it effectively mitigates the impact of outliers, which could otherwise skew the accuracy of scale calculations. This approach is beneficial in reducing the influence of anomalous weight values that may arise from noisy data or atypical client models.

This normalization process, with the scaling factor $\frac{1}{||M^{(l)}_{95\%,A}||}$, applied layer-wise, ensures that the aggregated model accurately reflects the diverse architectures in the network. Furthermore, the averaging component, represented by the numerator $\frac{1}{m}\sum_{\kappa\in\mathcal{C}sel}||M^{(l)}_{95\%,\kappa}||$, moderates the convergence speed by averaging the magnitude of updates across the participating clients. This leads to a more balanced and representative global model, adjusting for scale variations across different client models and enhancing the overall fairness of the FL system.

4.4 Global Model Distribution Step

The model distribution step, detailed in Algorithm 3 in the Appendix, focuses on tailoring the aggregated global model to align with the unique architectural requirements of each client (line 7 in Algorithm 1). This critical process involves modifying the global model to conform to each client's model's specific depth and width weights. To achieve this, the algorithm systematically adjusts the global model by reducing its depth (lines 3-6 in Algorithm 3) and width (lines 8-11 in Algorithm 3) to those specified by each client's architecture. By following this procedure, the global model is effectively customized, making it compatible with the diverse architectures of all participating clients in the FL network.

5 Experimental Evaluation

This section evaluates FedFA's performance by benchmarking its testing accuracy, robustness against backdoor attacks, and computational complexities. This evaluation uses local and global test datasets, comparing FedFA with previous aggregation strategies offering width and depth flexibility.

5.1 Experimental Setup

We assess Pre-ResNet, MobileNetV2, and EfficientNetV2 using the CIFAR-10, CIFAR-100, and Fashion MNIST datasets in IID and non-IID environments. In IID settings, each client has samples from all classes, with a uniform data distribution where the minimum number of samples for a client can be up to half the maximum number of samples for any other client.

For Non-IID settings, clients get samples from 20% of the dataset classes but maintain equal samples for each class they hold. Here, during local training, clients zero-out logits for absent classes. We replace typical batch normalization

Table 1. Global and local testing accuracies for Pre-ResNet, MobileNetV2, and EfficientNetV2 on CIFAR-10, CIFAR-100, and Fashion MNIST datasets in IID and non-IID scenarios. FedFA's depth-, width-, and both depth and width-flexible approaches lead to higher test accuracy with no malicious clients and lower accuracy drops (under attacks with intensity $\lambda = 20$ and 20% malicious clients) compared to prior depth-, width-, and both depth and width-flexible approaches. The maximum and minimum test accuracies are highlighted in bold for each scenario.

CIFAR 10 – Pre-ResNet

λ	Malicious Client	IID						Non IID											
		Global Test Accuracy						Local Test Accuracy						Global Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
1	0 %	77.0	**75.9**	**79.7**	78.1	77.9	77.2	87.0	**88.1**	90.7	90.2	89.9	**91.0**	48.9	**48.7**	**51.9**	51.5	48.7	51.2
1	1 %	77.3	**75.8**	**79.7**	76.8	78.0	77.3	89.3	**88.0**	**89.8**	88.2	89.7	88.6	48.7	**47.0**	**51.7**	50.7	48.3	49.8
1	20 %	73.3	**71.6**	**76.8**	73.4	74.0	71.9	88.2	**91.1**	89.0	89.2	89.8	**87.3**	41.8	39.9	**46.6**	40.7	42.8	**39.5**
20	20 %	33.4	23.3	35.2	25.2	**40.5**	**22.8**	**82.7**	73.8	77.3	**70.4**	81.5	74.5	**25.5**	**12.5**	21.1	14.3	24.3	13.6
Accuracy Drop		56.6	69.3	55.8	67.7	**48.0**	**70.5**	**4.9**	16.2	14.8	**22.0**	9.3	18.1	**47.9**	**74.3**	59.3	72.2	50.1	73.4

CIFAR 100 – MobileNetV2

λ	Malicious Client	IID						Non IID											
		Global Test Accuracy						Local Test Accuracy						Global Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
1	0 %	31.5	**30.2**	**39.0**	33.7	32.2	31.5	47.9	47.1	**53.0**	**47.0**	48.1	45.5	31.2	30.3	**37.8**	32.6	31.9	**30.1**
1	1 %	31.0	**30.2**	**38.7**	33.1	32.8	31.1	48.4	48.0	**52.5**	49.2	46.5	**44.4**	30.9	**29.4**	**37.4**	32.1	31.6	29.7
1	20 %	28.6	**27.4**	**34.9**	30.1	29.4	27.7	44.1	46.2	**49.9**	43.5	46.2	**41.8**	26.2	**24.1**	**32.2**	26.8	27.6	24.9
20	20 %	17.2	**1.6**	14.4	2.5	**18.2**	2.6	37.3	**18.5**	33.9	32.0	**38.8**	22.2	**14.2**	**1.6**	12.4	8.3	14.2	2.8
Accuracy Drop		45.4	**94.7**	63.1	92.6	**43.5**	91.7	22.1	**60.7**	36.0	31.9	**19.3**	51.2	**54.5**	**94.7**	67.2	74.5	55.5	90.7

Fashion MNIST – EfficientNetV2

λ	Malicious Client	IID						Non IID											
		Global Test Accuracy						Local Test Accuracy						Global Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
1	0 %	87.2	87.1	**87.7**	**86.7**	87.4	87.0	96.3	95.4	**95.1**	**96.5**	95.5	96.4	52.8	**43.9**	**56.8**	56.7	50.7	52.5
1	1 %	87.1	87.4	**87.8**	**86.7**	87.6	86.8	96.4	**96.7**	95.6	**92.5**	94.5	95.4	**46.9**	49.0	50.3	**51.9**	47.0	49.2
1	20 %	86.2	86.0	**86.7**	**85.3**	86.4	85.4	**96.7**	95.0	95.1	95.9	**93.8**	94.6	45.7	47.5	**50.1**	49.1	**45.2**	44.4
20	20 %	45.7	**31.3**	**56.0**	53.3	39.1	38.2	86.3	**73.5**	**89.2**	82.3	84.5	82.0	24.2	**15.0**	**25.0**	17.9	23.2	17.4
Accuracy Drop		47.6	**64.1**	**36.1**	38.5	55.3	56.1	10.4	**23.0**	**6.2**	14.7	11.5	14.9	**54.2**	65.8	56.0	**68.4**	54.2	66.9

layers with static versions, as seen in HeteroFL [6]. We also utilized a language model with a Transformer using WikiText-2 to demonstrate that our method is generalizable. Detailed network structures are presented in Table 4, and more training details are in Table 6 in the Appendix.

Evaluations were conducted under four scenarios with varying impacts of backdoor attacks from malicious clients. Here, the backdoor attacks involve the random shuffling of the data labels among clients to induce misclassification. Scenarios have different portions of malicious clients over entire local clients (0 %, 1 %, and 20 %) and two intensities of attacks, ($\lambda = 1, 20$ in Eq. 1). Also, for all scenarios, we assume that half of the clients have limited computational resources and choose the smallest architectures. The other clients choose their architectures employing ZiCo [18], a cost-effective NAS method that requires

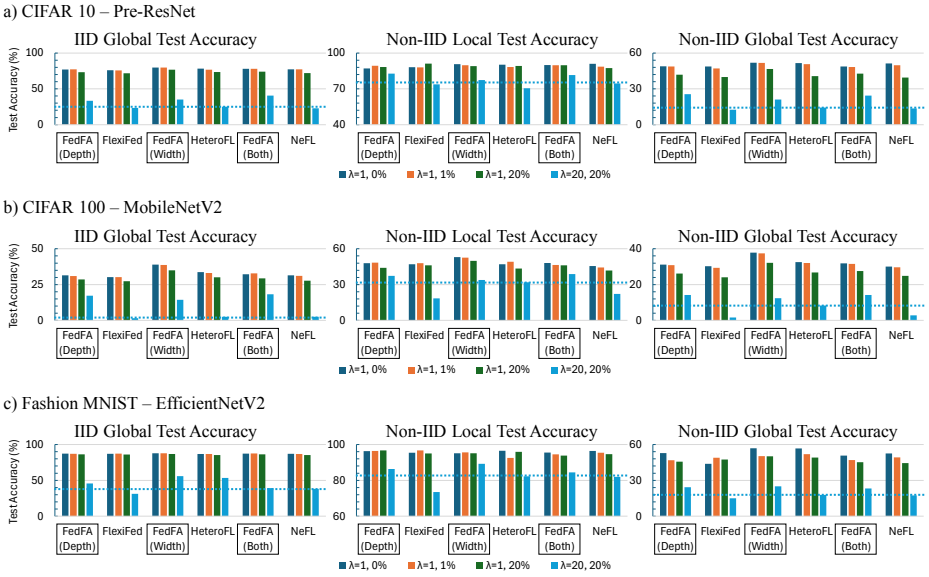

Fig. 3. Visualizations of FedFA's robustness against backdoor attacks in different FL settings across the CIFAR-10 with Pre-ResNet, CIFAR-100 with MobileNetV2, and Fashion MNIST with EfficientNetV2 datasets. The blue dotted lines are positioned below the lowest accuracy of FedFA but above the next highest accuracy among FlexiFed, HeteroFL, and NeFL. They underscore the robustness of FedFA when the attack intensity $\lambda = 20$ with 20% malicious clients.

only forward passes and uses an evolutionary algorithm. This method decides local network architectures among the network candidates specified in Table 5 in the Appendix, based on local data for each client.

5.2 Baselines and Metrics

After aggregating the local models, we calculate the *global testing accuracy* using a global test dataset. In non-IID settings, we additionally use several local test datasets extracted from the global dataset, ensuring they reflect the local clients' class distributions. After local training and before aggregation, we test the local models to determine an *average local testing accuracy*. This metric allows us to measure the effectiveness of local personalization. For the Transformer model, we use *average local perplexity* to assess the performance of the local clients' language models after local training for every round. Here, perplexity measures how well a probability model predicts a sample.

To evaluate *computational complexity*, we rely on multiply-accumulate (MAC) calculations. $\text{MACS}_{n=i}$ is the MAC for one local epoch of a local model with given local data, differentiated by architecture $n = i$. $N_{n=i}$ counts such architectures in the FL system. The average MAC is $\overline{\text{MACS}}$ given varied complexities

Table 2. Computational complexities for Pre-ResNet, MobileNetV2, and EfficientNetV2 on CIFAR-10, CIFAR-100, and Fashion MNIST datasets in IID and non-IID scenarios. FedFA shows comparable complexity to its baselines.

CIFAR 10 – Pre-ResNet													
		IID						Non IID - Local Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
Computational Complexity	MACE (T)	2.05	2.02	2.29	2.26	5.10	5.03	2.05	2.02	2.29	2.26	5.10	5.03
	T·E	357.0	357.0	359.0	358.0	356.0	356.5	206.0	206.0	210.0	208.5	208.0	207.0
	TMAC (T)	730.31	721.29	821.12	808.72	1814.35	1794.47	421.41	416.21	480.32	471.00	1060.07	1041.95

CIFAR 100 – MobileNetV2													
		IID						Non IID - Local Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
Computational Complexity	MACE (T)	0.22	0.22	0.23	0.22	0.34	0.34	0.22	0.22	0.23	0.22	0.34	0.34
	T·E	130.5	131.5	132.0	131.0	132.5	132.0	131.0	131.0	131.0	131.5	131.5	131.0
	TMAC (T)	28.59	28.45	29.77	29.18	45.39	44.66	28.69	28.34	29.54	29.29	45.05	44.33

Fashion MNIST – EfficientNetV2													
		IID						Non IID - Local Test Accuracy					
		FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL	FedFA (Depth)	Flexi Fed	FedFA (Width)	Hetero FL	FedFA (Both)	NeFL
Computational Complexity	MACE (T)	0.25	0.25	0.26	0.26	0.42	0.42	0.25	0.25	0.26	0.26	0.42	0.42
	T·E	163.2	163.4	162.8	162.6	162.6	163.0	64.6	66.0	67.4	64.0	64.4	68.4
	TMAC (T)	40.57	40.11	42.42	41.84	68.53	67.85	16.06	16.20	17.56	16.47	27.14	28.47

among local models. The MAC is $\text{MACE} = N_p \cdot \overline{\text{MACS}}$ for a single local epoch with all clients. The FL system's total MAC, TMAC, is found by multiplying total rounds of aggregation steps, T, by local epochs, E.

