This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

LaVFL: Efficient Verifiable Federated Learning for Large Language Models

Tianyou Zhang, Haiyang Yu*, *Member, IEEE*, Zhen Yang, *Member, IEEE*, Yuwen Chen, *Member, IEEE*, Shui Yu, *Fellow, IEEE*

***Abstract*—Federated Learning (FL) represents a distributed machine learning approach, enabling the joint training of a global model through the aggregation of gradients from participating clients without necessitating the exchange of raw data. Prior research has explored methods for verifying the correctness of aggregation in this context and mitigating the overhead associated with the verification process. Nonetheless, the advent of Large Language Models (LLMs), with their parameters numbering in the billions, presents ongoing challenges in devising efficient verification mechanisms in FL for large models. In this paper, we propose an innovative Efficient Verifiable Federated Learning scheme LaVFL, which focusing on addressing the verification challenges incurred by LLM. Specifically, we propose an efficient layer-by-layer verification approach for LLMs by designing a Convolution Gradient Compression (CGC) method without compromising model accuracy. Additionally, to minimize computational and communication overheads, we propose an efficient verification strategy PGS, namely, a Probabilistic Gradient Sampling strategy, which aims to reduce the gradient dimensions for each round of verification while ensuring a high probability of comprehensive verification. We implement a prototype of LaVFL, and extensive experimental results demonstrate that LaVFL achieves over a $300\times$ speedup in the aggregation verification phase and reduces communication overheads by more than 75%, compared to VeriFL under the same experimental setup.**

***Index Terms*—Efficient Verification, Federated Learning, Gradient Compression, Large Language Models.**

I. Introduction

Federated Learning (FL) [1], [2] has emerged as a promising distributed machine learning paradigm that enables collaborative training of a global model by aggregating gradients from participating clients without the need for raw data exchange. This approach offers significant advantages in terms of data privacy and security, as the raw data remains on the client devices, and only the model updates are shared with the central server [3]. Moreover, FL promotes inclusivity by accommodating heterogeneous data sources and device types. Since the training process occurs locally on each device, FL can handle variations in data distributions, device capabilities, and network conditions. This flexibility enables FL to be applied in diverse scenarios, ranging from healthcare and finance to smart cities and industrial IoT [4].

Despite the advantages of FL, ensuring the security and correctness of the aggregation process remains a critical challenge. Verifiable Federated Learning (VFL) [5] has emerged as a promising approach to address this issue by providing a means to verify the integrity of the aggregated model updates without compromising data privacy [6]. VFL introduces additional verification mechanisms that allow the clients to check the correctness of the central server's contributions, thereby detecting and preventing any malicious or erroneous updates from being incorporated into the global model [7]–[9].

However, the advent of LLMs [10], [11] poses two significant challenges for efficient verification in VFL. The first challenge is the inefficiency introduced by the multi-layer aggregation inherent in transformer architectures used in LLMs. Transformer models typically consist of dozens or even hundreds of layers, each requiring aggregation during the training process. This multi-layer aggregation significantly impacts the efficiency of the aggregation process, making it computationally intensive and time-consuming to verify updates at each layer. For example, in real-world applications such as natural language processing (NLP) tasks, where models like GPT-3 are used, verifying each layer's updates requires substantial computational resources. In a federated setting, where data is distributed across different parties (such as healthcare institutions or financial organizations), verifying updates at each layer would demand extensive coordination and computational effort, thus slowing down the overall process and hindering scalability.

The second challenge is related to the massive number of parameters associated with LLMs, often exceeding billions. Even when mechanisms are employed to reduce the number of aggregation layers, such as layer-wise aggregation or selective update schemes, the vast parameter space still poses a significant hurdle. For instance, in real-world deployment scenarios like personalized medicine or autonomous driving, where LLMs are trained on highly diverse and sensitive data across multiple parties, the sheer volume of parameters means that even aggregating a single layer can be computationally prohibitive. This not only impacts the efficiency of preprocessing steps but also the verification performance of VFL tasks. In practical applications, verifying the integrity and correctness of these aggregated updates would require extensive computational power and memory, which might not be feasible for all participating entities, especially in edge computing environments with limited resources [12].

Existing verification approaches for VFL can be categorized into solutions based on secure multiparty computation [13]–[17], homomorphic encryption [18]–[20], differential privacy [21], [22], and zero-knowledge proofs [23]–[25]. While these techniques offer strong security guarantees, they present significant challenges when applied to large-scale models like LLMs due to their high computational and communication

* Corresponding Author

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

overhead.

While secure multiparty computation (SMC) protocols guarantee privacy, their multiple communication rounds incur prohibitive latency when verifying billions of parameters. Homomorphic encryption permits computation on ciphertexts but suffers from extreme computational overhead as model size grows. Differential privacy protects individual data by adding noise, yet the extra noise-injection steps and parameter distortion can degrade LLM performance where precision is paramount. Zero-knowledge proofs (ZKPs) offer strong integrity guarantees without revealing secrets, but generating proofs at LLM scale explodes both proof complexity and communication cost.

In this paper, we aim to address the challenges of efficient and secure verification of FL aggregation in the context of LLMs by proposing the following question: *Is it possible to design an efficient verifiable federated learning scheme that achieves great performance in the verification of LLMs in FL settings?* To the best of our knowledge, the proposed LaVFL is the first Efficient Verifiable Federated Learning scheme that specifically addresses verification challenges incurred by LLMs. Our scheme adopts a layer-by-layer verification strategy tailored to the distinctive architectures of LLMs like BERT [26]. Inspired by techniques used in image processing [27], this paper introduces a novel Convolution Gradient Compression (CGC) method for gradient verification. Typically, convolution operations process image data to highlight local features and reduce spatial dimensions. Analogously, we apply convolution to the two-dimensional gradient matrix, treating it as an "image" to achieve effective gradient compression. Moreover, LaVFL integrates a novel random layer parameter verification mechanism that enhances the integrity and security of the model updates. Complementing this, our scheme employs a Probabilistic Gradient Sampling (PGS) strategy, optimizing the number of gradient dimensions extracted for verification based on the iteration count, thereby reducing overhead.

The main contributions of our paper are as follows:

- We propose the first Efficient Verifiable Federated Learning scheme (LaVFL) for LLMs. LaVFL combines layer-by-layer verification with CGC to efficiently validate aggregated updates. By reducing gradient dimensionality while preserving model accuracy, CGC enables efficient verification of large parameter counts incurred by LLMs. Additionally, to ensure update integrity and security, we introduce a random layer parameter verification mechanism in LaVFL. This mechanism ensures reliable global parameter acquisition for unvalidated layers, enhancing federated learning robustness.
- We propose an efficient verification strategy called PGS to minimize the computational and communication overheads associated with verification. PGS aims to reduce the gradient dimensions for each round of verification by selectively sampling a subset of the gradients based on their magnitude. The overarching aim is to ensure a high success rate throughout the entirety of the verification process. This approach drastically cuts down on unnecessary data transmission and computation, thereby enhancing efficiency.
- Our rigorous security analysis proves that our scheme guarantees the correctness of aggregation results, resistance to malicious collusion, and verifiability. A large number of experimental results show that LaVFL achieves an acceleration of more than $300\times$ in aggregation verification while reducing communication overhead by more than 75% compared to VeriFL [14] under the same experimental setup.

The rest of the paper is organized as follows: we briefly outline related work in Section II. After that, we present a structure overview of the LaVFL and elaborate on its procedures in Section III. The correctness and security analysis for LaVFL are provided in Section IV. Performance evaluation of the proposed scheme is presented in Section V. Finally, we conclude our paper in Section VI.

II. RELATED WORK

A. Privacy-Preserving FL.

The evolution of privacy-preserving Federated Learning (FL) [28] has catalyzed the development of innovative schemes designed to address both data privacy and the integrity of the FL process. Notably, the VOSA [29] introduces a secure aggregation approach combining verifiability with obliviousness, ensuring that individual updates are both indiscernible to and verifiable by the aggregator. While this approach ensures strong privacy guarantees, the cryptographic operations required impose a substantial computational burden. This can limit its applicability in managing large-scale models or real-time applications, which is a concern that LaVFL addresses by achieving significant speedups in verification while maintaining robust privacy guarantees.

In contrast, PVD-FL [30] emphasizes the decentralization aspect of FL, promoting a model that eliminates the need for a centralized aggregator. This shift enhances privacy by dispersing the aggregation focus but complicates the consensus process on the aggregated model among decentralized nodes without a central authority. Furthermore, VCD-FL [31] addresses the threat of participant collusion in FL. It incorporates dynamic participant selection and cryptographic verification to mitigate collusion risks, but these additions increase the complexity and may introduce latency, affecting operational efficiency. In comparison, LaVFL focuses on improving verification efficiency without significantly increasing computational overhead, thus mitigating latency issues that arise in schemes like VCD-FL. VeriFL [14] offers a communication-efficient and swiftly verifiable aggregation scheme for FL, ensuring safe and robust model updates while reducing communication overhead. Its performance is considered state-of-the-art, yet it lacks the capability to implement sampling strategies, limiting its effectiveness in scenarios involving LLMs with numerous parameters. LaVFL extends upon VeriFL's efficiency by addressing the specific challenges posed by large-scale models, such as LLMs, by incorporating efficient aggregation and verification mechanisms that can handle the complexity

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

of such models while maintaining low communication and computational overhead.

