A Framework for Double-Blind Federated Adaptation of Foundation Models

Supplementary Material

A. Proof of Proposition 1

Proof of Proposition 1. Let $\boldsymbol{P} = \boldsymbol{A}^{-1}\boldsymbol{B}, \boldsymbol{Q} = \boldsymbol{B}^{-1}\boldsymbol{C}$, and $\boldsymbol{R} = \boldsymbol{C}^{-1}\boldsymbol{A}$, where $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$ are known products derived from matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$. Each of $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{C}$ is a permutation matrix, defined by:

$$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{P}_n = \big\{\boldsymbol{X} \in \{0,1\}^{n\times n} : \boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}, \boldsymbol{X}\mathbf{1} = \mathbf{1}\big\}, \tag{9}$$

where $\mathbb{P}_n$ denotes the set of $n \times n$ permutation matrices, $\boldsymbol{X}^T$ is the transpose of $\boldsymbol{X}$, $\mathbf{1}$ is a vector of ones, and $\boldsymbol{X}^T = \boldsymbol{X}^{-1}$.

Non-uniqueness of solutions: To recover $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ uniquely, we would need the products $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$ to uniquely determine a single set of matrices $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$. However, for any valid solution $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ that satisfies $\boldsymbol{P} = \boldsymbol{A}^{-1}\boldsymbol{B}$, $\boldsymbol{Q} = \boldsymbol{B}^{-1}\boldsymbol{C}$, and $\boldsymbol{R} = \boldsymbol{C}^{-1}\boldsymbol{A}$, we can construct alternative solutions by applying a left multiplication with any permutation matrix $\boldsymbol{S} \in \mathbb{P}_n$. Define alternative matrices $\boldsymbol{A}' = \boldsymbol{S}\boldsymbol{A}$, $\boldsymbol{B}' = \boldsymbol{S}\boldsymbol{B}$, and $\boldsymbol{C}' = \boldsymbol{S}\boldsymbol{C}$, then:

$$\begin{aligned} \boldsymbol{A}'^{-1}\boldsymbol{B}' &= (\boldsymbol{S}\boldsymbol{A})^{-1}(\boldsymbol{S}\boldsymbol{B}) \\ &= \boldsymbol{A}^{-1}\boldsymbol{S}^{-1}\boldsymbol{S}\boldsymbol{B} \\ &= \boldsymbol{A}^{-1}\boldsymbol{B} = \boldsymbol{P}, \end{aligned} \tag{10}$$

and similarly, $\boldsymbol{B}'^{-1}\boldsymbol{C}' = \boldsymbol{Q}$ and $\boldsymbol{C}'^{-1}\boldsymbol{A}' = \boldsymbol{R}$. Thus, the products $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$ are also satisfied by the set $(\boldsymbol{A}', \boldsymbol{B}', \boldsymbol{C}')$. Since there are $n!$ possible choices for $\boldsymbol{S}$, there are $n!$ distinct sets of matrices $(\boldsymbol{A}', \boldsymbol{B}', \boldsymbol{C}')$ that satisfy the products $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$.

Brute-force as an optimal attack strategy: Given the non-uniqueness property above, an attacker aiming to recover $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ must resort to brute force, testing all possible configurations of $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ that satisfy the products $(\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R})$. The total number of combinations of $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ the attacker would need to test is $n!$ and for $n = 16$, we need $16! \approx 2 \cdot 10^{13}$ combinations.

□

B. Feature Similarity-based Attack

A potential attack strategy involves matching the $(\mathbf{b}_\ell, \mathbf{b}_{\ell+1})$ pairs, with $\ell \in [L-1]$, based on their similarity (e.g., L2 distance). After clients decrypt the received permuted representations $\mathbf{b}_\ell$, they could attempt to identify approximate nearest representations between the neighboring block outputs, thereby aiding in recovering individual permutation matrices Π_ℓ.

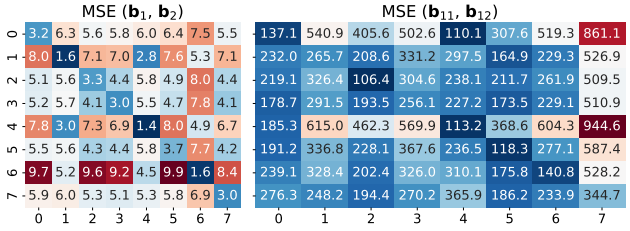

Figure 6. Heatmap visualization of the similarity (L2 distance) between corresponding block outputs or hidden representations: (i) between block outputs $\mathbf{b}_1$ and $\mathbf{b}_2$, and (ii) between block outputs $\mathbf{b}_{11}$ and $\mathbf{b}_{12}$. The X- and Y-axes represent the samples, with a batch size of 8. An ideal outcome for the attacker would be a perfect minimal diagonal, indicating strong alignment between the representations. While this is not fully observed in the plot, such outcome would reduce the number of brute-force trials required, as outlined in Proposition 1.