5.3 Evaluation

Testing Performance and Robustness Against Backdoor Attacks Table 1 shows the testing results of scenarios with varying model depths and widths for FedFA and previous flexible aggregation strategies in width and depth. Each scenario was tested three times, and the table presents the average results.

When only depth is varied, *FedFA outperforms FlexiFed* with a 1.00 (equivalent) to 1.04 times improvement globally in IID, and 0.98 to 1.02 times locally and 1.00 to 1.20 times globally in non-IID settings. With width variations, *FedFA exceeds HeteroFL*, achieving 1.01 to 1.16 times better accuracy globally in IID, 0.99 to 1.13 times locally, and 1.00 to 1.16 times globally in non-IID settings. For combined width and depth changes, *FedFA surpasses NeFL*, showing a 1.00 to 1.02 times improvement globally in IID, 0.99 to 1.06 times locally, and 0.95 to 1.06 times globally in non-IID settings. Overall, *FedFA outperforms other heterogeneous strategies in testing accuracy* except in 2 out of 9 scenarios.

As shown in Figure 3, backdoor attack scenarios reveal more distinct differences. When varying only the depth, FlexiFed experiences a more significant drop in testing accuracy than FedFA, with decreases of 1.22 to 2.09 times globally in IID, 2.21 to 3.31 times locally, and 1.21 to 1.74 times globally in non-IID settings. With width variation only, HeteroFL sees a testing accuracy drop of 1.07 to 1.47 times globally in IID, 0.89 to 2.37 times locally, and 1.11 to 1.22 times

Table 3. Average local testing perplexities for Transformer on the WikiText-2 dataset. FedFA's three variants yield much lower perplexities than its competitors.

	FedFA (Depth)	FlexiFed	FedFA (Width)	HeteroFL	FedFA (Both)	NeFL
Perplexity	205.5	**925.0**	**128.1**	157.5	143.1	153.5

globally in non-IID settings compared to FedFA. When the accuracy was 0.89 times higher (specifically in the CIFAR-100 local test scenario), FedFA consistently achieved much greater accuracy than HeteroFL under normal and severe attack conditions. For scenarios with width and depth variation, NeFL shows a drop in testing accuracy of 1.01 to 2.11 times globally in IID, 1.30 to 2.65 times locally, and 1.23 to 1.63 times globally in non-IID settings compared to FedFA. To summarize, FedFA generally surpasses other heterogeneous strategies in testing accuracy except in 3 out of 27 scenarios. These findings indicate that *FedFA is remarkably robust against backdoor attacks on the global model.*

Lastly, to demonstrate the generality of FedFA, we also examine its performance with transformers. The earlier strategies exhibit perplexities that are 1.07 to 4.50 times higher than those of FedFA, as detailed in Table 3.

Computational Complexity FedFA, employing layer grafting and scalable aggregation, has slightly higher computational complexity than earlier heterogeneous methods. Yet, for targeted testing accuracies—70% (IID) and 40% (non-IID) in CIFAR-10, 25% (both IID and non-IID) in CIFAR-100, and 80% (IID) and 30% (non-IID) in Fashion MNIST—the computational complexities are only 0.95 to 1.02 times higher. This indicates that FedFA's computational overhead is not marginally higher than earlier strategies, as presented in Table 2.

6 Future Work and Limitations

Future research could enhance FedFA's scalability for larger and more complex networks by optimizing algorithms to reduce the communication overhead and computational burden on clients. Developing dynamic client participation algorithms based on resource availability and network conditions could improve resource efficiency and model convergence speed but require careful mechanisms to handle clients' heterogeneous architectures. Advanced security mechanisms are necessary to detect and mitigate a broader range of adversarial attacks beyond the backdoor attacks we consider, including those exploiting model aggregation vulnerabilities. Further personalizing model architectures based on client data characteristics using advanced NAS techniques could improve local model performance. Additionally, integrating FedFA with edge computing paradigms could address latency, bandwidth, and real-time processing challenges in highly distributed environments.

Despite its advantages, FedFA has limitations that need to be addressed. One significant limitation is that all clients must employ the same type of network

architecture, such as all using ResNets, MobileNets, or EfficientNets. This uniformity can restrict the flexibility and efficiency of the system, especially when dealing with diverse client capabilities and requirements. Future work should focus on enabling support for heterogeneous network types within the same federated learning framework to accommodate the variety of client devices better and improve overall performance and scalability.

7 Conclusion

This paper introduces FedFA, a width- and depth-flexible aggregation strategy designed for clients with diverse computational and communication requirements and their local data. FedFA safeguards the global model against backdoor attacks through the layer grafting method. Furthermore, it introduces a scalable aggregation method to manage scale variations among networks of differing complexities. Compared to previous heterogeneous network aggregation methods, FedFA has shown superior testing performance and robustness to backdoor attacks, establishing its feasibility as a solution for various FL applications.

Acknowledgements

We thank A. Datta and P. Mardziel for access to computing resources for completing our experiments. This work was partially supported by the National Science Foundation under grants CNS-1751075, CNS-2106891, and CNS-2312761.

References

1. Abad, G., Paguada, S., Ersoy, O., Picek, S., Ramírez-Durán, V.J., Urbieta, A.: Sniper backdoor: Single client targeted backdoor attack in federated learning. In: 2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML). pp. 377–391. IEEE (2023)
2. Antunes, R.S., André da Costa, C., Küderle, A., Yari, I.A., Eskofier, B.: Federated learning for healthcare: Systematic review and architecture proposal. ACM Transactions on Intelligent Systems and Technology (TIST) **13**(4), 1–23 (2022)
3. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: International conference on artificial intelligence and statistics. pp. 2938–2948. PMLR (2020)
4. Demertzis, K., Kikiras, P., Skianis, C., Rantos, K., Iliadis, L., Stamoulis, G.: Federated auto-meta-ensemble learning framework for ai-enabled military operations. Electronics **12**(2), 430 (2023)
5. Deng, Y., Chen, W., Ren, J., Lyu, F., Liu, Y., Liu, Y., Zhang, Y.: Tailorfl: Dual-personalized federated learning under system and data heterogeneity. In: Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems. pp. 592–606 (2022)
6. Diao, E., Ding, J., Tarokh, V.: Heterofl: Computation and communication efficient federated learning for heterogeneous clients. arXiv preprint arXiv:2010.01264 (2020)

7. Duan, Q., Hu, S., Deng, R., Lu, Z.: Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions. Sensors **22**(16), 5983 (2022)
8. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010)
9. Greff, K., Srivastava, R.K., Schmidhuber, J.: Highway and residual networks learn unrolled iterative estimation. arXiv preprint arXiv:1612.07771 (2016)
10. Han, D.J., Bhatti, H.I., Lee, J., Moon, J.: Accelerating federated learning with split learning on locally generated losses. In: ICML 2021 workshop on federated learning for user privacy and data confidentiality. ICML Board (2021)
11. Hanin, B.: Which neural net architectures give rise to exploding and vanishing gradients? Advances in neural information processing systems **31** (2018)
12. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
14. 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: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1314–1324 (2019)
15. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. pmlr (2015)
16. Kang, H., Cha, S., Shin, J., Lee, J., Kang, J.: Nefl: Nested federated learning for heterogeneous clients. arXiv preprint arXiv:2308.07761 (2023)
17. Kumar, A., Yin, B., Shaikh, A.M., Ali, M., Wei, W.: Corrnet: pearson correlation based pruning for efficient convolutional neural networks. International Journal of Machine Learning and Cybernetics **13**(12), 3773–3783 (2022)
18. Li, G., Yang, Y., Bhardwaj, K., Marculescu, R.: Zico: Zero-shot nas via inverse coefficient of variation on gradients. arXiv preprint arXiv:2301.11300 (2023)
19. Li, L., Fan, Y., Tse, M., Lin, K.Y.: A review of applications in federated learning. Computers & Industrial Engineering **149**, 106854 (2020)
20. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. IEEE signal processing magazine **37**(3), 50–60 (2020)
21. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. Proceedings of Machine learning and systems **2**, 429–450 (2020)
22. Liu, F., Ren, X., Zhang, Z., Sun, X., Zou, Y.: Rethinking skip connection with layer normalization. In: Proceedings of the 28th international conference on computational linguistics. pp. 3586–3598 (2020)
23. Lyu, L., Yu, H., Yang, Q.: Threats to federated learning: A survey. arXiv preprint arXiv:2003.02133 (2020)
24. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
25. Mhaskar, H.N., Poggio, T.: Deep vs. shallow networks: An approximation theory perspective. Analysis and Applications **14**(06), 829–848 (2016)

26. Murshed, M.S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F.: Machine learning at the network edge: A survey. ACM Computing Surveys (CSUR) **54**(8), 1–37 (2021)
27. Nguyen, D.C., Ding, M., Pathirana, P.N., Seneviratne, A., Li, J., Poor, H.V.: Federated learning for internet of things: A comprehensive survey. IEEE Communications Surveys & Tutorials **23**(3), 1622–1658 (2021)
28. Noci, L., Li, C., Li, M.B., He, B., Hofmann, T., Maddison, C., Roy, D.M.: The shaped transformer: Attention models in the infinite depth-and-width limit. arXiv preprint arXiv:2306.17759 (2023)
29. Oh, S., Park, J., Vepakomma, P., Baek, S., Raskar, R., Bennis, M., Kim, S.L.: Locfedmix-sl: Localize, federate, and mix for improved scalability, convergence, and latency in split learning. In: Proceedings of the ACM Web Conference 2022. pp. 3347–3357 (2022)
30. Park, J., Yoon, D., Yeo, S., Oh, S.: Amble: Adjusting mini-batch and local epoch for federated learning with heterogeneous devices. Journal of Parallel and Distributed Computing **170**, 13–23 (2022)
31. Pfeiffer, K., Rapp, M., Khalili, R., Henkel, J.: Federated learning for computationally-constrained heterogeneous devices: A survey. ACM Computing Surveys (2023)
32. Ribero, M., Vikalo, H.: Communication-efficient federated learning via optimal client sampling. arXiv preprint arXiv:2007.15197 (2020)
33. Rodríguez-Barroso, N., Jiménez-López, D., Luzón, M.V., Herrera, F., Martínez-Cámara, E.: Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges. Information Fusion **90**, 148–173 (2023)
34. Ruan, Y., Zhang, X., Liang, S.C., Joe-Wong, C.: Towards flexible device participation in federated learning. In: International Conference on Artificial Intelligence and Statistics. pp. 3403–3411. PMLR (2021)
35. Samikwa, E., Di Maio, A., Braun, T.: Ares: Adaptive resource-aware split learning for internet of things. Computer Networks **218**, 109380 (2022)
36. Tan, M., Le, Q.: Efficientnetv2: Smaller models and faster training. In: International conference on machine learning. pp. 10096–10106. PMLR (2021)
37. Tolpegin, V., Truex, S., Gursoy, M.E., Liu, L.: Data poisoning attacks against federated learning systems. In: Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25. pp. 480–501. Springer (2020)
38. Turina, V., Zhang, Z., Esposito, F., Matta, I.: Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). pp. 250–260. IEEE (2021)
39. Vahidian, S., Morafah, M., Lin, B.: Personalized federated learning by structured and unstructured pruning under data heterogeneity. In: 2021 IEEE 41st international conference on distributed computing systems workshops (ICDCSW). pp. 27–34. IEEE (2021)
40. Veit, A., Wilber, M.J., Belongie, S.: Residual networks behave like ensembles of relatively shallow networks. Advances in neural information processing systems **29** (2016)
41. Wang, K., He, Q., Chen, F., Chen, C., Huang, F., Jin, H., Yang, Y.: Flexifed: Personalized federated learning for edge clients with heterogeneous model architectures. In: Proceedings of the ACM Web Conference 2023. pp. 2979–2990 (2023)

A Testing Specifications

Table 4. Details of the network architectures for Pre-ResNet, MobileNetV2, EfficientNetV2, and Transformer are presented.