These schemes illustrate the diverse strategies employed to balance privacy preservation, operational integrity, and practical demands of Federated Learning, highlighting the field's continuous innovation and the challenges it faces.

B. Adaptive Pre-training in FL Environments.

The burgeoning interest in augmenting the efficacy of language models, particularly those predicated on the BERT [32] architecture, within the ambit of Federated Learning (FL), has catalyzed the development of innovative paradigms such as FEDBFPT [33]. These paradigms endeavor to surmount the distinct challenges posed by the application of FL to NLP, with an emphasis on efficiency, scalability, and the enhancement of model performance. FEDBFPT, among analogous endeavors, seeks to exploit the distributed essence of FL to refine language models through additional pre-training [34] on decentralized data repositories, thereby not infringing upon user privacy. Notwithstanding, the incorporation of differential privacy within FEDBFPT to bolster user privacy engenders a persistent conundrum-the equilibrium between privacy safeguarding and model precision. The imposition of noise to guarantee differential privacy may attenuate model efficacy, especially for nuanced NLP tasks [35] that demand high accuracy and comprehension. Balancing these aspects while maintaining strong privacy guarantees is a delicate and challenging endeavor.

C. Federated Large Language Models

The integration of LLMs into FL frameworks has driven significant advancements and posed unique challenges. Researchers have proposed various schemes to adapt LLMs to the federated setting, addressing issues related to communication, computation, and efficiency. [36] explores the integration of LLMs in FL, emphasizing the necessity for efficient communication protocols and aggregation techniques to manage the large parameter sizes inherent in these models. The study highlights the potential of model compression and quantization techniques to mitigate the communication overhead. However, despite these techniques, the communication overhead can still be substantial due to the large size of LLMs, leading to significant delays and increased costs in federated learning scenarios. The multi-layer nature of transformer models, central to LLMs, introduces significant inefficiencies in the aggregation process. [37] investigates these inefficiencies, focusing on the computational burden posed by multi-layer aggregation in a federated setting. They propose layer-wise aggregation strategies to reduce this burden, demonstrating improvements in training time and resource utilization. Nonetheless, aggregating updates across numerous layers can still introduce latency and computational complexity, impacting the overall training performance. Another notable challenge is the massive number of parameters in LLMs. Techniques such as federated dropout and selective parameter updates have been proposed to address this issue. For instance, [38] introduces a method for selectively updating parameters based on their importance, thereby reducing the overall communication and computation costs. Despite these strategies, the vast number of parameters imposes a heavy computational burden on participating devices. This can be particularly challenging for organizations with limited computational resources, hindering their ability to participate effectively in federated learning.

III. CONSTRUCTION OF OUR LaVFL

TABLE I
GLOSSARY OF NOTATIONS

Notation	Description
$\mathcal{S}_G$	Global model on server
$\mathcal{N}_G$	Number of layers in global model
$\mathcal{L}_k$	Local model of client k
$\mathcal{N}_{\mathcal{L}_k}$	Number of layers in client k's model
$\mathbf{G}$	Gradient matrix for convolution and encryption
$\mathbf{G}'$	Zero-padded $\mathbf{G}$ for gradient compression
$\mathbf{R_t}$	Invertible matrix from TA for gradient protection
$\mathbf{U}_t^j$	Random vectors summing to $\mathbf{R_t}$
$\mathbf{K_t}$	Convolution kernel for gradient compression
$\mathbf{K_s}$	Convolution kernel for layer selection
$\mathbf{W}'_{l,i}$	Transformed gradient after convolution
$\mathbf{h}_{l,i}$	Homomorphic hash of transformed gradient
$\mathbf{C}_{l,j}$	Encrypted gradient from client j
$\mathbf{E}_l$	Aggregated encrypted gradient for layer l
$\mathcal{C}$	Set of clients in the system
d	Original gradient matrix dimension
M	Convolution kernel size
O_{out}	Output dimension after convolution
$\mathbf{C}_s^k$	Encryption matrix for client k

In this section, we clarify the architecture of our proposed scheme, as depicted in Figure 1, which aims to fulfill the designated design objectives. Specifically, our scheme comprises several crucial steps: frame initialization, training and updating, verification of aggregation results, verification of random layer parameters, and PGS. To enhance clarity, we provide a summary of the key notations and their respective descriptions in Table I.

A. Overview

LaVFL optimizes the training and validation of LLMs in a federated learning context. During each update cycle, the initial transformer layer ($l = 0$) of the LLM is trained and validated. Concurrently, the server randomly selects specific layers (e.g., $l = (2, 4)$) for validation and then uses their parameters for subsequent layers ($l = (1, 2)$) in the next training round, ensuring these layers are neither trained nor validated on the client side, thus reducing computational load. The process involves obtaining initialization parameters from a trusted authority (TA), training the designated layer, performing CGC on the gradient, exchanging hash values for integrity checks, uploading encrypted gradients, aggregating gradients and randomly selecting layer parameters, distributing

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

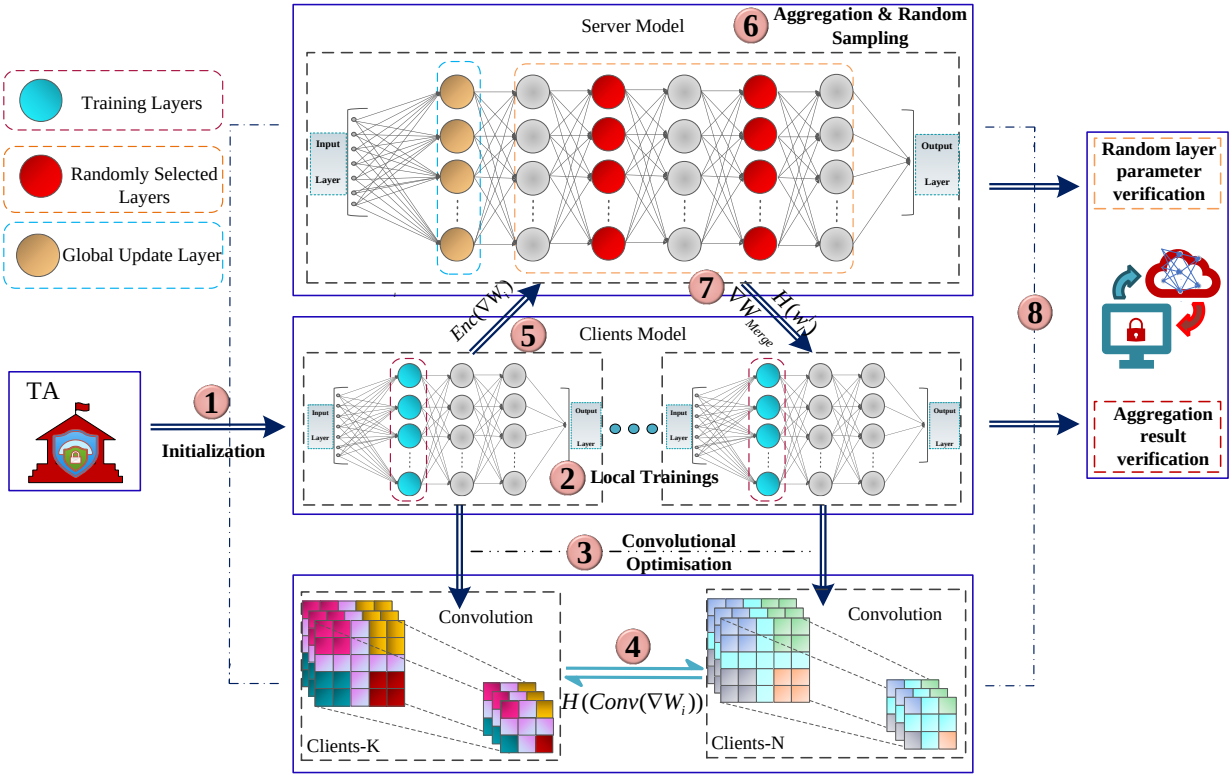


Fig. 1. Overview of our LaVFL. The transformer layer $l = 0$ of the LLM is being trained and validated during the current update. The server randomly selects layers $l = (2, 4)$ for validation and uses their parameters for layers $l = (1, 2)$ in the next round of training, where layers $l = (1, 2)$ are not trained and validated on the client side. Steps: (1) Obtain initialization parameters from TA and create local client LLM (BERT, etc.); (2) Train layer $l - th$; (3) Perform CGC on gradient; (4) Exchange hash values; (5) Upload encrypted gradient; (6) Aggregate gradients and randomly select layer parameters; (7) Distribute aggregated gradient and encrypted parameters; (8) Verify aggregated gradient and layer parameters.

the aggregated gradient and encrypted parameters, and finally verifying both aggregated gradient and layer parameters.