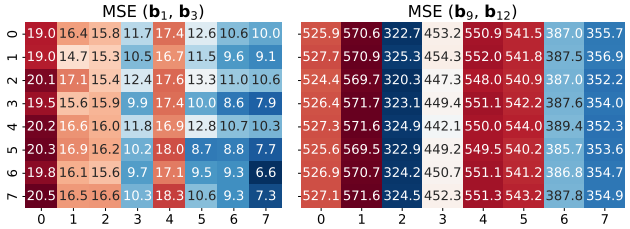

Figure 7. Heatmap visualization of the similarity (L2 distance) between non-consecutive block outputs or hidden representations: (i) between block outputs $\mathbf{b}_1$ and $\mathbf{b}_3$, and (ii) between block outputs $\mathbf{b}_9$ and $\mathbf{b}_{12}$. The X- and Y-axes represent the samples, with a batch size of 8.

To demonstrate this, we provide Figure 6, which illustrates that neighboring blocks exhibit similarity, making it feasible to infer permutation mappings. However, if clients receive randomly sampled blocks instead of sequential ones, the same attack strategy fails (see Figure 7). To mitigate this risk, we introduce stochastic sampling of hidden representations/attention blocks in every forward pass. This ensures that the permutation matrices are never fully exposed, preventing adversaries from reconstructing them (see Proposition 1).

Additionally, we illustrate the sampling probabilities of transformer blocks under different strategies in Figure 8. Specifically, we compare uniform sampling (included for illustrative purposes), our proposed stochastic sampling strat-

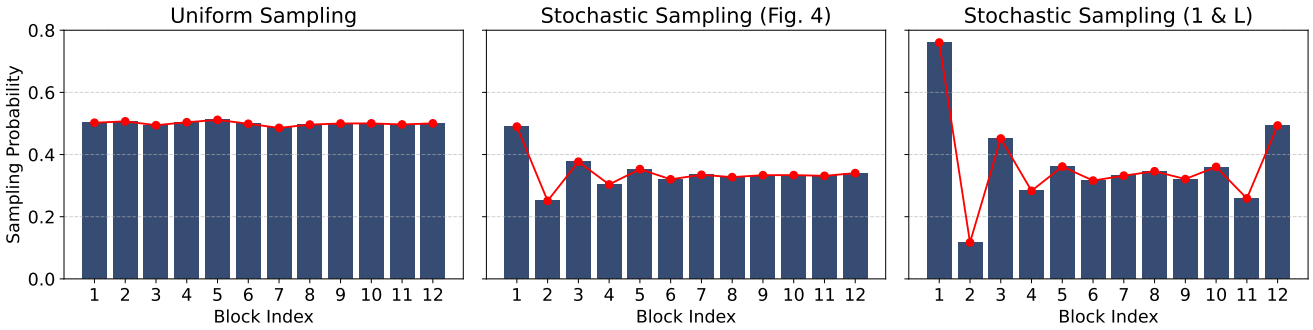

Figure 8. Visualization of the sampling probabilities for $L = 12$ transformer blocks under three different scenarios: (i) uniform random sampling with probability 0.5, (ii) our proposed stochastic sampling strategy, and (iii) stochastic sampling constrained to include either the first or last block.

egy depicted in Figure 5, and a constrained version of this stochastic strategy where either the first or last transformer block must always be sampled. The latter strategy ensures meaningful local updates by guaranteeing non-zero input entries for clients. Despite this constraint, the overall expected number of sampled blocks remains unchanged and aligns with the formulation in Eq. 14. Also, the learnable scaling parameter in each adapter layer (see Figure 3), denoted by α, addresses potential scaling issues that may arise due to zeroed-out block outputs/hidden representations during stochastic block sampling.