	Pre-ResNet	MobileNetV2	EfficientNetV2	Transformer		
Section 1	$[3\times3, 64]3\times1$	$\begin{bmatrix}[3\times3,32]\\[3\times3,32]\\[1\times1,w_1]\end{bmatrix}\times1$	$[3\times3,24]\times1$	Encoder	Attention	192-d fc 64-d fc
Section 2	$\begin{bmatrix}[3\times3,w_1]\\[3\times3w_1]\end{bmatrix}\times d_1+1$	$\begin{bmatrix}[1\times1,6w_1]\\[3\times3,6w_1]\\[1\times1,w_2]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_2]\\[3\times3,6w_2]\\[1\times1,w_2]\end{bmatrix}\times1$	$\begin{bmatrix}[3\times3,24]\\[1\times1,w_1]\end{bmatrix}\times1$ $\begin{bmatrix}[3\times3,w_1]\\[1\times1,w_1]\end{bmatrix}\times1$		FeedForward	$\begin{bmatrix}[3\times3,64]\\[3\times3,64]\end{bmatrix}\times2$
Section 3	$\begin{bmatrix}[3\times3,w_2]\\[3\times3,w_2]\end{bmatrix}\times d_2+1$	$\begin{bmatrix}1\times1,6w_2\\3\times3,6w_2\\1\times1,w_3\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_3]\\[3\times3,6w_3]\\[1\times1,w_3]\end{bmatrix}\times d_1+1$	$\begin{bmatrix}[3\times3,4w_1]\\[1\times1,w_2]\end{bmatrix}\times1$ $\begin{bmatrix}[3\times3,4w_2]\\[1\times1,w_2]\end{bmatrix}\times d_1$		Attention	192-d fc 64-d fc
Section 4	$\begin{bmatrix}[3\times3,w_3]\\[3\times3,w_3]\end{bmatrix}\times d_3+1$	$\begin{bmatrix}[1\times1,6w_3]\\[3\times3,6w_3]\\[1\times1,w_4]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_4]\\[3\times3,6w_4]\\[1\times1,w_4]\end{bmatrix}\times d_2+2$	$\begin{bmatrix}[3\times3,4w_2]\\[1\times1,w_3]\end{bmatrix}\times1$ $\begin{bmatrix}[3\times3,4w_3]\\[1\times1,w_3]\end{bmatrix}\times d_2$		FeedForward	$\begin{bmatrix}[3\times3,64]\\[3\times3,64]\end{bmatrix}\times2$
Section 5	$\begin{bmatrix}[3\times3,w_4]\\[3\times3,w_4]\end{bmatrix}\times d_4+1$	$\begin{bmatrix}[1\times1,6w_4]\\[3\times3,6w_4]\\[1\times1,w_5]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_5]\\[3\times3,6w_5]\\[1\times1,w_5]\end{bmatrix}\times d_3+1$	$\begin{bmatrix}[1\times1,4w_3]\\[3\times3,4w_3]\\[1\times1,w_4]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,4w_4]\\[3\times3,4w_4]\\[1\times1,w_4]\end{bmatrix}\times d_3$			28782-d fc
Section 6	10-d fc	$\begin{bmatrix}[1\times1,6w_5]\\[3\times3,6w_5]\\[1\times1,w_6]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_6]\\[3\times3,6w_6]\\[1\times1,w_6]\end{bmatrix}\times d_4+1$	$\begin{bmatrix}[1\times1,6w_4]\\[3\times3,6w_4]\\[1\times1,w_5]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_5]\\[3\times3,6w_5]\\[1\times1,w_5]\end{bmatrix}\times(d_4+1)$	Decoder	Attention	192-d fc 64-d fc
Section 7		$\begin{bmatrix}[1\times1,6w_6]\\[3\times3,6w_6]\\[1\times1,w_7]\end{bmatrix}\times1$	$\begin{bmatrix}[1\times1,6w_5]\\[3\times3,6w_5]\\[1\times1,w_6]\end{bmatrix}\times1$ $\begin{bmatrix}[1\times1,6w_6]\\[3\times3,6w_6]\\[1\times1,w_7]\end{bmatrix}\times(d_5+3)$		FeedForward	$\begin{bmatrix}[3\times3,w_1]\\[3\times3,64]\end{bmatrix}\times d_1$
Section 8		$[1\times1,1280]\times1$ 100-d fc	$[1\times1,1792]\times1$ 10-d fc			192-d fc 64-d fc
						$\begin{bmatrix}[3\times3,w_1]\\[3\times3,64]\end{bmatrix}\times d_1$
				Classifier		28782-d fc

Table 5. The candidate values for the networks' depth d_k and width w_k in Table 4.

		Pre-ResNet	MobileNetV2	EfficientNetV2	Transformer
Depth	d_1	2, 3, 4, 5	2, 3, 4, 5	2, 3, 4, 5	2, 3, 4, 5
	d_2	2, 3, 4, 5	2, 3, 4, 5	2, 3, 4, 5	
	d_3	2, 3, 4, 5	2, 3, 4, 5	2, 3, 4, 5	
	d_4	2, 3, 4, 5	2, 3, 4, 5	2, 3, 4, 5	
	d_5			2, 3, 4, 5	
Width	w_1	64, 72, 80, 88	16, 24	24	64, 72, 80, 88
	w_2	128, 144, 160, 176	24, 32	24, 32	
	w_3	256, 288, 320, 352	32, 40, 48	48, 56, 64	
	w_4	512, 576, 640, 704	64, 70, 80, 88	64, 72, 80, 88	
	w_5		96, 112, 120, 136	128, 144, 160, 176	
	w_6		160, 184, 200, 224	160, 184, 200, 224	
	w_7		320, 360, 400, 440	256, 288, 320, 352	

Table 6. Test conditions for evaluating layer similarity in Section B, scale variations in Section F in the Appendix, and performance comparisons in Section 5.

Evaluation	**Similarity and Scale Variation Tests**			**Performance Comparison**						
				IID	**Non IID**	**IID**	**Non IID**	**IID**	**Non IID**	-
Number of clients	1			100		100		100		100
Fraction of active clients C	1			0.1						
Number of classes for each client	10	100	10	10	2	100	20	10	2	-
Number of samples for each client	50,000	50,000	60,000	100 ~ 250		250 ~ 500		100 ~ 250		15,360~20,480
Data	CIFAR10	CIFAR100	Fashion-MNIST	CIFAR10		CIFAR100		Fashion-MNIST		WikiText-2
Model	Pre-ResNet	MoblieNetV2	EffcientNetV2	Pre-ResNet		MoblieNetV2		EffcientNetV2		Transformer
Local epochs E	200	300	100	5		5		5		5
Local mini-batch size B	128			50						128
Communication Rounds	1			200	250	250	300	150	200	200
Optimizer	SGD									
Momentum	0.9									
Weight decay	1e-4			1e-4		1e-4		1e-4		Not applied
Learning rate η	0.1			0.01		0.01		0.01		0.1
Learning rate 0.1x decay schedule	[100, 150]	[150, 220]	[50, 75]	Not applied						Not applied
Batch normalization layer	Non-static			Static						

B Understanding Residual Blocks and Skip Connections in Convolutional Neural Networks

B.1 Structure and Properties of Convolutional Neural Networks with Skip Connections

Typical Convolutional Neural Network (CNN) architectures comprise a sequence of input layers, followed by sections of residual blocks, and concluding with output linear layers. A key feature in many such architectures is the use of skip connections, which connect layers across these residual blocks. The pivotal

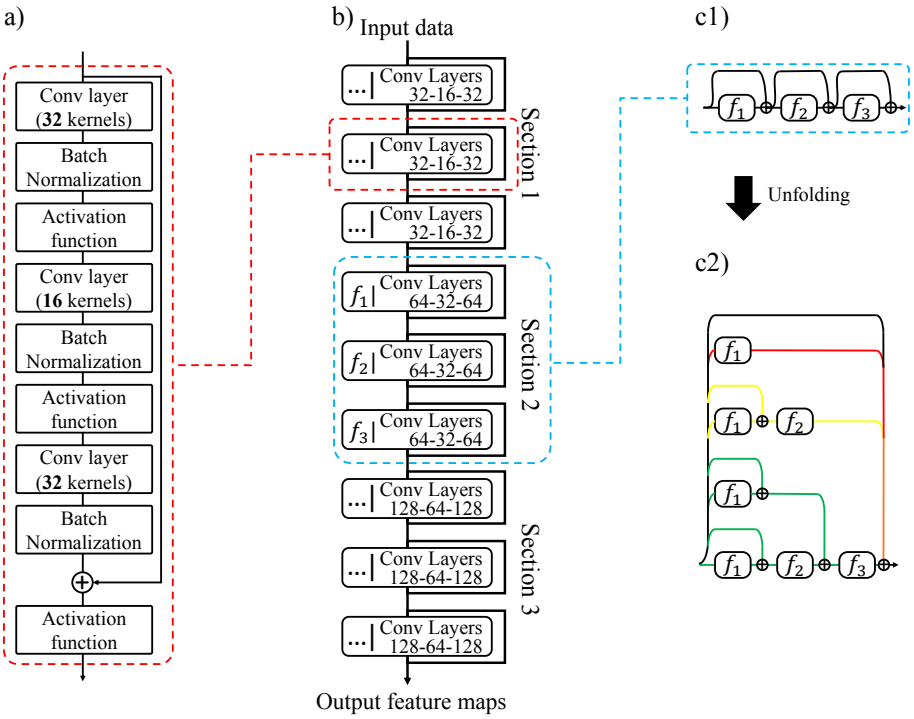

Fig. 4. a) A residual block in a CNN, including convolutional and batch normalization layers. b) A skip connection network with residual blocks. c1) Residual blocks from Section 2 of the network. c2) The unfolded network, highlighting how skip connections enable an ensemble-like system.

property of CNNs with skip connections, and the focus of our layer grafting approach, is that CNNs with skip connections exhibit residual blocks with similar weight values. This similarity makes it possible to graft the last residual blocks of each section, playing a crucial role in guiding the aggregation of local models

of different depths and informing the selection of layers during global model dissemination to local clients.

B.2 Residual Block Similarity and Its Implications

The similarity among residual blocks within a given section of a CNN can be empirically observed and has significant implications for model performance. Veit et al. [40]'s results suggest that CNNs' performance is not drastically affected by removing or swapping certain residual blocks. This observation is crucial in understanding the resilience of CNNs with skip connections and forms the basis of our layer grafting strategy.