Inspired by image processing techniques [27], we introduce a novel CGC method for gradient verification. Typically used to highlight local features and reduce spatial dimensions in images, convolution operations are applied here to the two-dimensional gradient matrix, treating it as an "image" to achieve effective gradient compression. Importantly, this convolution operation is solely used to reduce verification overhead and is not applied to the aggregation of gradients, thereby ensuring that model accuracy remains unaffected.

Additionally, LaVFL incorporates a random layer parameter verification mechanism, enhancing the integrity and security of model updates by ensuring that randomly selected layers are accurately validated before their parameters are used in subsequent training rounds. This mechanism helps prevent model degradation caused by corrupted or malicious updates.

To further optimize the process, we employ a PGS strategy. This strategy dynamically adjusts the number of gradient dimensions extracted for verification based on the iteration count, balancing verification thoroughness and computational efficiency. As training progresses, PGS fine-tunes the sampling probability to minimize overhead while maintaining robust verification.

B. Frame Initialization

In our LaVFL implementation, we use BERT, renowned for its effectiveness in various natural language processing tasks. The LaVFL architecture is hierarchical: the server hosts a global model $\mathcal{S}_G$, encompassing the full BERT-base architecture with $\mathcal{N}_G$ layers, while each client k operates a localized version $\mathcal{L}_k$. These local models share embedding and output layers with $\mathcal{S}_G$ but have fewer transformer layers $\mathcal{N}_{\mathcal{L}_k}$, to suit devices with limited computational resources. We ensure that $\mathcal{N}_G > \mathcal{N}_{\mathcal{L}_k}$ across all clients, maintaining efficiency without sacrificing the ability to capture essential data features.

During initialization (as per Algorithm 1), the TA generates and distributes essential parameters within the federated network. We adapt the gradient matrix $\mathbf{G}$ for the scheme's convolution and encryption operations by zero-padding it to a square form $\mathbf{G}'$, if non-square. The expanded dimensions are subsequently eliminated during the parameter update phase. Given that the expansion is zero-padded, it incurs no additional computational cost.

To mitigate gradient leakage, the TA synthesizes an invertible matrix $\mathbf{R_t}$ and a sequence of random vectors $\{\mathbf{U}_t^1, \mathbf{U}_t^2, \ldots, \mathbf{U}_t^n\}$, summing to $\mathbf{R_t}$. Convolution kernels $\mathbf{K_t}$ and $\mathbf{K_s}$, fitting within the expanded dimensions, facilitate the model training and layer parameter selection phases, respectively. For added security, the TA generates invertible matrices $\{\mathbf{C}_s^1, \mathbf{C}_s^2$

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

Algorithm 1: Frame Initialization

Input: Gradient matrix $\mathbf{G} \in \mathbb{R}^{m \times n}$.
Output: $\mathbb{U}_t$, $\mathbf{K_t}, \mathbf{K_s}$, $\mathbf{R_t}$, $\mathbb{C}_s$, $\mathbb{C}_s^{-1}$.

Define $d = \max(m, n)$
Select a large number $\mathcal{N}$
Initialize the matrix sequence $\mathbb{C}_s$ to an empty list
Initialize the inverse matrix sequence $\mathbb{C}_s^{-1}$ to an empty list
Initialize a matrix sequence $\mathbb{U}_t$ to store $\mathbf{U}_t^i$ matrices
Initialize $S_U = 0 \leftarrow \sum \mathbf{U}_t^i$
Generate a random reversible matrix $\mathbf{R_t} \in \mathbb{R}^{d \times d}$
for $i = 1$ **to** $N - 1$ **do**
  Generate a random matrix $\mathbf{U}_t^i \in \mathbb{R}^{d \times d}$
  $S_U = S_U + \mathbf{U}_t^i$
  Add $\mathbf{U}_t^i$ to $\mathbb{U}_t$
end
Calculate $\mathbf{U}_t^N$ — $\sum_{i=1}^{N} \mathbf{U}_t^i \equiv \mathbf{R_t} \mod \mathcal{N} \leftarrow \mathbf{R_t} - S_U + (N-1) \times S_U \mod \mathcal{N}$
Add $\mathbf{U}_t^N$ to $\mathbb{U}_t$
for $j \in \{s, t\}$ **do**
  Generate a random convolution kernel $\mathbf{K}_j \in \mathbb{R}^{M \times M} (M <= d, \mathbf{K_s} \neq \mathbf{K_t})$
end
for $k = 1$ **to** N **do**
  Generate a random reversible matrix $\mathbf{C}_s^k \in \mathbb{R}^{d \times d}$
  Add $\mathbf{C}_s^k$ to $\mathbb{C}_s$
  Compute the inverse of $\mathbf{C}_s^k$, denoted as $\mathbf{C}_s^{-k}$
  Add $\mathbf{C}_s^{-k}$ to $\mathbb{C}_s^{-1}$
end
return $\mathbb{U}_t$, $\mathbf{K_t}, \mathbf{K_s}$, $\mathbf{R_t}$, $\mathbb{C}_s$, $\mathbb{C}_s^{-1}$

Algorithm 2: CGC and Random Layer Parameter

Input: Set of N clients $\mathcal{C}$, Gradient matrices $\mathbf{G'}_{l,i} \in \mathbb{R}^{d \times d}$ for client i at layer l, $\mathbb{U}_t$, K_t, K_r, R_t, $\mathbb{C}_s$, $\mathbb{C}_s^{-1}$, Θ, $\mathcal{N}_G$, $\mathcal{N}_{\mathcal{L}_k}$.
Output: Encrypted aggregated $\mathbf{E}_l$, $\mathbb{H}_i$, and $\mathbb{S}_k$.

for $i = 1$ *to* N **do**
  Convolve gradient matrix $\mathbf{G'}_{l,i}$ with kernel $\mathbf{K_t}$ for layer l :
  for $x = 0, y = 0$ *to* $d - M$ **do**
    $\mathbf{W}'_{l,i}[x, y] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} \mathbf{G'}_{l,i}[x + u, y + v] \cdot \mathbf{K_t}[u, v]$
  end
  Compute homomorphic hash: $\mathbf{h}_{l,i} \leftarrow \mathcal{H}(\mathbf{W}'_{l,i})$
  Broadcast hash $\mathbf{h}_{l,i}$ to other $N - 1$ clients
end
for $j = 1$ *to* N **do**
  Encrypt gradient: $\mathbf{C}_{l,j} \leftarrow \mathbf{W}'_{l,j} + \frac{\mathbf{R_t}}{N} - \mathbf{U}_t^j$
  Upload encrypted gradient $\mathbf{C}_{l,j}$ to server
end
Server aggregates encrypted gradients: $\mathbf{E}_l \leftarrow \sum_{i=1}^{N} \mathbf{C}_{l,i}$
for $l' = l + 1$ *to* $\mathcal{N}_G - \mathcal{N}_{\mathcal{L}_k} + l$ $(\{l_1, l_2, \ldots, l_{\mathcal{N}_{\mathcal{L}_k} - l}\} \leftarrow \{l + 1, \ldots, \mathcal{N}_G - \mathcal{N}_{\mathcal{L}_k} + l\}, l_1 < l_2 < \ldots < l_{\mathcal{N}_{\mathcal{L}_k} - l})$ **do**
  Convolve gradient matrix $\mathbf{G'}_{l'}$ with kernel $\mathbf{K_s}$ for layer l':
  for $x = 0, y = 0$ *to* $d - M$ **do**
    $\mathbf{W}'_{l'}[x, y] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} \mathbf{G'}_{l'}[x + u, y + v] \cdot \mathbf{K_s}[u, v]$
  end
  Compute homomorphic hash: $\mathbf{h}_{l'} \leftarrow \mathcal{H}(\mathbf{W}'_{l'})$
  for $k = 1$ *to* N **do**
    Encrypt each layer's gradient: $\mathbf{S}_{l',k} \leftarrow \mathbf{W}'_{l'} \cdot \mathbf{C}_s^k$
  end
end
Collect hash sequence and encrypted sequence:
$\mathbb{H}_i \leftarrow$ Collect $\mathbf{h}_{l'}$ of all $l'(i \in N)$
$\mathbb{S}_k \leftarrow$ Collect $\mathbf{S}_{l',k}$ of all $l'(k \in N)$
Broadcast $\mathbf{E}_l$, $\mathbb{H}_i$, and $\mathbb{S}_k$ to all clients
return $\mathbf{E}_l$, $\mathbb{H}_i$, *and* $\mathbb{S}_k$

$, \ldots, \mathbf{C}_s^n\}$, and computes their inverses $\{\mathbf{C}_s^{-1}, \mathbf{C}_s^{-2}, \ldots, \mathbf{C}_s^{-n}\}$.