B.1. HE-Compatible Strategy

To maintain privacy while adhering to the computational constraints of homomorphic encryption (HE), we employ the following strategy:

- Non-selected blocks are set to zeros, ensuring they are not used during the local learning.
- Due to limited multiplicative depth in HE, we still transmit the non-selected blocks but introduce random noise before sending them to clients.
- Clients decrypt, re-encrypt, and return the updates. The server then subtracts the added noise before proceeding to the next block, ensuring that adversaries cannot infer the original representations.

This defense mechanism effectively counters feature similarity-based attacks while maintaining the efficiency of our privacy-preserving adaptation framework. A detailed explanation of how sampling is done is provided in Section 4.3.2.

B.2. Complexity Analysis of the Sampling Strategy

Consider a sequence consisting of L consecutive blocks, each assigned either a value of 1 (sampled) or 0 (not sampled), according to the probabilistic rules depicted in Figure 5:

- The first block is sampled (assigned the value 1) with probability 0.5 and not sampled (assigned 0) with probability 0.5.
- For each subsequent block $\ell \geq 2$:
  - If block $\ell - 1$ was sampled, then block ℓ is deterministically set to 0.
  - If block $\ell - 1$ was not sampled, then block ℓ is sampled/not sampled with probability 0.5.

Let p_ℓ denote the probability of block ℓ being sampled. Formally, this probability can be expressed recursively as:

$$p_\ell = 0.5 \times (1 - p_{\ell-1}), \quad \text{with } p_1 = 0.5. \tag{11}$$

Expected Number of Sampled Blocks. Solving this recurrence relation reveals an equilibrium behavior for large L. Specifically, as L gets larger, p_ℓ converges to a stationary probability p, satisfying:

$$p = 0.5(1 - p). \tag{12}$$

Solving this yields the stationary probability $p = \frac{1}{3}$. Therefore, each block has approximately a $\frac{1}{3}$ probability of being sampled. For L blocks, the expected number of sampled blocks S can be expressed as:

$$\mathbb{E}[S] = \sum_{\ell=1}^{L} p_\ell \approx \frac{L}{3}. \tag{13}$$

When the sampling is repeated for T times, the total expected number of sampled blocks is given by:

$$\mathbb{E}[S_T] \approx T \times \frac{L}{3}. \tag{14}$$

C. How Does the Distillation Work?

We adopt the transformer distillation approach [24], which consists of two key phases: (i) transformer-layer distillation and (ii) prediction-layer distillation. For generality,

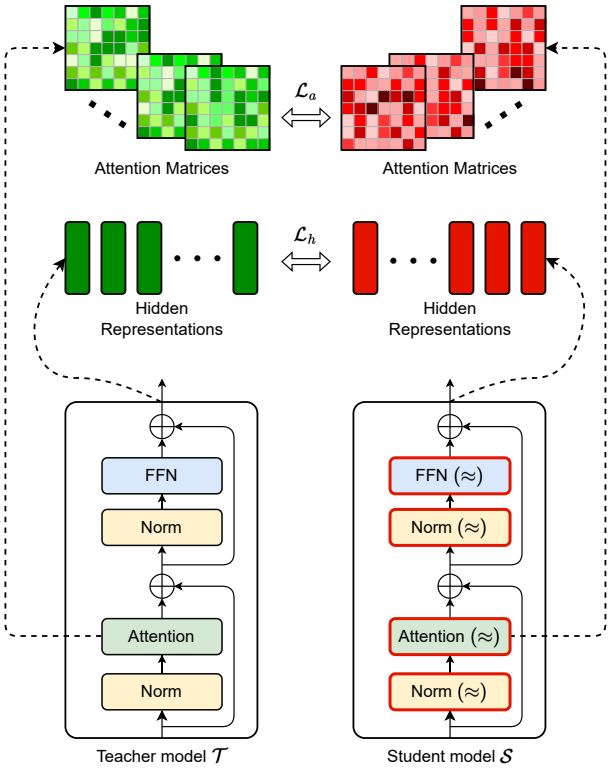

Figure 9. Schematic illustration of the layer-wise knowledge distillation (Equations 17 and 18).

let the original transformer model be referred to as the teacher model ($\mathcal{T}$) and the approximation-enabled transformer model as the student model ($\mathcal{S}$). We assume that both models share the same architecture but differ only in their non-linear components (Softmax, GELU, LayerNorm). Further, we outline the primary sub-layers of the transformer blocks:

- **Attention.** The attention function is formulated as follows:

$$\mathbf{A} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}, \tag{15}$$

  followed by a softmax operation. We are specifically interested in an un-normalized attention matrix $\mathbf{A}$.
- **Feed-forward network** is formulated as follows:

$$\mathbf{H}(x) = \text{GELU}(x\boldsymbol{W}_1 + b_1)\boldsymbol{W}_2 + b_2. \tag{16}$$

In the first stage of distillation, we perform an attention-based distillation (Eq. 15) and hidden representations based distillation (Eq. 16). More precisely,

$$\mathcal{L}_a = \frac{1}{h}\sum_{i=1}^{h} \left\| \mathbf{A}_i^{\mathcal{S}} - \mathbf{A}_i^{\mathcal{T}} \right\|^2, \tag{17}$$

where h is the number of attention heads. And the distillation of the hidden representation is formulated as follows:

$$\mathcal{L}_h = \|\mathbf{H}^{\mathcal{S}} - \mathbf{H}^{\mathcal{T}}\|^2, \tag{18}$$

where the matrices $\mathbf{H}^{\mathcal{S}}$ and $\mathbf{H}^{\mathcal{T}}$ are hidden representations of the respective models. For a detailed look, please refer to Figure 9, which illustrates how layer-wise distillation works. Then, the total loss of the first stage is simply defined as:

$$\mathcal{L} = \mathcal{L}_a + \mathcal{L}_h. \tag{19}$$

Further, we perform a prediction-layer distillation following the knowledge distillation approach of [19] by matching logits and using the following objective function:

$$\mathcal{L}_p = \mathcal{L}_{CE}(\boldsymbol{z}^{\mathcal{S}}/\tau, \boldsymbol{z}^{\mathcal{T}}/\tau), \tag{20}$$

where $\boldsymbol{z}^{\mathcal{S}}$ and $\boldsymbol{z}^{\mathcal{T}}$ are logit vectors produced by student and teacher models, respectively. $\mathcal{L}_{CE}$ is a cross-entropy loss and τ is a temperature parameter to produce softer probability distributions over classes. We set τ to 5 in all our experiments. Finally, the final loss objective is defined as:

$$\mathcal{L} = \begin{cases} \mathcal{L}_a + \mathcal{L}_h & \triangleright \text{ Stage I,} \\ \mathcal{L}_p & \triangleright \text{ Stage II.} \end{cases} \tag{21}$$

As outlined in the main paper, the total number of epochs is set to 30, with 15 epochs allocated to each stage.

D. Additional Experimental Results

D.1. Baseline Approaches

In this section, we describe the baseline methods used for comparison. **Full fine-tuning** involves updating all parameters of the pre-trained model by initializing them with pre-trained weights and subsequently training them with gradient updates. In federated experiments, this corresponds to the standard FedAvg algorithm [40]. **Linear probing** [4] fine-tunes only the classifier head while keeping the backbone frozen. In federated experiments, only the trainable parameters of the head are shared and averaged. **LoRA** (Low-Rank Adaptation) [21] is a popular parameter-efficient fine-tuning approach, freezing the original weights while training additional low-rank matrices. Its federated counterpart, the FedIT algorithm [65], applies FedAvg specifically to LoRA parameters. **Adapter tuning** [20] involves inserting lightweight adapter layers between self-attention (attention) and feed-forward modules, connected via residual connections. Each adapter layer comprises two fully connected layers with biases, separated by a nonlinearity (ReLU activation).

Datasets	Methods	Is double-blind?	No. of params (M)		Latency (ms)		Memory (GB)
CIFAR-10 / SVHN	Full fine-tuning	✗	82.1096	(11247.89 ×)	47.4	(3.14 ×)	18.13
	LoRA	✗	0.2886	(39.54×)	38.5	(2.55×)	15.73
	Adapter tuning	✗	3.3933	(464.68×)	37.9	(2.51×)	14.72
	Linear probing	✓	0.0073	(1.00 ×)	15.1	(1.00 ×)	9.39
	BlindFed	✓	0.2536	(34.74 ×)	15.7	(1.04 ×)	9.08
CIFAR-100	Full fine-tuning	✗	82.1756	(1121.09 ×)	47.4	(3.03 ×)	18.13
	LoRA	✗	0.3546	(4.84×)	38.4	(2.45×)	16.42
	Adapter tuning	✗	3.4593	(47.19×)	37.8	(2.41×)	15.41
	Linear probing	✓	0.0733	(1.00 ×)	15.7	(1.00 ×)	9.40
	BlindFed	✓	0.3196	(4.36 ×)	16.3	(1.04 ×)	9.08
Fed-ISIC2019	Full fine-tuning	✗	82.1082	(13916.64 ×)	51.2	(2.54 ×)	17.32
	LoRA	✗	0.2871	(48.66×)	38.5	(1.91×)	15.73
	Adapter tuning	✗	3.3919	(574.89×)	37.9	(1.88×)	14.72
	Linear probing	✓	0.0059	(1.00 ×)	20.2	(1.00 ×)	8.71
	BlindFed	✓	0.2522	(42.75 ×)	21.1	(1.05 ×)	8.39