Consider a skip connection network as shown in Figure 4 b), divided into sections with their residual blocks (f_x, for $x = 1, 2, 3$). Each block within a section typically employs a similar convolutional layer structure, and the model's output can be conceptualized as an average of results from various sub-model paths formed by these blocks.

For an input x, the output of a section with residual blocks can be expressed as:

$$\begin{aligned} output = x + f_1(x) + f_2(x) + f_3(x) + f_2(f_1(x)) \\ + f_3(f_1(x)) + f_3(f_2(x)) + f_3(f_2(f_1(x))) \end{aligned}$$

Swapping blocks, say f_1 and f_2, alters the equation but does not significantly affect the output, suggesting that $f_1(x)$ and $f_2(x)$ have similar contributions. This observation leads us to an important relationship:

$$f_2(f_1(x)) + f_3(f_2(f_1(x))) \approx f_1(f_2(x)) + f_3(f_1(f_2(x)))$$

Algebraic manipulation then brings us to the conclusion that $f_1 \approx f_2 \approx f_3$: *residual blocks within a given CNN are similar*. This reasoning can be generalized to larger numbers of residual blocks with various layer structures.

B.3 Statistical Evidence for Layer Similarity

We also statistically validate layer similarity. We consider the three different CNNs shown in Table 4: Pre-ResNet, MobileNetV2, and EfficientNetV2, which are respectively trained on the CIFAR-10, CIFAR-100, and Fashion MNIST datasets. We use models of different depths d_k, with d_k determining the number of residual blocks in each convolutional layer section, as shown in the table.

We do this by using the Pearson Correlation Coefficient (PCC). While the PCC has been used to assess correlations between filter weights, e.g., to facilitate model pruning [17], we directly utilize the PCC to gauge the similarity between convolutional layers and thus residual blocks. A high PCC value, combined with our previous result on similarities in weight scales, would indicate similarities in residual block weights.

To measure the similarity of two residual blocks, we first consider using the PCC to assess the similarities of two convolutional layers, $\mathbb{A}$ and $\mathbb{B}$, in distinct

residual blocks. Each layer contains N_{Cout} filters (i.e., output channels), with each filter possessing N_{Cin} weight maps (i.e., input channels). We use $r_{k,l}^{i,j}$ to represent the PCC value of the k-th weight map of the i-th filter of layer $\mathbb{A}$ and the l-th weight map of the j-th filter of layer $\mathbb{B}$.

We use $\mathbf{R}^{i,j}$ to denote the matrix of the PCCs for all pairs of weight maps in the i-th filter of layer $\mathbb{A}$ and the j-th filter of layer $\mathbb{B}$. Our goal is now to represent $\mathbf{R}^{i,j}$ with a single scalar encapsulating the overall similarities of filters i and j.

Since weights in networks are initialized randomly for every new training iteration, filter sequences can differ, even with identical input feature maps, leading to diverse output feature map sequences (Figure 5) [8,12]. Importantly, these outputs then serve as input for subsequent convolutional layers, influencing the sequence of weight maps within each filter.

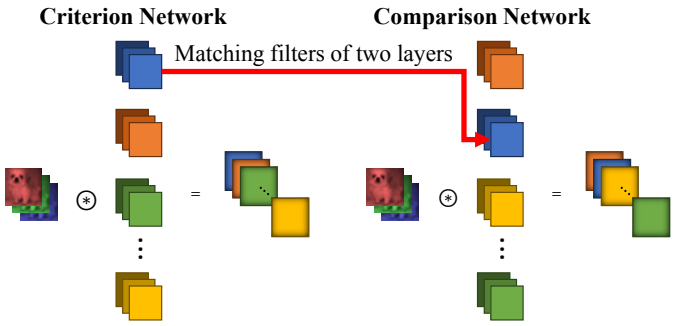

Fig. 5. Randomness in sequences of filters and weight maps in convolutional layers.

To match these sequences, we thus select the element with the highest PCC in each row of $\mathbf{R}^{i,j}$, with the constraint that only one element per column of $\mathbf{R}^{i,j}$ is selected. We then compute the average of the selected N_{Cin} elements, denoted as $\hat{r}^{ij}$. Since we need to account for potential overlapping weight maps, any column with a selected element is excluded in subsequent steps.

When assessing two convolutional layers with N_{Cout} filters each, there are N_{Cout}^2 filter pair combinations. Just as with weight map similarity, we create a one-to-one filter matching, since previous research has highlighted the existence of many similar filters within a single layer [17]. This one-to-one matching avoids matching a given filter with an excessive number of overlapping filters and overstating the overall layer similarity. We then average the PCCs $\hat{r}^{i,j}$ of the matched filters.

The similarities between the convolutional layers of two models with varying depths are presented in Tables 7, 8, and 9. We consider models at the 0 epoch (i.e., before training) and at the epoch where they achieve their highest testing accuracy. Specifically, we exclude the first residual block of each section from

our analysis, as it typically has a different input channel size compared to the other residual blocks in the same section.

From Tables 7, 8, and 9, all convolutional layers within a particular section have similarities greater than 0.5, regardless of the presence of skip connections. CNNs with skip connections usually show increased correlations post-training, with exceptions in 43 of the 138 cases (emphasized in bold). In contrast, CNNs without skip connections reveal decreased correlations post-training in 76 of the 138 cases (emphasized in bold).

Prior to training, high similarities are observed in both types of CNNs, suggesting potential matching of filters or weight maps, possibly due to the law of large numbers. This trend is even more evident as filter counts rise from lower to higher sections. Notably, while CNNs with skip connections mostly show an increase in similarity, over half the cases without them deviate from this trend.

This data shows that *the weights of residual blocks within the same section of CNNs with skip connections remain similar*, irrespective of depth or structural variations. We thus validate layer grafting to aggregate client models, as well as our method of sending model weight subsets to each client.

B.4 Layer Grafting Based on Residual Block Similarity

The observed similarity among residual blocks in CNNs allows us to infer that any block from the same section is a viable candidate for grafting. This similarity supports the layer grafting method in two ways:

- **Depth Modification**: When modifying the depth of the global model for local models, any residual block from the same section can be used, ensuring consistency in feature representation and learning capability.
- **Model Adaptability**: The similarity in blocks enables a more flexible approach to model aggregation and dissemination, as blocks can be added or removed without significantly impacting model performance.

B.5 Summary

The statistical analysis of the layer grafting method in CNNs with skip connections underscores the viability of this approach in FL. By leveraging the inherent similarity among residual blocks, the layer grafting method not only preserves model integrity but also enhances adaptability and robustness across diverse client models. This analysis affirms the soundness of layer grafting as a key component in our FedFA framework in Figure 6.

Table 7. Similarities of convolutional layers for Pre-ResNet.

a) Pre-ResNet (CIFAR10)

Section 2			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.661	0.661	0.660	0.668	0.666	0.666
		Non-Skip	0.661	0.659	0.662	0.706	0.719	0.707
d_k=2	Block2	Skip	0.660	0.660	0.660	0.668	0.671	**0.656**
		Non-Skip	0.660	0.660	0.663	0.701	0.704	0.694
	Block3	Skip	0.661	0.663	0.659	**0.658**	0.671	0.665
		Non-Skip	0.658	0.660	0.660	0.722	0.736	0.743

Section 4			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.770	0.769	0.770	0.802	0.801	0.798
		Non-Skip	0.770	0.769	0.770	**0.693**	**0.563**	**0.497**
d_k=2	Block2	Skip	0.770	0.770	0.769	0.803	0.801	0.797
		Non-Skip	0.769	0.769	0.770	**0.758**	**0.579**	**0.509**
	Block3	Skip	0.770	0.770	0.769	0.805	0.807	0.804
		Non-Skip	0.769	0.769	0.769	**0.731**	**0.558**	**0.528**

Section 3			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.719	0.720	0.719	0.756	0.756	0.759
		Non-Skip	0.721	0.719	0.720	0.778	0.730	0.651
d_k=2	Block2	Skip	0.719	0.719	0.719	0.758	0.757	0.760
		Non-Skip	0.720	0.720	0.719	0.782	0.735	0.653
	Block3	Skip	0.719	0.719	0.720	0.754	0.754	0.757
		Non-Skip	0.721	0.720	0.720	0.801	0.794	0.728

Section 5			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.811	0.811	0.811	0.832	0.834	0.834
		Non-Skip	0.811	0.811	0.811	**0.447**	**0.538**	**0.731**
d_k=2	Block2	Skip	0.811	0.811	0.811	0.839	**0.796**	**0.802**
		Non-Skip	0.811	0.811	0.811	**0.577**	**0.615**	**0.681**
	Block3	Skip	0.811	0.811	0.811	0.831	**0.810**	0.822
		Non-Skip	0.811	0.811	0.811	**0.517**	**0.626**	**0.739**

Table 8. Similarities of convolutional layers for MobileNetV2.

b) MobileNetV2 (CIFAR100)

Section 3			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.732	0.726	0.738	0.783	0.754	0.749
		Non-Skip	0.729	0.739	0.729	0.785	0.781	0.751
d_k=2	Block2	Skip	0.730	0.731	0.734	0.794	0.784	0.767
		Non-Skip	0.727	0.720	0.729	0.789	0.783	**0.723**
	Block3	Skip	0.726	0.735	0.743	0.747	0.754	0.764
		Non-Skip	0.732	0.731	0.721	0.776	0.771	0.738

Section 5			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.801	0.802	0.799	0.839	0.827	0.835
		Non-Skip	0.803	0.807	0.801	**0.763**	**0.680**	**0.662**
d_k=2	Block2	Skip	0.805	0.794	0.800	0.856	0.828	0.843
		Non-Skip	0.797	0.806	0.806	**0.790**	**0.743**	**0.711**
	Block3	Skip	0.807	0.803	0.793	0.825	0.835	0.846
		Non-Skip	0.808	0.799	0.808	**0.792**	**0.777**	**0.741**

Section 4			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.777	0.781	0.781	0.813	0.797	0.798
		Non-Skip	0.770	0.783	0.776	0.790	**0.705**	**0.705**
	Block3	Skip	0.782	0.776	0.776	0.809	0.796	0.797
		Non-Skip	0.781	0.773	0.777	0.784	**0.760**	**0.776**
d_k=2	Block2	Skip	0.783	0.787	0.780	0.807	0.800	0.794
		Non-Skip	0.776	0.791	0.776	0.836	**0.745**	**0.719**
	Block3	Skip	0.786	0.783	0.778	0.816	0.821	0.796
		Non-Skip	0.778	0.783	0.784	0.786	0.822	0.807
	Block4	Skip	0.778	0.781	0.772	0.803	0.817	0.819
		Non-Skip	0.770	0.772	0.782	0.781	0.788	0.816

Section 6			0 epoch / d_k=3			Best epoch / d_k=3		
			Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip	0.834	0.830	0.831	0.881	0.869	0.857
		Non-Skip	0.830	0.831	0.832	**0.715**	**0.760**	**0.743**
d_k=2	Block2	Skip	0.832	0.828	0.828	0.883	0.851	0.834
		Non-Skip	0.835	0.830	0.832	**0.728**	**0.818**	**0.728**
	Block3	Skip	0.834	0.827	0.835	0.859	0.879	0.874
		Non-Skip	0.838	0.831	0.829	**0.732**	**0.748**	**0.759**

Table 9. Similarities of convolutional layers for EfficientNetV2.

c) EfficientNetV2 (Fashion MNIST)