C. Training and Updating

1) Model Training and Server Aggregation:

In a federated learning environment consisting of N clients $\mathcal{C}$, each client possesses gradient matrices $\mathbf{G'}_{l,i} \in \mathbb{R}^{d \times d}$ for a given layer l. We introduce a novel CGC method by applying a convolution operation to this matrix, which effectively achieves dimensionality reduction of the gradient. Summarized from Algorithm 2, the process begins with a convolution operation on the gradient matrix. Specifically, for an original gradient matrix $\mathbf{G'}_{l,i} \in \mathbb{R}^{d \times d}$ (where d represents the dimension of the gradient matrix), the convolution operation is defined as:

$$\mathbf{W}'_{l,i}[x, y] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} \mathbf{G'}_{l,i}[x + u, y + v] \cdot \mathbf{K_t}[u, v], \quad (1)$$

where (x, y) denotes the position in the transformed gradient matrix $\mathbf{W}'_{l,i}$, (u, v) represents the position within the convolution kernel $\mathbf{K_t}$, which has a size of $M \times M$, and $\mathbf{G'}_{l,i}[x + u, y + v]$ refers to the element from the original gradient matrix being multiplied by the corresponding kernel value $\mathbf{K_t}[u, v]$. The output matrix $\mathbf{W}'_{l,i}$ is the result of this convolution operation and will have a reduced size compared to the original gradient matrix. Specifically, the output dimension is given by:

$$O_{\text{out}} = d_{\text{orig}} - M_{\text{kernel}} + 1, \quad (2)$$

where d_{orig} is the original dimension of the gradient matrix, and M_{kernel} is the size of the convolution kernel. For an original matrix of size $d \times d$ and a convolution kernel of size $M \times M$, the new matrix $\mathbf{W}'_{l,i}$ will have dimensions:

$$O_{\text{out}} = d - M + 1,$$

for both width and height. Thus, the transformed matrix $\mathbf{W}'_{l,i}$ will have dimensions $(d - M + 1) \times (d - M + 1)$. After performing the convolution, each client computes a

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

homomorphic hash $\mathbf{h}_{l,i}$ of its transformed gradient matrix $\mathbf{W}'_{l,i}$. This hash serves as a means to verify the integrity of the data without exposing the actual gradient values. These homomorphic hashes $\mathbf{h}_{l,i}$ are then broadcasted to all $N-1$ other clients for verification. Next, each client encrypts its transformed gradient matrix $\mathbf{W}'_{l,i}$ using a matrix of random noise factors $\mathbf{R_t}$, which are tuned by a unique user factor $\mathbf{U}_t^j$. This encryption ensures the confidentiality of the gradient data. The encrypted gradient matrix $\mathbf{C}_{l,j}$ is then uploaded to the server by each client.

On the server side, the encrypted gradient matrices from all clients are aggregated to generate $\mathbf{E}_l$, representing the encrypted aggregated gradient information for layer l. This aggregation embodies a secure confluence of learning updates from all clients, enabling the update of the global model without disclosing the data of any individual client. Specifically, each client j encrypts its gradient matrix $\mathbf{W}'_{l,j}$ utilizing a shared key along with a factor of randomness denoted as $\frac{\mathbf{R_t}}{N}$, and concurrently subtracts a value $\mathbf{U}_t^j$ associated with it. Here, $\mathbf{R_t}$ denotes a random value common to all clients, while $\mathbf{U}_t^j$ is unique to each client. However, the aggregate of all $\mathbf{U}_t^j$ across clients equals $\mathbf{R_t}$. Consequently, upon the server-side aggregation of these encrypted gradient matrices, all instances of $\frac{\mathbf{R_t}}{N}$ collectively sum up to $\mathbf{R_t}$, and similarly, the sum of all $\mathbf{U}_t^j$ matches $\mathbf{R_t}$, thereby neutralizing each other in the aggregation process. The outcome of this operation, $\mathbf{E}_l$, effectively manifests as the sum of the original, unencrypted gradient matrices from all clients, albeit in an encrypted guise. This methodology ensures that, despite the server's inability to discern specific gradient values for each client during the aggregation phase, it can still obtain the cumulative sum of all gradients without compromising any client-specific data.

2) Random Layer Parameter: In the context of LLMs, we propose an additional step for handling the unvalidated layers within the client model.

For the unvalidated layers $l' \in \{l+1,\ldots,\mathcal{N}_G\} \setminus \{l+1,\ldots,\mathcal{N}_{\mathcal{L}_k}\}$ in the client model, excluding the current processing layer l, we randomly select $\mathcal{N}_{\mathcal{L}_k} - l$ layers.

For each chosen layer, an initial convolution operation is executed utilizing a convolution kernel $\mathbf{K_s}$ distinct from the $\mathbf{K_t}$ kernel applied to the current layer L. This selected kernel, $\mathbf{K_s}$, may vary in dimensions to facilitate the reduction of the original gradient's dimensionality. The underlying principle of the convolution operation remains consistent, where the kernel is maneuvered over the gradient matrix, generating a new matrix $\mathbf{W}'_{l'}$ through the aggregation of products between elements of the convolution kernel and the corresponding segments of the gradient matrix.

Upon completion of the convolution operation, a homomorphic hash of the newly formed matrix $\mathbf{W}'_{l'}$ is computed. Homomorphic hashing permits the execution of hash calculations on encrypted data without necessitating its decryption, thereby serving as a mechanism for verifying data integrity while preserving its confidentiality.

Following the hash computation, the subsequent phase involves encrypting the gradient matrix for each layer. Contrary to the earlier encryption phase, this step employs the matrix $\mathbf{C}_s^k$, associated with each client k, for encryption purposes. Such encryption ensures the server's lack of access to the original gradient information, with decryption exclusively possible by a client possessing the appropriate decryption key. This procedure culminates in the generation and server upload of a series of encrypted gradient matrices, subsequently aggregated. Ultimately, the server disseminates the collective encrypted gradient data, hash sequence, and encryption sequence to all clients. This selective processing method infuses randomness into the aggregation mechanism, significantly augmenting the algorithm's privacy-preserving attributes.

Algorithm 3: Aggregated Result Verification

Input: Set of N clients $\mathcal{C}$, $\mathbf{E}_l$, $\mathbf{K_t}$, Received hashes$\{\mathbf{h}_{l,1}, \mathbf{h}_{l,2}, ..., \mathbf{h}_{l,N-1}\}$ from other $N-1$ clients.

Output: Verification result.

foreach *client* $i \in \mathcal{C}$ **do**
  Perform convolution on $\mathbf{E}_l$ using kernel $\mathbf{K_t}$ to produce $\mathbf{E}'_l$:
  for $x = 0, y = 0$ *to* $d - M$ **do**
    $\mathbf{E}'_l[x,y] = \sum_{u=0}^{M-1}\sum_{v=0}^{M-1} \mathbf{E}_l[x+u, y+v] \cdot \mathbf{K_t}[u,v]$
  end
  Compute the homomorphic hash: $\mathbf{h}_l^* \leftarrow \mathcal{H}(\mathbf{E}'_l)$
  Initialize homomorphic product $\mathbf{H}_l = 1$
  for *each received hash* $\mathbf{h}_j$ **do**
    $\mathbf{H}_l \leftarrow \mathbf{H}_l \otimes \mathbf{h}_{l,j}$
  end
  $\mathbf{H}_l^* \leftarrow \mathbf{H}_l \otimes \mathbf{h}_{l,i}$
  Check if $\mathbf{H}_l^* \stackrel{?}{=} \mathbf{h}_l^*$. If not, $\mathcal{C}_i$'s verification fails.
end

D. Aggregation Result Verification

In this phase, as defined by Algorithm 3, each client $i \in \mathcal{C}$ receives a common gradient matrix $\mathbf{E}_l \in \mathbb{R}^{d\times d}$, encapsulating the aggregated gradient information for a specific model layer l. Subsequent to the convolution step, each client computes a homomorphic hash $\mathbf{h}_l^* \in \mathbb{G}$ on $\mathbf{E}'_l$ using a homomorphic hash function $\mathcal{H}$.

The final stage involves validating the consistency of the aggregation result by comparing the calculated $\mathbf{H}_l^*$ by each client against its respective $\mathbf{h}_l^*$. Uniformity of $\mathbf{H}_l^*$ across all clients with their $\mathbf{h}_l^*$ indicates successful convolution processing verification, while a discrepancy signifies a failure in verification.