Table 4. Comparison of the efficiency of our method with baseline approaches in terms of the number of parameters, latency and the memory requirement across four datasets (CIFAR-10/SVHN, CIFAR-100, and Fed-ISIC2019). Latency is measured per data point and includes both forward and backward passes. The last column (Memory (GB)) indicates the GPU memory required to conduct the experiment.

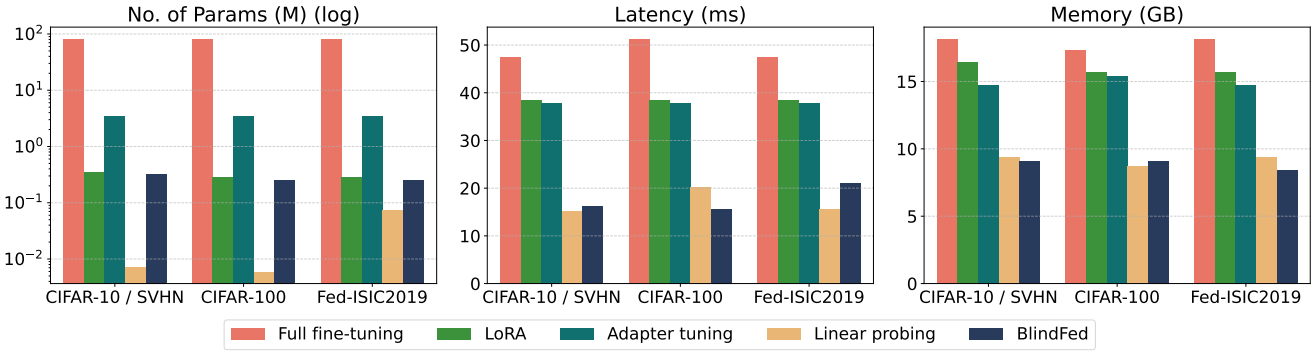

Figure 10. Comparison of the efficiency of different adaptation methods in terms of the number of parameters, latency, and memory usage across three datasets: CIFAR-10/SVHN, CIFAR-100, and Fed-ISIC2019. Each plot presents one of the three key metrics, with bars grouped by dataset and color-coded by method. Latency is measured per data point and includes both forward and backward passes. The memory column indicates the GPU memory required for training. Our proposed BlindFed method achieves a favorable balance between efficiency and performance. The results are derived from Table 4.

D.2. Main Results

Table 4 provides a comprehensive comparison of our method against several baseline approaches, including Full fine-tuning, LoRA, Adapter tuning, and Linear probing, across multiple datasets. This table extends the insights provided in Tables 1 and 2, focusing on three key aspects: the number of trainable parameters, latency, and GPU memory requirements. These results offer a deeper understanding of the trade-offs involved in adapting foundation models for federated learning while maintaining computational feasibility. As observed in the table, full fine-tuning consistently requires a significantly larger number of trainable parameters compared to all other methods, leading to substantially higher memory consumption and latency. In contrast, our method remains efficient, exhibiting a parameter count close to that of LoRA while requiring even less GPU memory.

Notably, across all datasets, our approach remains competitive with linear probing in terms of latency while requiring only a marginal increase in parameter storage. Specifically, in the CIFAR-10/SVHN setting, our method achieves near-optimal latency, operating at only a 4% increase over linear probing, despite introducing a significantly greater degree of adaptability. The reduction in trainable parameters compared to LoRA and Adapter tuning further highlights the effectiveness of our approach in maintaining

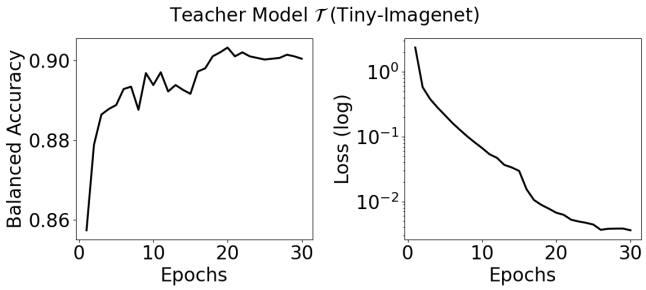

Figure 11. Performance plot of fine-tuning the teacher model on the public auxiliary dataset $\mathcal{D}_{aux}$ (Tiny-Imagenet). The left plot shows the balanced accuracy, while the right presents the loss performance (log scale).