Section 3				0 epoch / d_k=3			Best epoch / d_k=3		
				Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip		0.643	0.644	0.644	**0.624**	**0.630**	**0.634**
		Non-Skip		0.643	0.645	0.643	**0.604**	**0.600**	**0.600**
d_k=2	Block2	Skip		0.644	0.644	0.643	**0.616**	**0.624**	**0.625**
		Non-Skip		0.643	0.643	0.643	**0.630**	**0.619**	**0.618**
	Block3	Skip		0.643	0.642	0.642	**0.620**	**0.628**	**0.630**
		Non-Skip		0.644	0.642	0.642	**0.636**	**0.630**	**0.633**

Section 4				0 epoch / d_k=3			Best epoch / d_k=3		
				Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip		0.668	0.668	0.668	**0.652**	**0.656**	**0.659**
		Non-Skip		0.667	0.668	0.668	**0.629**	**0.640**	**0.635**
d_k=2	Block2	Skip		0.668	0.667	0.668	**0.652**	**0.655**	**0.656**
		Non-Skip		0.667	0.667	0.667	**0.637**	**0.641**	**0.640**
	Block3	Skip		0.667	0.668	0.668	**0.654**	**0.657**	**0.658**
		Non-Skip		0.668	0.667	0.668	**0.643**	**0.650**	**0.650**

Section 5				0 epoch / d_k=3			Best epoch / d_k=3		
				Block2	Block3	Block4	Block2	Block3	Block4
d_k=1	Block2	Skip		0.792	0.789	0.794	0.818	0.826	0.822
		Non-Skip		0.797	0.804	0.801	0.800	**0.779**	**0.749**
d_k=2	Block2	Skip		0.798	0.787	0.791	0.817	0.823	0.830
		Non-Skip		0.794	0.789	0.793	0.795	**0.778**	**0.781**
	Block3	Skip		0.795	0.788	0.794	0.814	0.818	0.820
		Non-Skip		0.797	0.792	0.804	**0.780**	0.799	**0.793**

Section 6				0 epoch / d_k=3			Best epoch / d_k=3		
				Block2	Block3	Block5	Block2	Block3	Block5
d_k=1	Block2	Skip		0.833	0.829	0.832	**0.823**	0.841	**0.826**
		Non-Skip		0.833	0.830	0.828	**0.820**	**0.824**	0.831
	Block3	Skip		0.829	0.830	0.827	**0.828**	**0.826**	**0.824**
		Non-Skip		0.829	0.832	0.829	**0.805**	**0.816**	0.830
d_k=2	Block2	Skip		0.827	0.827	0.826	0.831	0.827	0.827
		Non-Skip		0.830	0.832	0.831	0.831	0.837	0.834
	Block3	Skip		0.833	0.830	0.831	**0.822**	**0.827**	**0.826**
		Non-Skip		0.827	0.833	0.833	**0.823**	**0.821**	0.840
	Block4	Skip		0.831	0.831	0.833	**0.826**	0.839	0.835
		Non-Skip		0.832	0.833	0.832	**0.806**	**0.806**	0.837

Section 7				0 epoch / d_k=3			Best epoch / d_k=3		
				Block2	Block5	Block7	Block2	Block5	Block7
d_k=1	Block2	Skip		0.851	0.852	0.850	0.934	**0.741**	**0.728**
		Non-Skip		0.853	0.851	0.851	0.974	0.939	0.833
	Block3	Skip		0.854	0.853	0.851	**0.847**	**0.847**	**0.840**
		Non-Skip		0.853	0.852	0.851	0.960	0.971	0.865
	Block5	Skip		0.853	0.856	0.851	**0.752**	0.874	0.885
		Non-Skip		0.851	0.853	0.850	0.922	0.971	0.904
d_k=2	Block2	Skip		0.854	0.850	0.851	0.976	**0.692**	**0.673**
		Non-Skip		0.849	0.850	0.851	0.930	0.978	0.896
	Block4	Skip		0.852	0.854	0.853	**0.775**	0.863	0.877
		Non-Skip		0.849	0.852	0.850	0.977	0.952	**0.847**
	Block6	Skip		0.852	0.851	0.851	**0.669**	**0.839**	0.853
		Non-Skip		0.852	0.852	0.850	0.954	0.977	0.873

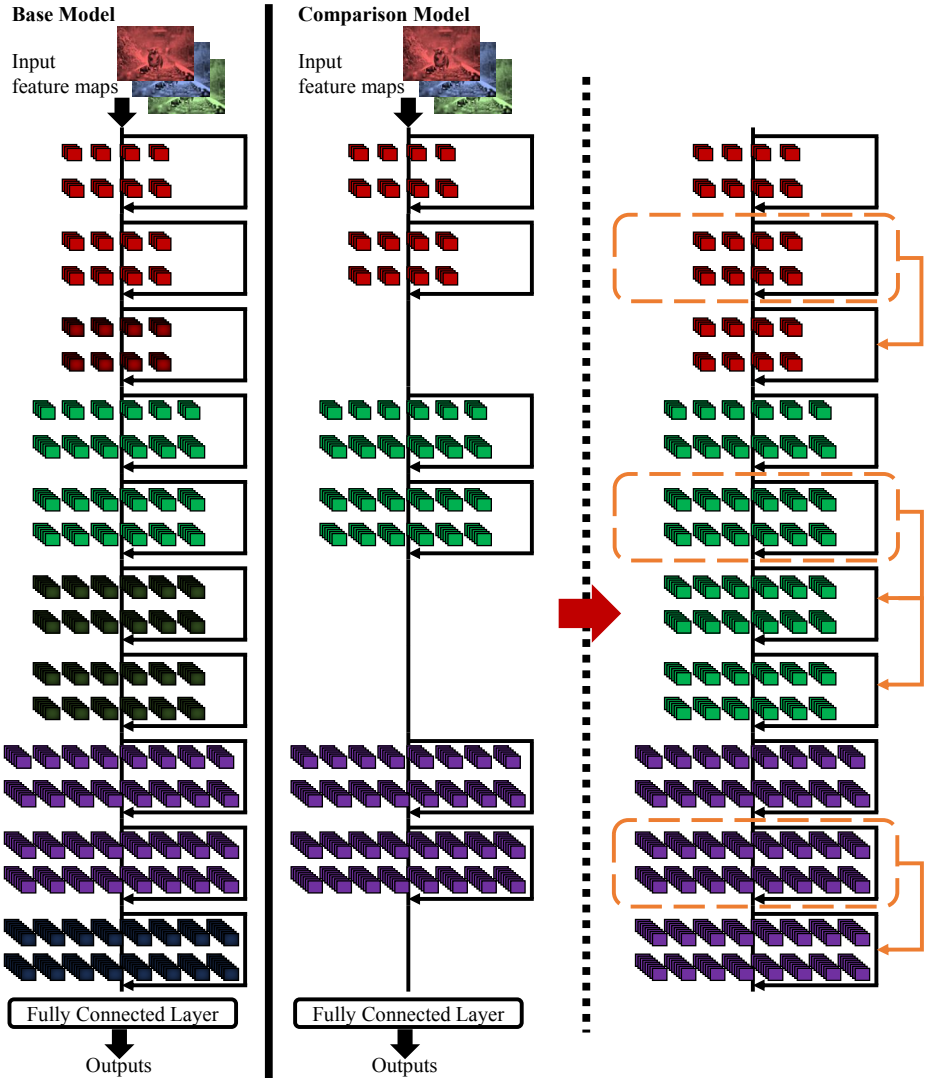

Fig. 6. The layer grafting method to standardize model depth by identifying the maximum depths of sections and augmenting each client's model with additional residual blocks until this depth is reached.

C Global Model distribution Algorithm

Algorithm 3 This algorithm customizes the aggregated global model M_G^t for each client c by adjusting its structure to align with its original local network architecture, M_c^{t-1} (line 7 of Algorithm 1). The global model has the number of residual blocks for section s, $N_{depth,\max}^{(s)}$, whereas the local model, M_c^{t-1}, has the number of residual blocks for each section, $N_{depth,c}^{(s)}$ and the input and the output channel sizes for of each layer, C_I and C_o. Customization involves reducing the depth and width of M_G by systematically removing residual blocks and filters over the client-specific thresholds. In the algorithm, the $\ominus$ operator signifies the reverse of the layer grafting process.

Require: Updated global model M_G^t.
Ensure: Each client c receives an appropriately configured version of M_G^t, M_c^t.

for each section $M_G^{(s)}$ in global model M_G^t **do**
  $\Delta D \leftarrow N_{depth,\max}^{(s)} - N_{depth,c}^{(s)}$
  for $d = 1$ to ΔD **do**
    $R_{last}^{(s)} \leftarrow$ last residual block in section s
    $M_G^{(s)} \leftarrow M_G^{(s)} \ominus R_{last}^{(s)}$
  end for
end for
for each layer $M_G^{(l)}$ in global model M_G^t **do**
  $C_I, C_o \leftarrow$ Input and output channel sizes of $M_c^{(l)}$
  $M_G^{(s)} \leftarrow M_G^{(s)}[: C_o, : C_I]$
end for
$M_c^t \leftarrow M_G^t$

D Efficacy of Heterogeneous Network Aggregation

This section investigates the theoretical underpinnings of heterogeneous network aggregation. Due to their simpler structure, shallow networks demonstrate a faster convergence rate than deeper networks. However, deeper networks are more adept at capturing complex features and hierarchical data structures, translating to superior performance on complex tasks [25]. Therefore, aggregating shallow and deep models can accelerate convergence relative to only deep models and enhance performance compared to solely shallow models. This theoretical examination focuses on how the speed of increase in prediction variance (the output logits of the classifier) differs between shallow and deep models in classification tasks.

D.1 Variance Analysis Across Model Complexities

When a model is fully trained in classification tasks, its output logits typically approach 1s for the correct classes and 0s for others, assuming one-hot encoding. This indicates that the variance of the output logits generally increases until the models are completely adapted to their data. This subsection explores the relationship between model complexity and variance in model predictions for classification tasks, analyzing models of varying complexities to discern their impact on predictive variability.

Preliminaries and Assumptions To guide our analysis, we define essential concepts and assumptions:

- **Model Output Functionality**: In classification tasks, consider the model's output logits vector, y, is a function of the input x and weight vectors w_i for the index of each layer i. We assume that the activation function g negates the correlated outputs of w_i for analytical simplicity. Additionally, appropriately clipped weights w_i can compute the output y without needing a softmax function. Thus, y can be expressed as a linear combination of $g(w_i, x)$, formulated as $y = \sum_i g(w_i, x)$. Also, the analysis assumes the optimization process employs full batch gradient descent.
- **Law of Large Numbers (LLN) Application**: The LLN indicates that as the number of trials increases, the average of the outcomes converges to the expected value. This principle is instrumental in understanding the variance behavior with increasing model complexity.

Model Complexity and Variance We explore the relationship between model complexity (number of weights) and variance in a structured manner. The total variance of y is articulated through the law of total variance:

$$\mathrm{Var}(y) = E[\mathrm{Var}(y|x)] + \mathrm{Var}(E[y|x])$$

Here, $E[\text{Var}(y|x)]$ signifies the expected value of the conditional variance of y given x, and $\text{Var}(E[y|x])$ represents the variance of the expected value of y given x.