E. Random Layer Parameter Verification

The random layer parameter verification phase, as described in Algorithm 4, is designed to verify the integrity and consistency of the parameters of randomly selected layers that were not validated during the model training phase. This process is illustrated in Figure 2. Let $\mathcal{L} = \{l_1, l_2, \ldots, l_t\}$ be the set of randomly selected layers for verification, where $t \le \mathcal{N}_{\mathcal{L}_k} - l$. For each layer $l' \in \mathcal{L}$, each client $i \in \mathcal{C}$ decrypts the received

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

Algorithm 4: Random Layer Parameter Verification

Input: $\mathbb{S}_k$, $\mathbb{C}_s^{-1}$, $\mathbf{R_s}$, $\mathbb{H}_i$.
Output: Verification result.

foreach *client* $i \in \mathcal{C}$ **do**
  foreach *l' in selected layers* **do**
    Decrypt received sequence: $\mathbf{S}'_{l',i} \leftarrow \mathbf{S}_{l',i} \cdot \mathbf{C}_s^{-i}$
    Convolve decrypted matrix with kernel:
    for $x = 0, y = 0$ *to* $d - M$ **do**
      $\mathcal{K}_l[x, y] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} \mathbf{S}'_{l',i}[x + u, y + v] \cdot \mathbf{K_s}[u, v]$
    end
    Compute homomorphic hash: $\mathbf{h}'^{*}_l \leftarrow \mathcal{H}(\mathcal{K}_l)$
    if $\mathbf{h}'^{*}_l \neq \mathbb{H}_i[l']$ **then**
      Verification fails for this layer;
      return *False*
    end
  end
  return *True*
end

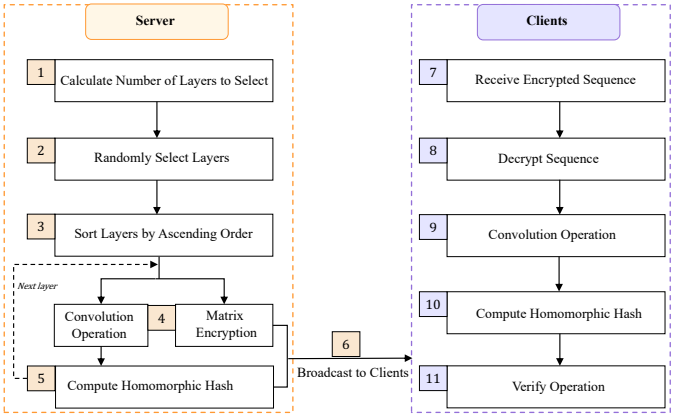


Fig. 2. Random layer parameter verification.

encrypted data sequence $\mathbf{S}_{l',i} \in \mathbb{R}^{d \times d}$ using the shared inverse encryption key $\mathbf{C}_s^{-i} \in \mathbb{R}^{d \times d}$.

After performing the necessary convolution and homomorphic hash operations, each client compares the computed hash $\mathbf{h}'^{*}_l \in \mathbb{G}$ with the corresponding hash value $\mathbb{H}_i[l'] \in \mathbb{G}$ from the received hash sequence $\mathbb{H}_i$. If the comparative analysis across all participating clients reveals uniform hash values, the verification is deemed successful. Otherwise, inconsistencies in hash values among any of the clients would indicate a verification failure, suggesting potential errors or tampering within the data processing workflow.

F. Probabilistic Gradient Sampling (PGS)

Although the CGC technique significantly reduces computational overhead, its effectiveness is somewhat limited when verifying LLMs with more than billion parameters. To further enhance the verification efficiency and complement the CGC approach while ensuring the correctness of the verification process, we introduce a PGS strategy. This strategy aims to substantially decrease the computational burden while retaining the reliability of the verification process. The PGS strategy is inspired by probabilistic sampling methods, which are commonly used in statistics and machine learning for efficiently approximating large datasets by selecting a representative subset of elements [39]. In our implementation, let $\mathbf{W}_{l,i}^{(s)} \in \mathbb{R}^{d \times d}$ denote the gradient matrix of client i at layer l in iteration round s, where d represents the gradient dimension. The PGS strategy selectively extracts a smaller-dimensional gradient matrix $\mathbf{G'}_{l,i} \in \mathbb{R}^{s \times s}$ from $\mathbf{W}_{l,i}^{(s)}$, where $s < d$. The selection of the smaller gradient matrix is performed randomly, with each element of $\mathbf{W}_{l,i}^{(s)}$ having an equal probability of being selected. This process ensures that the smaller matrix $\mathbf{G'}_{l,i}$ is an unbiased representative sample of the full gradient matrix.

The PGS strategy samples s^2 elements from $\mathbf{W}_{l,i}^{(s)}$ uniformly at random to form the smaller gradient matrix $\mathbf{G'}_{l,i}$. This approach ensures that each element of the gradient matrix has an equal chance of being selected, thereby providing an unbiased representation of the overall gradient information. After selecting the smaller gradient matrix $\mathbf{G'}_{l,i}$, a convolution operation is applied using a kernel $\mathbf{K_t} \in \mathbb{R}^{q \times q}$, where $q \leq s$, to obtain the convolved matrix $\mathcal{G}'_{l,i}$:

$$\mathcal{G}'_{l,i} = \sum_{u=0}^{q-1} \sum_{v=0}^{q-1} \mathbf{G'}_{l,i}[x + u, y + v] \cdot \mathbf{K_t}[u, v]. \tag{3}$$

The PGS strategy adjusts the sampling probability based on the iteration count s. In each iteration, the probability of selecting elements from the gradient matrix $\mathbf{W}_{l,i}^{(s)}$ is dynamically updated as the iteration progresses. Specifically, the probability distribution is controlled by a function $p(s)$, which is defined as:

$$p(s) = \frac{1}{1 + \alpha \cdot s},$$

where α is a positive constant that determines the rate of adjustment, and s represents the current iteration index. As the number of iterations increases, $p(s)$ decreases monotonically, leading to a gradual reduction in the overall sampling rate. This adjustment mechanism improves the efficiency and consistency of the verification process by controlling the number of selected elements. The updated sampling probabilities are then applied to select a reduced gradient matrix $\mathbf{G'}_{l,i}$ at each iteration.

The convolved matrix $\mathcal{G}'_{l,i}$ is then used in the subsequent steps of the verification process, as described in the original scheme. During the random layer parameter phase, the selection of the gradient matrix follows the same procedure as described above. Let $\mathbf{W}_{l',i}^{(s)} \in \mathbb{R}^{d \times d}$ denote the gradient matrix of client i at layer l' in iteration round s, where l' is a randomly selected layer. The PGS strategy is applied to $\mathbf{W}_{l',i}^{(s)}$ to obtain the smaller gradient matrix $\mathbf{G'}_{l',i} \in \mathbb{R}^{s \times s}$, which is then used in the subsequent steps of the random layer parameter verification process.

The PGS strategy significantly boosts the efficiency of the verification process by reducing the computational overhead associated with convolution operations and homomorphic hash computations. Moreover, by performing multiple rounds of random sampling and verification, the PGS strategy ensures a

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

high probability of successful system verification. The verification probability and efficiency are further discussed in Section V-D.

IV. SECURITY ANALYSIS

A. Assumptions and Threat Model

We assume that our main effort is to thwart adversaries from distorting aggregation results, thereby deviating from their authentic results. This assumption highlights our dedication to ensuring that the aggregation process meticulously reflects the collective contributions of all participants. Nonetheless, our examination does not encompass scenarios in which adversaries refine their inputs by leveraging historical interactions with the protocol, such as model substitution attacks [40]. This constraint acknowledges the inherent challenge of differentiating between maliciously engineered inputs and those legitimately submitted by compromised entities without prior discernment. Furthermore, we assume uninterrupted involvement of all clients throughout the model's training and verification phases, precluding instances of client dropout or sporadic availability.

We consider the same type of adversary as described in [41]. This malicious adversary has the added capability of instructing a corrupted server to arbitrarily forge the aggregation result, utilizing the knowledge gained from previously observed transcripts. Specifically, within the LaVFL paradigm, Trusted Authorities are posited as trustworthy and precluded from colluding with any entities. In contrast, the remainder of the participants are categorized as honest but curious, meaning they follow the prescribed protocols correctly but have an inherent intent to glean insights into the private data of other participants. The cloud server is considered malicious, indicating it may attempt to manipulate the aggregation results or other aspects of the protocol to its advantage. In particular, our model allows the cloud server to engage in collusion with multiple clients, thereby augmenting its capacity for potential malfeasance. Additionally, cloud servers are endowed with the ability to fabricate proofs (albeit collusion for the purpose of proof forgery is expressly prohibited) and to alter computational outputs with the aim of misleading clients.

B. Aggregation Correctness

Theorem 1. *In the LaVFL, assuming each entity executes the protocol honestly, participants can obtain accurate aggregated gradients to update the model*

Proof. Each client adds two masks to its convolved gradient: a share of a global random matrix $\mathbf{R_t}/N$ and a private matrix $\mathbf{U}_t^i$. The global shares sum to $\mathbf{R_t}$; the private masks are set up so that their sum is *also* $\mathbf{R_t}$. When the server adds all ciphertexts, the two $\mathbf{R_t}$ terms cancel, leaving only the sum of the clients' true convolved gradients. Thus, provided no party deviates from the steps, the server's output is exactly the gradient aggregate everyone needs.