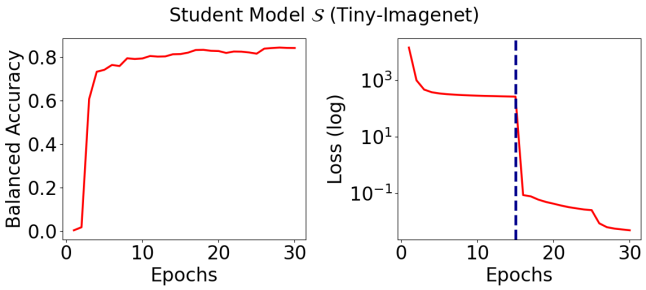

Figure 12. Performance plot of the distillation process on the Tiny-Imagenet dataset. The left plot depicts the balanced accuracy across both stages, while the right shows the loss performance, with the dashed line indicating the beginning of Stage 2 distillation (Eq. 21).

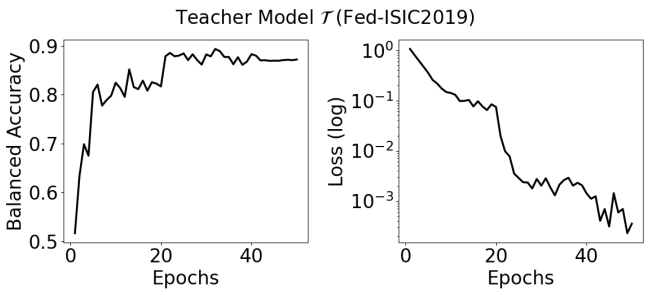

Figure 13. Performance plot of fine-tuning the teacher model on the Fed-ISIC2019 dataset with center=0. The plot on the left shows the balanced accuracy, while the plot on the right presents the loss performance (in a logarithmic scale).

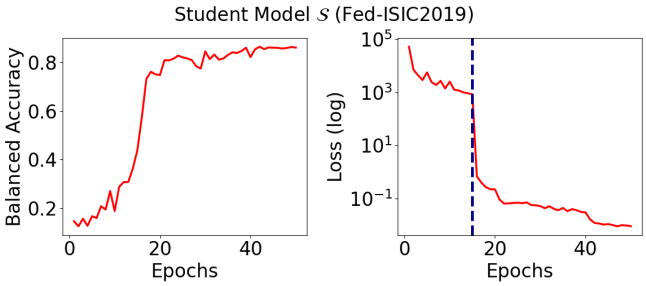

Figure 14. Performance plot of the distillation process on the Fed-ISIC2019 dataset with center=0. The plot on the left depicts the balanced accuracy across both stages, while the right plot shows the loss performance, with the dashed line indicating the beginning of Stage 2 distillation (Eq. 21).

model compactness without sacrificing computational efficiency.

From a resource perspective, the memory footprint of our method remains consistently low (due to the elimination of the need for backpropagation through the backbone), demonstrating an advantage over more conventional fine-tuning techniques. While full fine-tuning incurs nearly twice the memory cost, our method achieves comparable memory usage to linear probing while outperforming LoRA and Adapter tuning in terms of both efficiency and adaptability.

These findings underscore the scalability and practicality of our method in federated learning applications. By achieving an optimal balance between parameter efficiency, computational cost, and adaptability, our approach effectively mitigates the overhead typically associated with fine-tuning. Moreover, it demonstrates compatibility with homomorphic encryption techniques, achieving lower computational costs compared to conventional fine-tuning methods.