Analyzing each component, assuming $g(w_i, x)$ is independent for each i:
1. Expectation of Conditional Variance $E[\text{Var}(y|x)]$

$$\begin{aligned} E[\text{Var}(y|x)] &= E[\text{Var}(\sum_i g(w_i, x)|x)] \\ &= E[\sum_i \text{Var}(g(w_i, x)|x)] + 2E[\sum_{i \neq j} \text{Cov}(g(w_i, x), g(w_j, x)|x)] \\ &= E[\sum_i \text{Var}(g(w_i, x)|x)] + 0 = \sum_i E[\text{Var}(g(w_i, x)|x)] \end{aligned}$$

2. Variance of the Expected Value $\text{Var}(E[y|x])$

$$\text{Var}(E[y|x]) = \text{Var}\left(E[\sum_i g(w_i, x)|x]\right) = \text{Var}\left(\sum_i E[g(w_i, x)|x]\right)$$

Given that $g(w_i, x)$ is independent for each i:

$$\text{Var}(E[y|x]) = \sum_i \text{Var}\left(E[g(w_i, x)|x]\right)$$

Combining both components yields:

$$\text{Var}(y) = \sum_i E[\text{Var}(g(w_i, x)|x)] + \sum_i \text{Var}\left(E[g(w_i, x)|x]\right)$$

Averaging Effect of Weights The LLN posits that as the number of weights n increases, the individual contributions to the output variance by each weight, $\text{Var}(g(w_i, x)|x)$, average out. This results in a reduction of the overall variance attributable to any single weight.

Comparative Analysis of Deep and Shallow Models Consider a deep model with a large number of weights (N_{deep}) and a shallow model with fewer weights (N_{shallow}). Assuming N_{shallow} is sufficient for the LLN to apply, we define the contribution of each weight to the output logits' variance for deep and shallow models as Var_{deep} and Var_{shallow}, respectively:

$$\begin{aligned} \text{Var}_{\text{deep}} &= \frac{1}{N_{\text{deep}}} \left(\sum_{i=1}^{N_{\text{deep}}} E[\text{Var}(g(w_i, x)|x)] + \sum_{i=1}^{N_{\text{deep}}} \text{Var}(E[g(w_i, x)|x]) \right) \\ &= \frac{1}{N_{\text{deep}}} \text{Var}(y) \end{aligned}$$

$$\begin{aligned}\mathrm{Var}_{\mathrm{shallow}} &= \frac{1}{N_{\mathrm{shallow}}}\left(\sum_{i=1}^{N_{\mathrm{shallow}}} E[\mathrm{Var}(g(w_i,x)|x)] + \sum_{i=1}^{N_{\mathrm{shallow}}} \mathrm{Var}(E[g(w_i,x)|x])\right) \\ &= \frac{1}{N_{\mathrm{shallow}}}\mathrm{Var}(y)\end{aligned}$$

For a given target variance $\mathrm{Var}(y)$, and considering $N_{\mathrm{deep}} > N_{\mathrm{shallow}}$, the LLN implies that the average variance attributable to individual weights in the deep model is less than that in the shallow model:

$$\frac{1}{N_{\mathrm{deep}}}\mathrm{Var}(y) \leq \frac{1}{N_{\mathrm{shallow}}}\mathrm{Var}(y)$$

Consequently:

$$\mathrm{Var}_{\mathrm{deep}} \leq \mathrm{Var}_{\mathrm{shallow}}$$

D.2 Post-Training Variance Dynamics

This section delves into the increase in the variance of output logits post-training, reflecting a shift from initial model outputs to a more diversified set of predictions due to learning from the training data.

Foundational Premises

- **Weight Initialization**: Before training, weights w_i are uniformly initialized (e.g., all ones), leading to a deterministic output for any given input x, assuming w_i remains fixed. Nevertheless, in a more complex model where the output nonlinearly depends on w_i, even with uniform initialization, there exists a potential for non-zero variance in the output if the model's function $\sum_i g(w_i, x)$ maps the same w_i to varying outputs for different inputs x.
- **Training Dynamics**: During training, w_i is adjusted to minimize a loss function $L(y, y^*)$, where y^* represents the target output vector. This adjustment adheres to the full batch gradient descent methodology at iteration t, represented as:

  $$w_{i,t+1} = w_{i,t} + \eta \nabla L(y_t, y^*)$$

  Here, η is the learning rate.

Pre-Training Variance Dynamics Before training commences, all weights w_i (i.e., $w_{i,0}$) are initialized to a uniform value, leading to a theoretical scenario where the model's output variance might show minimal but non-zero values due to the differential mapping of inputs x by $\sum_i g(w_{i,0}, x)$. The initial assumption is that:

$$\mathrm{Var}(y_0) = \mathrm{Var}(\sum_i g(w_{i,0}, x)) \approx 0$$

In a simple or linear model where f represents the model function, $\mathrm{Var}(y_0)$ could be nearly zero if $\sum_i g(w_{i,0}, x)$ yields consistent outputs. However, this variance may not be negligible for more complex models, though it remains relatively small.

Post-Training Variance Dynamics After training, the weights w_i are updated to minimize the loss function L, introducing variability in $w_{i,t+1}$ which, in turn, injects diversity into the model's output:

$$w_{i,t+1} = w_{i,t} + \eta \nabla L(y_t, y^*) \quad \text{and} \quad L(y_{t+1}, y^*) \leq L(y_t, y^*)$$

As $w_{i,t+1}$ is continually adjusted to reduce L, reflecting the learning process from the training data, the variance $\mathrm{Var}(y_{t+1})$ tends to increase, acknowledging the diversity in responses due to this learned variability:

$$\mathrm{Var}(y_{t+1}) \geq \mathrm{Var}(y_t)$$

for the training data x.

D.3 Toward a Specific Target Variance

This subsection systematically examines how the weights of deep and shallow models affect the speed toward a specific target variance $\mathrm{Var}_{\mathrm{target}}$ and finally shows that aggregating deep and shallow models is beneficial.

From earlier discussions:

$$\frac{1}{N_{\mathrm{shallow}}}\mathrm{Var}(y_t) = \mathrm{Var}_{\mathrm{shallow},t} \geq \mathrm{Var}_{\mathrm{deep},t} = \frac{1}{N_{\mathrm{deep}}}\mathrm{Var}(y_t)$$

$$\mathrm{Var}(y_{t+1}) \geq \mathrm{Var}(y_t)$$

We know from the given conditions that the accumulated variances for shallow and deep models equalize at certain times T_{shallow} and T_{deep} for y^*, respectively:

$$\mathrm{Var}(y^*) = N_{\mathrm{shallow}}\mathrm{Var}_{\mathrm{shallow},T_{\mathrm{shallow}}} = N_{\mathrm{deep}}\mathrm{Var}_{\mathrm{deep},T_{\mathrm{deep}}}$$

Let r_{shallow} and r_{deep} represent the average rates of variance growth for shallow and deep models, respectively. These rates are defined as the change in variance per training iteration. Over time, the variances for shallow and deep models can be represented as a function of their growth rates and time:

$$\mathrm{Var}_{\mathrm{shallow},T_{\mathrm{shallow}}} = r_{\mathrm{shallow}}T_{\mathrm{shallow}}$$

$$\mathrm{Var}_{\mathrm{deep},T_{\mathrm{deep}}} = r_{\mathrm{deep}}T_{\mathrm{deep}}$$

Plugging these into the equation of accumulated variances gives:

$$N_{\mathrm{shallow}}r_{\mathrm{shallow}}T_{\mathrm{shallow}} = N_{\mathrm{deep}}r_{\mathrm{deep}}T_{\mathrm{deep}}$$

To find a direct relationship between r_{shallow} and r_{deep}, rearrange the equation:

$$r_{\text{shallow}} = \frac{N_{\text{deep}} T_{\text{deep}}}{N_{\text{shallow}} T_{\text{shallow}}} r_{\text{deep}}$$

and knowing $T_{\text{deep}} \geq T_{\text{shallow}}$ [25], Therefore, we can infer that:

$$r_{\text{shallow}} \geq r_{\text{deep}}$$

This implies that shallow models exhibit a faster increase in variance compared to deep models, illustrating that the variance in predictions of a shallow model increases more quickly during training.

Aggregation Dynamics of Deep and Shallow Model Weights The aggregation of weights from both deep and shallow models creates a new dynamic in the variance increase rate of the combined model. This can be quantified as:

$$r_{\text{combined}} = \zeta r_{\text{deep}} + (1 - \zeta) r_{\text{shallow}} \geq r_{\text{deep}}$$

where ζ is a factor that balances the contributions of deep and shallow model weights with $0 < \zeta < 1$. This formula shows that combining the weights of deep and shallow models in the combined model configuration yields a more marked increase in the variance rate per weight compared to using only the deep model's weights.

E Convergence Analysis of Gradient Aggregation in Federated Learning with Flexible Architectures

In this section, we show the convergence rate of our FedFA's object function in the context of FL [24].

Preliminary Concepts

1. **Mitigation of Skewness Introduced by Data Heterogeneity**: Different client local weights and data distributions, denoted as $\omega_1, \omega_2, \ldots, \omega_c$ and $D_1, D_2, \ldots, D_c$, yield gradients $\nabla f_1, \nabla f_2, \ldots, \nabla f_c$ that can vary significantly in magnitude. Formally:

$$\nabla f_c(\omega_c, D_c) = \frac{\partial}{\partial \omega_c} f_c(\omega_c, D_c)$$

   To counteract this variability, we employ the L2 norm for scaling. The L2 norm is calculated based on the central 95% of the weights under the 95th percentile:

$$\nabla f_c^{\text{scaled}} = \left(\frac{\frac{1}{m} \sum_{\kappa \in \mathcal{C}_{sel}} \left\| \omega_{95\%,\kappa} \right\|}{\left\| \omega_{95\%,c} \right\|} \right) \nabla f_c(\omega_c, D_c) = \alpha_c \nabla f_c(\omega_c, D_c)$$

   Unlike the FedFA's complete Algorithm 1, we assume ω_c is a one-layered neural network architecture for simplicity. Let $|\mathcal{C}|$ symbolize the total client count within the FL framework. C is the participating rate of clients for each round. $\mathcal{C}_{sel}$ indicates the client indices participating in a given round, and m is the count of these clients, such that $m = |C_{sel}|$. It is ensured that $m = C \times |\mathcal{C}|$.
2. **Preservation of Gradient Direction**: After scaling, the gradient's direction remains unchanged:

$$\text{Direction}(\nabla f_c^{\text{scaled}}) = \text{Direction}(\nabla f_c)$$

   This conservation guarantees that the unique insights from each client's local data are maintained post-scaling.
3. **Robustness to Outliers**: Utilizing the central weights under the 95th percentile for scaling ensures resilience against extreme gradient values:

$$\omega_{95\%,c} = \text{Central 95\% of the weights}$$

   This mechanism normalizes potential outliers, preventing them from disproportionately influencing the global update.
4. **Global Model Stability**: By aggregating the scaled gradients across all clients, we derive the global gradient as:

$$\nabla f_{G,\text{w/ scaled}} = \sum_{c \in C_{sel}} p_c \times \nabla f_c^{\text{scaled}}$$

   Here, p_c denotes weights, typically reflecting the data distribution of client c.

Convergence Analysis To understand the convergence of our algorithm in federated learning, we analyze it under two primary mathematical properties: strong convexity and Lipschitz continuity of the gradient.

Assumptions

1. The global loss function $f(\omega)$ is μ-strongly convex, where μ is a positive real number, ω is weights. This property guarantees a unique minimum for $f(\omega)$, which aids in the convergence of the optimization process.
2. The gradient of $f(\omega)$ exhibits Lipschitz continuity with a constant denoted as L. Given that L is a non-negative real number, this continuity ensures that the gradient variations between consecutive iterations are bounded, ensuring stability during the optimization updates.