Consider the steps in Algorithm 2, where each client i convolves its gradient matrix $\mathbf{G'}_{l,i}$ with the convolution kernel $\mathbf{K_t}$, producing the convolved gradients $\mathbf{W}'_{l,i}$. Each client encrypts its gradient as $\mathbf{C}_{l,i}$ and uploads it to the server, which then aggregates these into $\mathbf{E}_l$. According to the setup of the algorithm, we have:

$$\begin{aligned}\mathbf{E}_l &= \sum_{i=1}^{N} \mathbf{C}_{l,i} \\ &= \sum_{i=1}^{N} (\mathbf{W}'_{l,i} + \frac{\mathbf{R_t}}{N} - \mathbf{U}_t^i) \\ &= \sum_{i=1}^{N} \mathbf{W}'_{l,i} + \mathbf{R_t} - \sum_{i=1}^{N} \mathbf{U}_t^i.\end{aligned} \tag{4}$$

Since Algorithm 1 ensures that $\sum_{i=1}^{N} \mathbf{U}_t^i = \mathbf{R_t}$, it follows that:

$$\mathbf{E}_l = \sum_{i=1}^{N} \mathbf{W}'_{l,i}. \tag{5}$$

This proves that the server's final aggregated gradient $\mathbf{E}_l$ correctly reflects the sum of all clients' gradients. □

C. Data Privacy

Theorem 2. *In the LaVFL, given a participant's gradient matrix $\mathbf{W}_l$ for layer l, the scheme ensures that $\mathbf{W}_l$ is not disclosed to the server or any unauthorized entity.*

Proof. Before diving into the formal details, note that each client first *hides* its gradient by (i) convolving it with a public kernel, which scatters the information, and (ii) masking the result with fresh one-time random matrices that are known only to the client (and the TA). Because all clients use independent randomness, these masks cancel out only *after* the server sums the ciphertexts, leaving the server with a single aggregated value. Crucially, the server never learns the individual masks or the kernel-inverse, so it cannot "unmix" the sum to recover any single gradient matrix.

In Algorithm 2, due to the properties of the encryption scheme, the random matrix $\mathbf{R_t}$ cancels out during aggregation, and the server only obtains the sum of the convolved gradients and the sum of the participant-specific matrices $\mathbf{U}_t^i$. The server cannot recover the individual gradient matrices $\mathbf{W}'_{l,i}$ from the aggregated result $\mathbf{E}_l$ because:

- The convolution operation with kernel $\mathbf{K_t}$ obfuscates the original gradient values, i.e., $\mathbf{G'}_{l,i} \neq \mathbf{W}'_{l,i}, \forall i \in \mathcal{C}$.
- The encryption scheme using the random matrix $\mathbf{R_t}$ and participant-specific matrices $\mathbf{U}_t^i$ ensures that the server cannot separate the individual gradients from the aggregated result, i.e., $\nexists i \in \mathcal{C} : \mathbf{W}'_{l,i} = f(\mathbf{E}_l)$, where f is any computational function.

Furthermore, the LaVFL employs a secure aggregation protocol Π_{SecAgg}, which ensures that the server only learns the aggregated result and not the individual gradient matrices, i.e.,

$$\Pi_{\text{SecAgg}}(\mathbf{C}_{l,1}, \mathbf{C}_{l,2}, \ldots, \mathbf{C}_{l,N}) \rightarrow \mathbf{E}_l. \tag{6}$$

The encryption keys and random matrices used in the encryption process are securely generated and distributed by

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

TA and are not accessible to the server or any unauthorized entity, i.e.,

$$\Pr[\mathcal{A}(\mathbf{R_t}, \mathbf{U}_t^i) = (\mathbf{R_t}', \mathbf{U}_t^{i'})] \leq \text{negl}(\lambda), \quad \forall i \in \mathcal{C}, \tag{7}$$

where $\mathcal{A}$ is any probabilistic polynomial-time adversary, λ is the security parameter, and $\text{negl}(\lambda)$ denotes a negligible function in λ. In the context of this paper, a negligible function $\text{negl}(\lambda)$ means that as the security parameter λ increases (i.e., as the system becomes more secure), the probability of a successful attack by any adversary becomes extremely small and decreases faster than any inverse polynomial function of λ. This implies that the adversary's chances of breaking the encryption become practically zero as λ grows.

Therefore, the LaVFL ensures that a participant's gradient matrix $\mathbf{W}_l$ for layer l remains confidential, preventing the server or any unauthorized entity from recovering individual gradients from the aggregated results, thus preserving participant privacy, i.e.,

$$\Pr[\mathcal{A}(\mathbf{E}_l) = \mathbf{W}'_{l,i}] \leq \text{negl}(\lambda), \quad \forall i \in \mathcal{C}. \tag{8}$$

□

Theorem 3. *In the LaVFL, if the aggregation server colludes with k ($k \leq N-2$) participants, the private gradients of other participants will not be leaked during the convolution and encryption stages.*

Proof. Every client "blends" its gradient twice before sending it out. First, a public convolution kernel shuffles the gradient's numeric pattern, so the raw values are no longer visible. Second, each client adds a one-time mask that is known only to itself (and the TA). When the server sums all ciphertexts, those masks cancel *only in aggregate*. Because the server (even if it teams up with up to k users) never learns the undisclosed masks of the honest users, it sees just a single, inseparable sum and cannot peel off any individual gradient.

In Algorithm 2, each participant i performs convolution on their gradient matrix $\mathbf{G'}_{l,i} \in \mathbb{R}^{d\times d}$ using the kernel $\mathbf{K_t} \in \mathbb{R}^{M\times M}$ to obtain $\mathbf{W}'_{l,i} \in \mathbb{R}^{d\times d}$. The convolution operation provides a first layer of obfuscation, as it mixes the gradient values and makes it difficult to recover the original gradients without knowing the kernel $\mathbf{K_t}$. Formally, the convolution operation can be represented by Equation (1). After convolution, each participant i encrypts their gradient using the mask $\mathbf{R_t}$ and the matrix $\mathbf{U}_t^i$ as follows:

$$\mathbf{C}_{l,i} \leftarrow \mathbf{W}'_{l,i} + \frac{\mathbf{R_t}}{N} - \mathbf{U}_t^i. \tag{9}$$

The mask $\mathbf{R_t}$ is divided by N to ensure that it cancels out when the encrypted gradients are aggregated. The matrix $\mathbf{U}_t^i$ is unique to each participant and is used to randomize the encryption, making it infeasible for the aggregation server or colluding participants to recover the original gradients. The encrypted gradients $\mathbf{C}_{l,i}$ are then uploaded to the aggregation server, which computes the aggregated encrypted gradient $\mathbf{E}_l$ as follows:

$$\mathbf{E}_l \leftarrow \sum_{i=1}^{N} \mathbf{C}_{l,i} = \sum_{i=1}^{N} \mathbf{W}'_{l,i} + \mathbf{R_t} - \sum_{i=1}^{N} \mathbf{U}_t^i. \tag{10}$$

Due to the properties of the encryption scheme, the masks $\mathbf{R_t}$ cancel out during aggregation, leaving only the sum of the convolved gradients and the sum of the $\mathbf{U}_t^i$ matrices. The aggregation server cannot recover the individual gradients from this aggregated result. For the unvalidated layers, each participant i computes the encrypted gradients $\mathbf{S}_{l',k}$ using the convolved gradient $\mathbf{W}'_{l'}$ and the matrix $\mathbf{C}_s^k$ as follows:

$$\mathbf{S}_{l',k} \leftarrow \mathbf{W}'_{l'} \cdot \mathbf{C}_s^k. \tag{11}$$

The matrix $\mathbf{C}_s^k$ is generated by the TA and is unknown to the aggregation server and the participants. This encryption ensures that the gradients of the unvalidated layers remain protected. In Algorithm 3, each participant verifies the integrity of the aggregated result using homomorphic hash functions $\mathcal{H}$. The homomorphic property of the hash functions allows the participants to check the correctness of the aggregation without revealing their individual gradients:

$$\mathcal{H}(\mathbf{E}'_l) = \mathcal{H}(\mathbf{W}'_{l,1}) \otimes \mathcal{H}(\mathbf{W}'_{l,2}) \otimes \ldots \otimes \mathcal{H}(\mathbf{W}'_{l,N}). \tag{12}$$

Therefore, even if the aggregation server colludes with up to k participants (where $k \leq N-2$), the collusion will not leak the gradients of the remaining honest participants. □

D. Verifiability

Theorem 4. *In the LaVFL, each participant can independently verify the correctness of the aggregated result after decryption, and the verification mechanism can detect forged or improperly aggregated results with an overwhelming probability.*

Proof. Think of the homomorphic hash as a "checksum" that behaves well with addition: $\text{hash}(\mathbf{A}+\mathbf{B}) = \text{hash}(\mathbf{A}) \otimes \text{hash}(\mathbf{B})$. Every client publishes the hash of its own (masked-and-convolved) gradient. When the server later returns the sum $\mathbf{E}_l$, each client can (i) hash that sum itself and (ii) multiply together all the individual hashes it previously collected. If the server aggregated honestly, the two values coincide; if the server forged, dropped, or altered even a single share, a mismatch appears. Because finding two different matrices that collide under the hash would require breaking a hard cryptographic assumption, any incorrect aggregation is caught with overwhelming probability.