D.3. Distillation Results

Following the findings discussed in Section C, we present performance plots for two key processes: (i) fine-tuning the teacher model $\mathcal{T}$ on a public auxiliary dataset $\mathcal{D}_{aux}$, and (ii) distilling the student model $\mathcal{S}$ on the same dataset $\mathcal{D}_{aux}$ using the trained teacher model $\mathcal{T}$. Figure 11 illustrates the performance metrics – balanced accuracy and loss – of the fine-tuning process on the Tiny-ImageNet dataset. After completing this training, we proceed with distillation for the approximation-enabled student model. Figure 12 shows the performance plot for the distillation process on the Tiny-ImageNet dataset. In the loss behavior plot (on the right), a dashed line marks the start of the second stage. During the first stage, as described in Eq. 21, transformer-layer distillation is performed (showing a higher value due to the high-dimensional attention and hidden representation matrices). Then, the second-stage distillation follows, which is a prediction-layer distillation [19]. The learning rate scheduler is employed at epochs 15 and 25 with a decay factor of 0.1. Figures 13 and 14 present analogous performance plots for the Fed-ISIC2019 dataset, using center=0 as the public auxiliary dataset $\mathcal{D}_{aux}$. The learning rate scheduler

Degree d	Memory (GB)	Latency (s)	Attention loss ($\mathcal{L}_a$)	Representation loss ($\mathcal{L}_h$)	One-epoch accuracy (%)
2	16.82	0.41	44.37	915.46	75.17
4	19.68	0.48	38.31	748.62	83.85
6	22.54	0.52	33.92	655.26	84.36
8	25.39	0.58	30.91	603.53	84.47
10	28.99	0.62	28.17	573.54	84.16
16	36.83	0.81	20.31	527.95	85.08
∞ **(True)**	15.36	0.35	13.59	508.64	86.91

Table 5. **Softmax approximation results.** Latency is computed per a batch of samples, when batch size is set to 8. One-epoch accuracy refers to the accuracy we obtain when performing one full pass over the data for only one epoch. This experiment is carried on the Fed-ISIC2019(center=0) dataset.

is employed at epochs 20 and 40 with a decay factor of 0.1.

D.4. Approximation Results

In this section, we revisit and elaborate on the approximations introduced in the main paper.

Softmax. Given a vector $\boldsymbol{z} \in \mathbb{R}^n$ with components z_i, the softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

where e^{z_i} denotes the exponential function applied to the component z_i. The goal is to approximate the softmax function using a polynomial approximation of the exponential function, $P_n(\boldsymbol{z}_i)$, and to estimate the maximum deviation in softmax values due to this approximation.
The Taylor series provides a polynomial approximation of e^x around $x = 0$ up to d terms:

$$e^x \approx P_d(x) = \sum_{k=0}^{d} \frac{x^k}{k!}$$

The error bound of this approximation is given by the remainder term $R_n(x)$, expressed as:

$$R_d(x) = \frac{e^{\xi}}{(d+1)!} x^{d+1}$$

for some ξ between 0 and x, indicating the error in approximating e^x with $P_d(x)$.

Softmax approximation results. Table 5 presents softmax approximations while keeping all other non-linear components fixed. Using Eq. 3, we vary the polynomial degree d for the approximation during a single epoch of knowledge distillation (Section C, Stage I) and compare the results to the true softmax. The table reports key performance metrics, including attention loss $\mathcal{L}_a$ (Eq. 17), representation loss $\mathcal{L}_h$ (Eq. 18), and accuracy. Additionally, it provides efficiency metrics such as GPU memory usage and latency for processing a batch of samples. Balancing the trade-offs between time, memory requirements, and performance drop, we chose $d = 6$ for all experiments.

Algorithm 1 Inverse algorithm

Input: $0 < x < 2$, degree $d \in \mathbb{N}$
Output: an approximation of $1/x$ (Eq. 5)

$a_0 \leftarrow 2 - x$
$b_0 \leftarrow 1 - x$
for $n \leftarrow 0$ to $d - 1$ **do**
    $b_{n+1} \leftarrow b_n^2$
    $a_{n+1} \leftarrow a_n \cdot (1 + b_{n+1})$
end for
return a_d

Inverse. Since homomorphic encryption supports only addition and multiplication operations, it cannot perform division directly. To make it work in the encryption domain, we approximate the inverse function. The details of this approximation are presented in Eq. 5, and the corresponding algorithm is outlined in Algorithm 1. To ensure that the value of x falls within the range $(0, 2)$, we determine the maximum possible value of x in the plaintext domain and scale it accordingly in the encrypted domain using this maximum value.