Analysis If a function $f(\omega)$ is μ-strongly convex, then for every ω_1 and ω_2:

$$f(\omega_1) \geq f(\omega_2) + \langle \nabla f(\omega_2), \omega_1 - \omega_2 \rangle + \frac{\mu}{2}\|\omega_1 - \omega_2\|^2$$

The function of $f(\omega)$ is L-Lipschitz continuous when:

$$\|f(\omega_1) - f(\omega_2)\| \leq L\|\omega_1 - \omega_2\|$$

The iterative process of full batch gradient descent updates the weight ω as:

$$\omega_{t+1} = \omega_t + \eta \nabla f(\omega_t)$$

Here, η is the learning rate. Thus, we can deduce:

$$\|\omega_{t+1} - \omega_t\| = \eta \|\nabla f(\omega_t)\|$$

We start with the strong convexity property:

$$\begin{aligned} f(\omega_{t+1}) &\geq f(\omega_t) + \langle \nabla f(\omega_t), \eta \nabla f(\omega_t) \rangle + \frac{\mu}{2}\|\eta \nabla f(\omega_t)\|^2 \\ &= f(\omega_t) + \eta \|\nabla f(\omega_t)\|^2 + \frac{\mu \eta^2}{2}\|\nabla f(\omega_t)\|^2 \end{aligned}$$

Considering the Lipschitz continuity, the function value change due to a step in the direction of the gradient is:

$$\|f(\omega_{t+1}) - f(\omega_t)\| \leq L\|\eta \nabla f(\omega_t)\|$$

Combining both inequalities, we get:

$$\begin{aligned} \|\eta \|\nabla f(\omega_t)\|^2 + \frac{\mu \eta^2}{2}\|\nabla f(\omega_t)\|^2\| &\leq \eta L \|\nabla f(\omega_t)\| \\ \frac{2 + \mu\eta}{2}\|\nabla f(\omega_t)\| &\leq L \\ \|\nabla f(\omega_t)\| &\leq \frac{2L}{2 + \mu\eta} \end{aligned}$$

This inequality gives us a bound on the magnitude of the gradient at iteration t. If the magnitude of the gradient decreases (or remains below a certain threshold), this indicates convergence towards an optimum.

To understand the convergence properties of the FedFA framework, let's break down the gradient's behavior and its associated norms. Given the global gradient at iteration t as:

$$\|\nabla f^t_{G,\text{w/ scaled}}\|$$

We can express it as an aggregation of the scaled gradients from each client:

$$\|\nabla \left(\sum_{c \in C_{sel}} p_c \times \nabla f_c^{t,\text{scaled}}) \right)\|$$

The scaling factor α_c^t alters the gradient norm:

$$\|\nabla f^t_{G,\text{w/ scaled}}\| \leq \max_{c \in C_{sel}}\{\alpha_c^t\} \times \|\nabla f^t_{G,\text{w/o scaled}}\|$$

Using the bound from the unscaled case:

$$\|\nabla f^t_{G,\text{w/ scaled}}\| \leq \max_{c \in C_{sel}}\{\alpha_c^t\} \times \frac{2L}{2+\mu\eta}$$

With the scaled full batch gradient descent update, the difference becomes:

$$\|\omega_{t+1} - \omega_t\| = \eta \|\nabla f^t_{G,\text{w/ scaled}}\|$$

Substituting the bound for the scaled gradient norm:

$$\|\omega_{t+1} - \omega_t\| \leq \eta \times \max_{c \in C_{sel}}\{\alpha_c^t\} \times \frac{2L}{2+\mu\eta}$$

Therefore, the convergence rate is:

$$\mathcal{O}\left(\max_{c \in C_{sel}} \left\{ \frac{\frac{1}{m}\sum_{\kappa \in C_{sel}} \left\|\omega_{95\%,\kappa}\right\|}{\left\|\omega_{95\%,c}\right\|} \right\} \times \frac{2L\eta}{2+\mu\eta} \right)$$

This suggests that the convergence of FedFA is sensitive to the learning rate, scaling factors, and the inherent attributes of the loss function. It is crucial to recognize that the convergence rate usually fluctuates over time based on the selection of the learning rate η, which could be either constant or adaptive, tending towards zero as the number of iterations t increases.

F Scale Variations According to Heterogeneous Architectures

F.1 Introduction to Scale Variations in Federated Learning

In FL, fair aggregation is challenging when local models' scales differ, a situation exacerbated by employing heterogeneous network architectures. This implies the necessity for scalable aggregation techniques. In this section, we empirically demonstrate scale variations across heterogeneous network architectures.

F.2 Empirical Analysis

We employed three types of architectures: Pre-ResNet, MobileNetV2, and EfficientNetV2, varying in depth and width as depicted in Table 4. Detailed training conditions are in Table 6. To determine the depth d_k and width w_k, we refer to the configuration in the second column of Table 10.

Visualization of Weight Distributions For each model type, from the Baseline Model and Models 1 to 6, we flattened and vectorized all weights in the first and last convolutional layers to visualize the weight distributions as shown in Figures 7, 8, and 9. These figures reveal unique weight distributions for each model, corresponding to their complexity.

Quantifying Scale Variations We set the Baseline model M to quantify scale variations with the minimum depth and width. In a single convolutional layer l in the baseline model, there are $N_{out}^{M^{(l)}}$ filters, each composed of $N_{In}^{M^{(l)}}$ weight maps $M_i^{(l)}$, where i is the index of each weight map.

We first examine the average magnitude:

$$\text{Average Magnitude} = \frac{1}{N_{out}^{M^{(l)}} \times N_{In}^{M^{(l)}}} \sum_i \|M_i^{(l)}\|_1$$

calculated as the averaged L1 Norms of weights in each weight map.

Next, we calculate the average distance from Models 1 to 6, separately from the Baseline model. To compare two networks of different complexities in width, we adopt the structured contiguous pruning concepts from HeteroFL [6] and NeFL [16], utilizing common parts of different layers. The pruned layer l_k of Model k is represented using indexing: $M_k^{(l_k)}[: N_{out}^{M^{(l)}}, : N_{In}^{M^{(l)}}]$.

Average distance is calculated by:

$$\text{Average Distance} = \frac{1}{N_{out}^{M^{(l)}} \times N_{In}^{M^{(l)}}} \sum_i \|M_i^{(l)} - M_k^{(l_k)}[: N_{out}^{M^{(l)}}, : N_{In}^{M^{(l)}}]\|_1$$

The average magnitudes of Baseline models and distances from the Baseline model to each Model k are presented in Table 10. Comparing distances from

Baseline models to each Model k, we observe variations according to network complexities, influenced by varying depth and width. Specifically, in Pre-ResNet, the distances are $0.98 - 1.36$ times greater than the Baseline models' average magnitude of weights. For MobileNetV2, these distances range from $1.20 - 1.68$ times the Baseline's average magnitude, and for EfficientNetV2, they are $1.35 - 1.70$ times greater. These findings empirically show the scale variations linked to network complexities and highlight the necessity for a scalable aggregation method in our FedFA framework.

Table 10. According to network complexities, each network has a distinct scale in its weights, leading to scale variations across heterogeneous network architectures.

Model	Network Architecture	First Layer	Last Layer
Pre-ResNet			
		Average Magnitude	
Baseline	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 64, w_2 = 128, w_3 = 256, w_4 = 512)$	1.2251	0.0130
		Average Distance	
Model 1	$(d_1 = 3, d_2 = 3, d_3 = 3, d_4 = 3)$ $(w_1 = 64, w_2 = 128, w_3 = 256, w_4 = 512)$	1.3600	0.0172
Model 2	$(d_1 = 4, d_2 = 4, d_3 = 4, d_4 = 4)$ $(w_1 = 64, w_2 = 128, w_3 = 256, w_4 = 512)$	1.3742	0.0164
Model 3	$(d_1 = 5, d_2 = 5, d_3 = 5, d_4 = 5)$ $(w_1 = 64, w_2 = 128, w_3 = 256, w_4 = 512)$	1.3036	0.0158
Model 4	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 72, w_2 = 144, w_3 = 288, w_4 = 572)$	1.3744	0.0177
Model 5	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 80, w_2 = 160, w_3 = 320, w_4 = 640)$	1.3299	0.0168
Model 6	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 88, w_2 = 176, w_3 = 356, w_4 = 704)$	1.2027	0.0163
MobileNetV2			
		Average Magnitude	
Baseline	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 16, w_2 = 24, w_3 = 32, w_4 = 64, w_5 = 96, w_6 = 160, w_7 = 320)$	2.5798	0.0449
		Average Distance	
Model 1	$(d_1 = 3, d_2 = 3, d_3 = 3, d_4 = 3)$ $(w_1 = 16, w_2 = 24, w_3 = 32, w_4 = 64, w_5 = 96, w_6 = 160, w_7 = 320)$	4.3276	0.0625
Model 2	$(d_1 = 4, d_2 = 4, d_3 = 4, d_4 = 4)$ $(w_1 = 16, w_2 = 24, w_3 = 32, w_4 = 64, w_5 = 96, w_6 = 160, w_7 = 320)$	4.2563	0.0614
Model 3	$(d_1 = 5, d_2 = 5, d_3 = 5, d_4 = 5)$ $(w_1 = 16, w_2 = 24, w_3 = 32, w_4 = 64, w_5 = 96, w_6 = 160, w_7 = 320)$	3.9278	0.0598
Model 4	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 40, w_4 = 70, w_5 = 112, w_6 = 184, w_7 = 360)$	4.2019	0.0594
Model 5	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 40, w_4 = 80, w_5 = 120, w_6 = 200, w_7 = 400)$	3.7759	0.0542
Model 6	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 48, w_4 = 88, w_5 = 136, w_6 = 224, w_7 = 440)$	4.1361	0.0577
EfficientNetV2			
		Average Magnitude	
Baseline	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2, d_5 = 2)$ $(w_1 = 24, w_2 = 24, w_3 = 48, w_4 = 64, w_5 = 128, w_6 = 160, w_7 = 256)$	3.2296	0.0064
		Average Distance	
Model 1	$(d_1 = 3, d_2 = 3, d_3 = 3, d_4 = 3, d_5 = 3)$ $(w_1 = 24, w_2 = 24, w_3 = 48, w_4 = 64, w_5 = 128, w_6 = 160, w_7 = 256)$	4.5784	0.0093
Model 2	$(d_1 = 4, d_2 = 4, d_3 = 4, d_4 = 4, d_5 = 4)$ $(w_1 = 24, w_2 = 24, w_3 = 48, w_4 = 64, w_5 = 128, w_6 = 160, w_7 = 256)$	4.8878	0.0092
Model 3	$(d_1 = 5, d_2 = 5, d_3 = 5, d_4 = 5, d_5 = 5)$ $(w_1 = 24, w_2 = 24, w_3 = 48, w_4 = 64, w_5 = 128, w_6 = 160, w_7 = 256)$	5.4750	0.0093
Model 4	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2, d_5 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 56, w_4 = 72, w_5 = 144, w_6 = 184, w_7 = 288)$	4.5560	0.0091
Model 5	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2, d_5 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 64, w_4 = 80, w_5 = 160, w_6 = 200, w_7 = 320)$	5.0516	0.0091
Model 6	$(d_1 = 2, d_2 = 2, d_3 = 2, d_4 = 2, d_5 = 2)$ $(w_1 = 24, w_2 = 32, w_3 = 64, w_4 = 88, w_5 = 176, w_6 = 224, w_7 = 352)$	4.6417	0.0087