After receiving the aggregated encrypted gradient $\mathbf{E}_l$ from the aggregation server, each participant i performs convolution on $\mathbf{E}_l$ using the kernel $\mathbf{K_t}$ to obtain $\mathbf{E_l}'$, as shown in Algorithm 3:

$$\mathbf{E}'_l[x,y] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} \mathbf{E}_l[x+u, y+v] \cdot \mathbf{K_t}[u,v]. \tag{13}$$

Each participant i then computes the homomorphic hash of the convolved aggregated gradient $\mathbf{E}'_l$:

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

$$\mathbf{h}_l^* \leftarrow \mathcal{H}(\mathbf{E}_l'), \tag{14}$$

where $\mathcal{H} : \mathbb{R}^{d\times d} \to \mathbb{G}$. Participant i also computes the homomorphic product of the received hashes $\mathbf{h}_{l,j}$ from other participants and their own local hash $\mathbf{h}_{l,i}$:

$$\mathbf{H}_l^* \leftarrow (\otimes_{j\neq i}\mathbf{h}_{l,j}) \otimes \mathbf{h}_{l,i}, \tag{15}$$

where $\otimes : \mathbb{G}\times\mathbb{G}\to\mathbb{G}$. Participant i verifies the correctness of the aggregated result by checking if $\mathbf{H}_l^*$ is equal to $\mathbf{h}_l^*$:

$$\mathbf{H}_l^* \stackrel{?}{=} \mathbf{h}_l^*. \tag{16}$$

If the equation holds, the verification succeeds; otherwise, it fails. The homomorphic hash function $\mathcal{H}$ has the property that for any two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d\times d}$:

$$\mathcal{H}(\mathbf{A}+\mathbf{B}) = \mathcal{H}(\mathbf{A}) \otimes \mathcal{H}(\mathbf{B}). \tag{17}$$

This property ensures that the homomorphic product of the hashes of the individual convolved gradients is equal to the hash of the sum of the convolved gradients:

$$\mathcal{H}(\mathbf{W}_{l,1}')\otimes\mathcal{H}(\mathbf{W}_{l,2}')\otimes\ldots\otimes\mathcal{H}(\mathbf{W}_{l,N}') = \mathcal{H}(\sum_{i=1}^{N}\mathbf{W}_{l,i}'). \tag{18}$$

If the aggregation server or any participant attempts to forge or improperly aggregate the results, the homomorphic product of the hashes $\mathbf{H}_l^*$ will not match the hash of the convolved aggregated gradient $\mathbf{h}_l^*$. The probability of a forged or improperly aggregated result passing the verification is negligible, as it would require finding a collision in the homomorphic hash function $\mathcal{H}$. Formally, for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\begin{aligned}\Pr[\mathcal{A}(\mathbf{E}_l) = (\mathbf{E}_l', \mathbf{h}_l) : \mathbf{H}_l = \mathbf{h}_l^* \wedge\ \mathbf{E}_l' \\ \neq \sum_{i=1}^{N}\mathbf{W}_{l,i}] \leq \text{negl}(\lambda).\end{aligned} \tag{19}$$

□

Theorem 5. *In the random layer parameter verification phase of the LaVFL, for each honest participant P_i, there exists a verification protocol $\mathcal{V}$ that enables P_i to independently verify the correctness of the aggregation result with overwhelming probability, even in the presence of malicious entities $\mathcal{M}$ that may tamper with the encrypted sequence.*

Proof. Verifying every layer after every round would be prohibitively expensive, so each client instead checks a *random sample* of layers. For every chosen layer the client (i) decrypts the ciphertext it just received, (ii) recomputes a homomorphic hash on that plaintext, and (iii) compares the result with the hash value all peers had previously published. Because the hash is collision-resistant and homomorphic, any single wrong coefficient introduced by a malicious server or colluding clients causes a mismatch except with negligible probability. Sampling layers at random keeps the overhead low while still ensuring that an incorrect aggregation is detected with overwhelming probability.

In Algorithm 4, each participant P_i receives the encrypted sequence $\mathbb{S}_k$, the inverse matrix sequence $\mathbb{C}_s^{-1}$, the random matrix $\mathbf{R_s}$, and the hash sequence $\mathbb{H}_i$. Let $\mathcal{L} = \{l_1, l_2, \ldots, l_t\}$ be the set of randomly selected layers for verification, where $t \leq \mathcal{N}_{\mathcal{L}_k - l}$. The verification protocol $\mathcal{V}$ is defined as the sequence of steps performed by each honest participant P_i for each randomly selected layer $l' \in \mathcal{L}$. Formally:

$$\mathcal{V} = \{(\mathbf{S}_{l',i}', \mathcal{K}_l, \mathbf{h}_l'^*, \mathbb{H}_i[l']) : l' \in \mathcal{L}\}. \tag{20}$$

Let $\mathcal{M}$ be the set of malicious entities that may tamper with the encrypted sequence $\mathbb{S}_k$ or the hash sequence $\mathbb{H}_i$. If any malicious entity $\mathcal{M}_j \in \mathcal{M}$ tampers with $\mathbb{S}_k$ or $\mathbb{H}_i$, the computed hash $\mathbf{h}_l'^*$ will not match the received hash $\mathbb{H}_i[l']$ with overwhelming probability, due to the collision resistance of the homomorphic hash function $\mathcal{H}$. Formally, for any probabilistic polynomial-time adversary $\mathcal{A}$:

$$\begin{aligned}\Pr[\mathcal{A}(\mathbb{S}_k, \mathbb{H}_i) = (\mathbb{S}_k', \mathbb{H}_i') : \exists l' \in \mathcal{L}, \mathbf{h}_l'^* \\ = \mathbb{H}_i'[l'] \wedge (\mathbb{S}_k' \neq \mathbb{S}_k \vee \mathbb{H}_i' \neq \mathbb{H}_i)] \\ \leq \text{negl}(\lambda).\end{aligned} \tag{21}$$

Since the probability of a malicious entity $\mathcal{M}_j \in \mathcal{M}$ successfully tampering with the encrypted sequence or the hash sequence without being detected is negligible, each honest participant P_i can independently verify the correctness of the aggregation result with overwhelming probability by following the verification protocol $\mathcal{V}$. The probability of a successful verification for an honest participant P_i is given by:

$$\begin{aligned}\Pr[\mathcal{V} \text{ succeeds for } P_i] = 1 - \Pr[\exists l' \in \mathcal{L}, \mathbf{h}_l'^* \\ \neq \mathbb{H}_i[l']] \geq 1 - \text{negl}(\lambda).\end{aligned} \tag{22}$$

In summary, the random layer parameter verification phase of the LaVFL provides a verification protocol $\mathcal{V}$ that allows each honest participant P_i to independently verify the correctness of the aggregation result with overwhelming probability, even in the presence of malicious entities $\mathcal{M}$. □

Discussion. Beyond the adversarial behaviors discussed above, additional threats in federated learning include *model poisoning* and *gradient inversion* attacks. Model poisoning involves crafting malicious updates that cause targeted misbehavior in the global model, often by mimicking benign gradients to evade detection. Defenses include robust aggregation (e.g., Krum, Trimmed Mean, Median) [42], [43] and anomaly detection based on gradient norms or model behavior. Gradient inversion aims to reconstruct private data from shared gradients, especially when gradients are exposed in plaintext [44]. Mitigations include differential privacy [45] and secure aggregation to conceal individual updates. While our current threat model does not explicitly cover these attacks, integrating such defenses into the LaVFL framework is essential for robust deployment in adversarial settings.

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

V. PERFORMANCE EVALUATION

A. Experimental Setup

In this study, we adopted the BERT-base-uncased model as our primary architecture. Training was conducted on a server equipped with an Intel(R) Xeon(R) Gold 6148 CPU at 2.40GHz and eight NVIDIA GeForce RTX 3090 GPUs, each with 24,576MB of video memory. Our software stack included Pytorch version 2.2.0 and CUDA version 12.4. To tailor the model for specific downstream tasks, initial pre-training was performed on the global model of each client based on this configuration.