Inverse approximation results. Table 6 summarizes the results of inverse function approximations, keeping all other non-linear components fixed and utilizing the softmax approximation with a 6th-degree polynomial. Following Algorithm 1 and Eq. 5, we varied the polynomial degree d under the same conditions described above. While the memory and time requirements remain nearly identical across different values of d, the focus is on performance metrics. Based on these considerations, we selected $d = 7$, as it closely approximates the true inverse in terms of losses and accuracy.

Degree d	Memory (GB)	Latency (ms)	Attention loss ($\mathcal{L}_a$)	Representation loss ($\mathcal{L}_h$)	One-epoch accuracy (%)
2	22.42	118.70	61.50	1698.19	57.87
4	22.44	119.64	57.42	1534.97	76.64
6	22.45	120.27	45.67	1352.91	81.77
7	22.46	120.43	44.11	1334.32	82.96
8	22.47	120.97	45.10	1352.55	83.23
10	22.48	121.70	43.23	1332.62	80.61
12	22.50	122.98	40.40	1302.41	82.55
16	22.53	124.76	38.86	1292.36	83.04
∞ **(True)**	22.46	10.57	38.77	1289.71	83.15

Table 6. **Approximation of the reciprocal (inverse function).** Latency is computed per a batch of samples (aggregate over L transformer blocks), when batch size is set to 8. One-epoch accuracy refers to the accuracy we obtain when performing one full pass over the data for only one epoch. This experiment is carried on the Fed-ISIC2019 (center=0) dataset.

Time taken to encrypt one sample	1062 ms
Time taken to decrypt one sample	168.7 ms
Time taken to encrypt a batch of 8 samples	4.65 s
Time taken to decrypt a batch of 8 samples	231.59 ms
Ciphertext size of one sample	17.33 MB
Plaintext size of one sample	6.21 MB
Encrypted inference time	136 s

Table 7. Computational and memory overhead for encrypted inference and encryption using FHE with the Tile Tensors framework. Results are presented per transformer block, evaluated at a 128-bit security level, including approximations for Softmax, inverse, and GELU operations.

D.5. Homomorphic Encryption Results

For the implementation of fully homomorphic encryption (FHE), we utilize the Tile Tensors framework [2, 8] on an NVIDIA A100-PCIE-40GB machine with 250 GB of available RAM. Table 7 summarizes the computational and memory requirements for our experiments, all evaluated at the standard 128-bit security level. The results presented in the table correspond to the encryption of one transformer block, accounting for approximations of operations such as Softmax, inverse, and GELU. Specifically, we report both the encrypted inference time and ciphertext memory size. The encrypted inference is executed on a powerful server, and the required time determines its computational complexity and latency, and the ciphertext size directly determines the communication overhead within the federated setting.

The encryption of a single data sample with a dimension $(577, 768)$ takes approximately 1062 ms, while decrypting the same sample takes significantly less time, around 168.7 ms, highlighting the asymmetry in encryption and decryption costs. Batch processing improves efficiency, as encrypting 8 samples takes 4.65 seconds, whereas decryption for the same sized batch completes in 231.59 ms. The ciphertext size of a single encrypted sample expands to 17.33 MB, larger than its plaintext counterpart (6.21 MB), indicating the additional communication overhead introduced by encryption. Moreover, encrypted inference for one transformer block requires approximately 136 seconds on the specified hardware, highlighting the computational overhead of inference under encryption. Nonetheless, this inference time could be substantially reduced with optimized cryptographic primitives, more powerful server infrastructure, and GPU acceleration [25, 27, 44, 60].

Computational Overhead. As with any cryptographic approach for privacy-preserving ML, BlindFed introduces non-negligible computational costs. However, it is designed to minimize the burden on thin clients by offloading most computations to a powerful server (Table 7). On the client side, only encryption/decryption of intermediate representations and a plaintext parallel adapter update are performed. On commodity CPUs, encrypting (decrypting) a single sample takes $\mathbf{1.06\,s}$ ($\mathbf{0.17\,s}$), and encrypting (decrypting) 8 samples in batch takes $\mathbf{4.7\,s}$ ($\mathbf{0.23\,s}$), requiring less than $\mathbf{1\,GB}$ of RAM. The server handles the expensive encrypted inference for each ViT block under CKKS, taking roughly $\mathbf{136\,s}$ per sample (amortized) and requiring more than $\mathbf{22\,GB}$ of memory. However, as mentioned above, these operations are embarrassingly parallel across clients, and GPU-accelerated CKKS implementations can further reduce the server-side inference time by over $10\times$.