a) Pre-ResNet

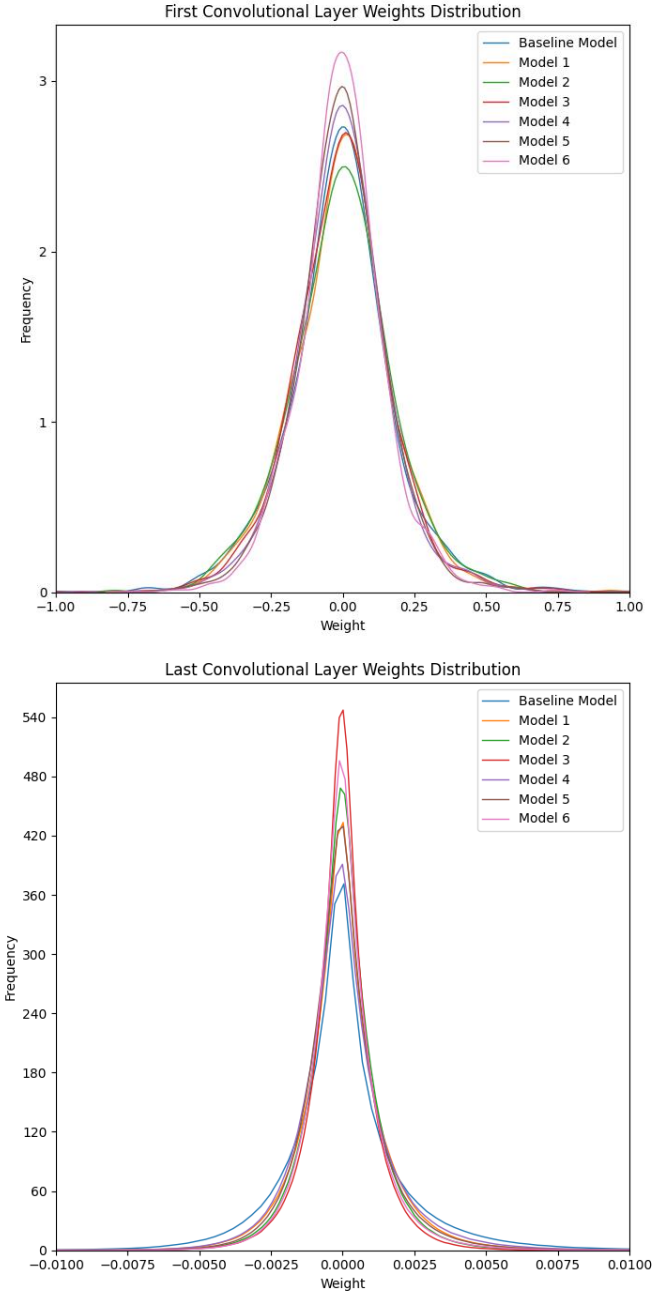

Fig. 7. Weights distributions across different architectures of the first (left) and the last layers (right) for Pre-ResNet.

b) MobileNetV2

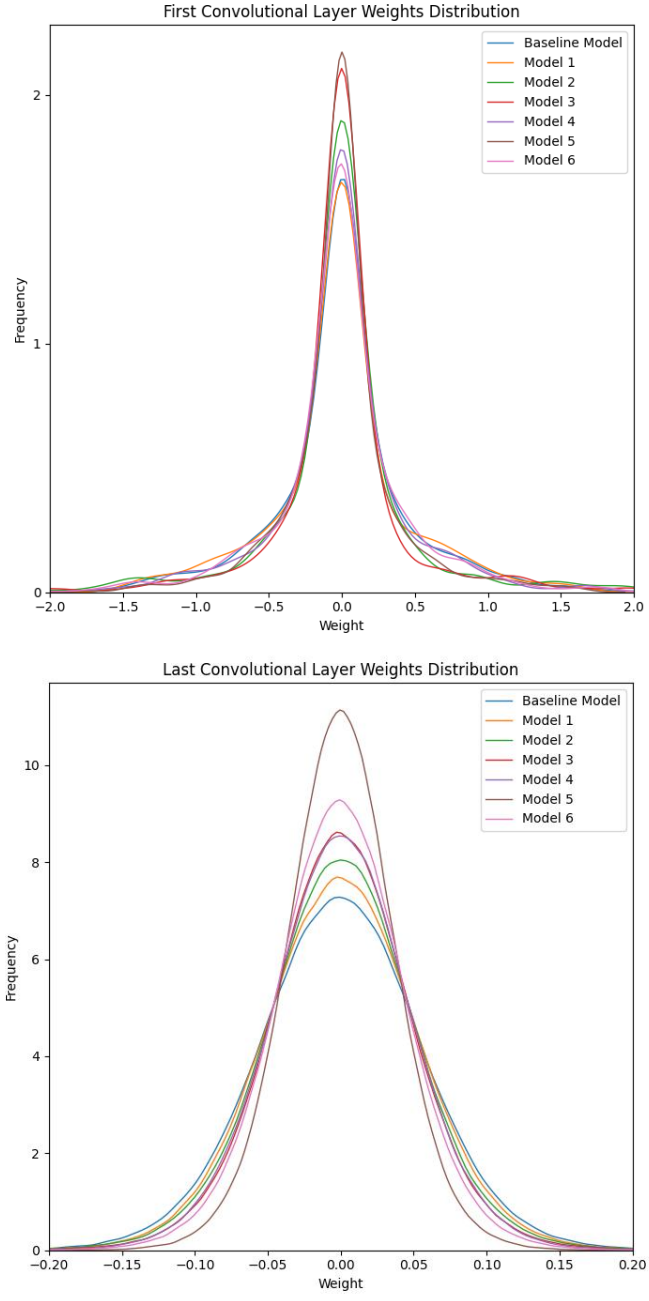

Fig. 8. Weights distributions across different architectures of the first (left) and the last layers (right) for MobileNetV2.

c) EfficientNetV2

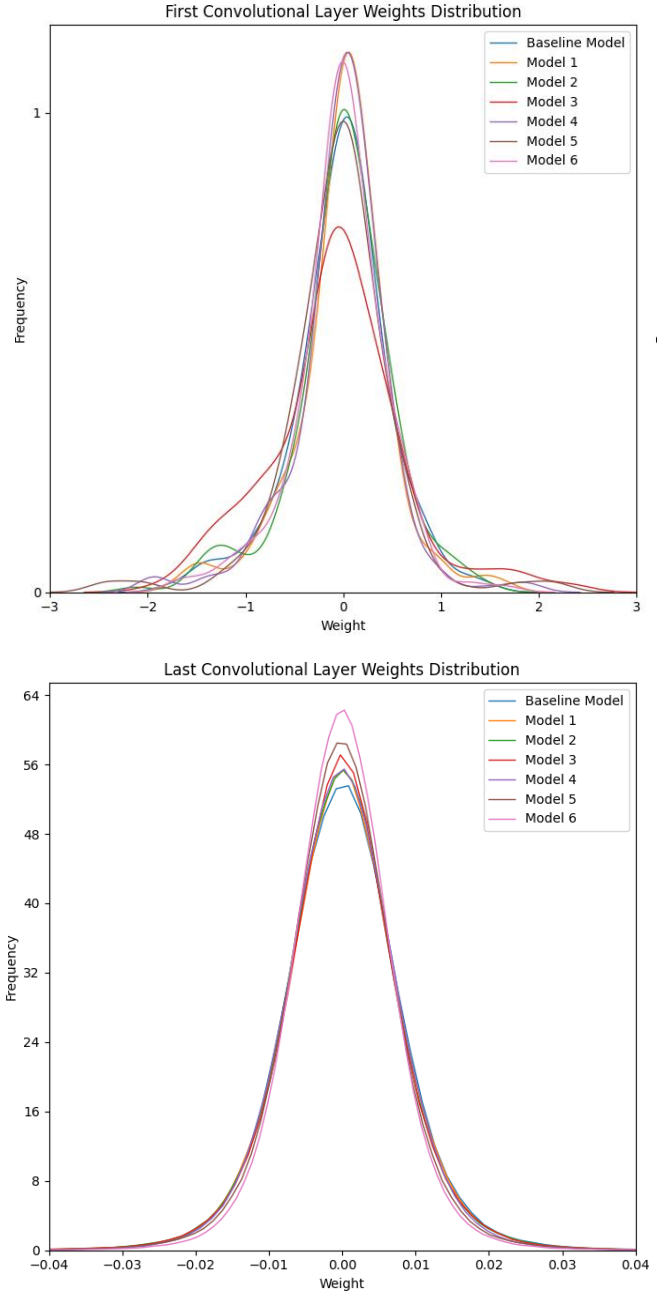

Fig. 9. Weights distributions across different architectures of the first (left) and the last layers (right) for EfficientNetV2.

G Analysis of Scaling factors in HeteroFL

G.1 Impact of Scaling Factors on Batch Normalization

In the HeteroFL framework [6], scaling factors are utilized as part of the network to compensate for scale variations due to heterogeneous architectures during the training phase. These factors are decided based on network complexities. However, understanding the effect of scaling factors on gradients during the Batch Normalization (BN) process is crucial. We consider two simple model outputs: $\hat{y_1} = \sum_i a_i x_i + b$ without scaling, and $\hat{y_2} = \alpha(\sum_i a_i x_i + b)$ with scaling. If we apply the BN process to $\hat{y_1}$:

$$BN(\hat{y_1}) = \frac{\hat{y_1} - \mu_1}{\sqrt{\sigma_1^2 + \epsilon_1}}$$

For $\hat{y_2}$, we adjust the mean (μ) and variance (σ^2) by the scaling factor α:

$$\mu_2 = \alpha\mu_1, \quad \sigma_2^2 = \alpha^2\sigma_1^2$$

$$BN(\hat{y_2}) = \frac{\hat{y_2} - \mu_2}{\sqrt{\sigma_2^2 + \epsilon_2}} = \frac{(\alpha\hat{y_1} - \alpha\mu_1)}{\sqrt{\alpha^2\sigma_1^2 + \epsilon_2}} \simeq \frac{\alpha(\hat{y_1} - \mu_1)}{\alpha\sqrt{\sigma_1^2 + \epsilon_2}}$$

The BN of the scaled output $\hat{y_2}$ simplifies to:

$$\simeq \frac{\hat{y_1} - \mu_1}{\sqrt{\sigma_1^2 + \epsilon_2}}$$

$$\Rightarrow BN(\hat{y_1}) \simeq BN(\hat{y_2})$$

For the quadratic loss function $L = \frac{1}{2}(y - BN(\hat{y}))^2$, the gradients with respect to coefficients a_i can be calculated for both outputs.
For $\hat{y_1}$ and its loss function L_1:

$$\frac{\partial L_1}{\partial a_i} = \frac{\partial L_1}{\partial BN(\hat{y_1})} \times \frac{\partial BN(\hat{y_1})}{\partial \hat{y_1}} \times \frac{\partial \hat{y_1}}{\partial a_i}$$

For $\hat{y_2}$ and its loss function L_2, using the equivalence of the BN outputs and considering the scaling effect in the derivative:

$$\frac{\partial L_2}{\partial a_i} = \frac{\partial L_2}{\partial BN(\hat{y_2})} \times \frac{\partial BN(\hat{y_2})}{\partial \hat{y_2}} \times \frac{\partial \hat{y_2}}{\partial a_i} \simeq \frac{\partial L_1}{\partial BN(\hat{y_1})} \times \frac{\partial BN(\hat{y_1})}{\alpha\partial \hat{y_1}} \times \frac{\alpha\partial \hat{y_1}}{\partial a_i}$$

$$= \frac{\partial L_1}{\partial BN(\hat{y_1})} \times \frac{\partial BN(\hat{y_1})}{\partial \hat{y_1}} \times \frac{\partial \hat{y_1}}{\partial a_i} = \frac{\partial L_1}{\partial a_i}$$

G.2 Summary: Negation of Scaling Factor Effect

This analysis demonstrates that the scaling factor α applied in $\hat{y_2}$ does not persist through the BN process. The equivalence in the BN outputs for both scaled and unscaled models leads to negating the α effect in the gradient computation. Consequently, applying scaling factors during the training may not adequately compensate for differences in model scales if networks have the BN layers, highlighting the need for another approach to managing scale variations.