For our Federated Learning (FL) setup, client devices featured a 13th Gen Intel(R) Core(TM) i7-13620H CPU at 2.40 GHz, 32.0 GB of RAM, and an NVIDIA GeForce RTX 4050 GPU. A local BERT-base-uncased model was deployed on each, with the pre-training dataset distributed evenly across these devices using the S2ORC dataset [46]. To address the challenges of high-dimensional gradient analysis, which is memory-intensive, comparative experiments were carried out on a laboratory server with 534.8 GB of RAM and an NVIDIA GeForce RTX 3090 GPU with 24,576 MB of video memory, ensuring sufficient capacity for rigorous testing.

For the PGS strategy, several hyperparameters were chosen to optimize both verification efficiency and accuracy. Instead of fixing the sample size, we set the verification probability to 99%, ensuring that the sampled gradient elements are sufficient to achieve a high-confidence verification. The sampling probability $p(s)$ was dynamically adjusted throughout the iterations, following the function $p(s) = \frac{1}{1+\alpha \cdot s}$, where α was set to 0.1. As the iteration count s increases, the probability of selecting important gradient elements also gradually increases to maintain verification reliability at $P_{\text{verify}} = 0.99$.

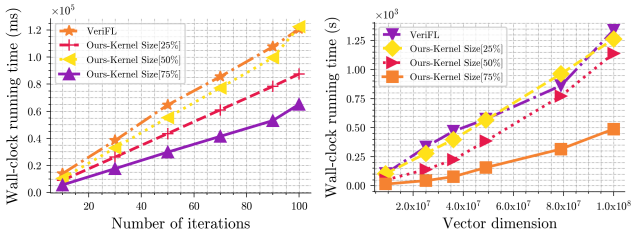
B. Computational Overhead

This section evaluates the computational overhead of our proposed scheme, focusing on the local training, verification process, and random layer parameter phase. We compare with VeriFL, currently the most efficient verification method, to demonstrate that our scheme can outperform.

To ensure a fair comparison, we integrate VeriFL's aggregation verification method with the corresponding component of our scheme, maintaining consistent parameters. This enables the construction of two comparative schemes that differ only in their aggregation verification approaches. Additionally, we provide specific parameter settings for VeriFL, including the use of a linearly homomorphic hash (LHH) over the NIST P-256 curve.

In addition, the computational overhead of our scheme is influenced by the dimensionality of the convolution kernel parameters. Therefore, we conduct a comparative analysis of the overhead associated with different parameter sizes (25%, 50%, and 75% relative to the local gradient dimension size) to elucidate the impact of varying parameter dimensions on computational overhead.

1) Local Training Overhead: In analyzing the computational overhead associated with the local training process, we focus on the variation in computational overhead across different numbers of iteration rounds and gradient vector dimensions. We set the number of local clients (N) to 600 and the gradient dimension $d = 1 \times 10^6$. Figure 3(a) and Table II show that our scheme exhibits superior computational efficiency to VeriFL across various iteration rounds when the convolutional kernel parameter, denoted as α, exceeds 50%. Optimal performance is achieved at a convolutional kernel setting of 100%, where the computational overhead is approximately 300 ms -significantly less than VeriFL's, marking a reduction to roughly $1/403$ of VeriFL's computational overhead. This enhanced efficiency is attributed to our framework's convolution operation, which serves as a mechanism for data dimensionality reduction, thereby significantly decreasing the dimensions required for hashing and lessening the computational load.



(a) (b)

Fig. 3. Comparison between our and VeriFL in terms of computational overhead associated with the local training process. (a) Overhead with different number of iterations. (b) Overhead with different dimensions d of a gradient.

TABLE II
COMPUTATIONAL OVERHEAD OF DIFFERENT KERNEL SIZES

Phase	Convolution kernel size (%)			
	25 %	50 %	75 %	100 %
Local training	8.859 s	10.315 s	5.521 s	25.598 ms
Verification	8.862 s	10.319 s	5.524 s	28.666 ms
Parameter sampling	9.195 s	10.535 s	5.762 s	29.344 ms

[1] The dimension $d = 1 \times 10^6$ of gradient, number of iterations $I = 10$, number of clients $N = 600$, and randomly select parameters layers $l = 5$.

The increased computational overhead with a 50% convolutional kernel parameter compared to 25% is due to the time consumed by the convolution operation itself. However, this is negligible compared to the time spent on hash operations at higher gradient dimensions. Figure 3(b) supports this inference, showing our scheme's superior performance over VeriFL when convolutional kernel parameters exceed 25%. With 75% and 100% convolutional kernel parameters, the computational overhead at $d = 1 \times 10^8$ is significantly reduced from 1338.615s to 187.819s and 3.698s for VeriFL, respectively. This highlights the efficiency of larger convolutional kernel parameters in reducing computational overhead.

2) Verification Process Overhead: The computational overhead during the verification process is similar to that of the

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

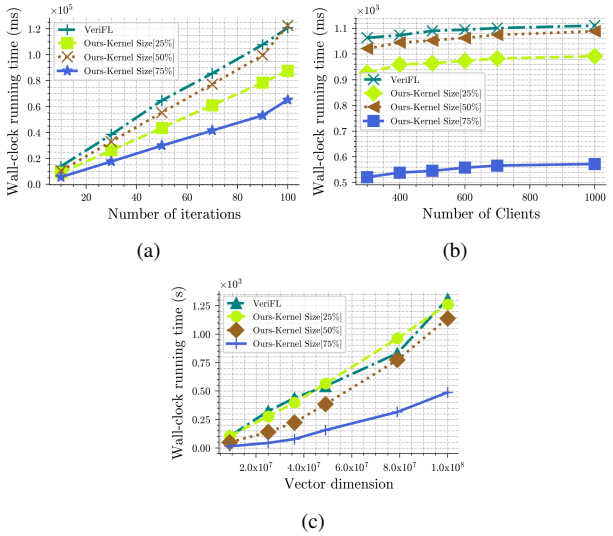


Fig. 4. Comparison between our and VeriFL in terms of computational overhead associated with the verification process. (a) Overhead with different number of iterations. (b) Overhead with different number of clients N. (c) Overhead with different dimensions d of a gradient.

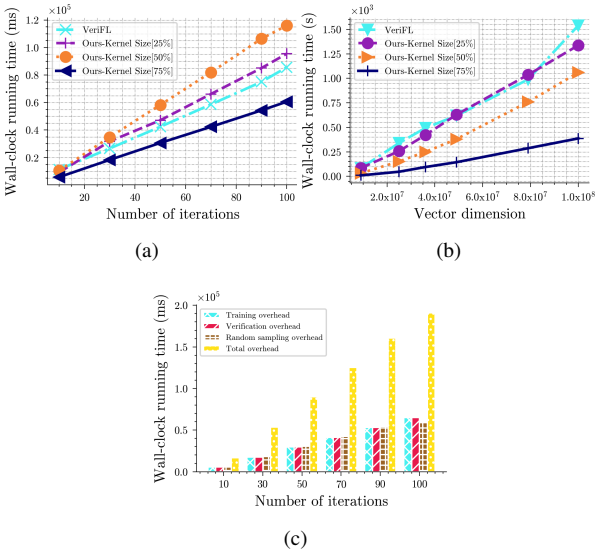


Fig. 5. Comparison between our and VeriFL in terms of computational overhead associated with the random layer parameter process. (a) Overhead with different number of iterations. (b) Overhead with different dimensions d of a gradient. (c) Comparison of the total computational cost with other parts for different iteration rounds.

training phase, with the main difference being the incorporation of multiple clients. Figure 4 provides a comparative analysis of computational overheads relative to the number of iterations, clients, and the gradient dimension, using a similar approach to the training phase for iterations and gradient dimensions.

When the convolutional kernel parameter is set below 50%, the disparity between our framework and VeriFL remains relatively narrow. However, a significant reduction in overhead is observed as the parameter is adjusted to 75%. Specifically, with the number of participants (N) at 1000, VeriFL's overhead registers at approximately 1110.694ms, whereas our framework's overhead is notably lower, at around 572.047ms, effectively halving the overhead in comparison. Furthermore, an increase in the convolutional kernel dimension promises even greater reductions in overhead.

This trend underscores the efficacy of our framework in mitigating computational overhead as user participation scales. It highlights the pivotal role of convolutional kernel parameter settings in optimizing the verification process, especially in scenarios involving a large number of users. The dramatic overhead decrease observed at a 75% convolutional kernel parameter setting demonstrates the potential for significant efficiency improvements, further affirming the advantage of our framework over traditional methods like VeriFL in handling verification processes with higher user counts.

3) Random Layer Parameter Overhead: In our scheme's training, each participant trains a single layer of the model at a time, requiring the transmission of updated parameters for the untrained layers back to them, along with randomly selected layer parameters from the server model. This introduces an additional overhead associated with the encryption of layer parameters. For this experiment, we set the number of randomly selected layers (l) to 5. Figure 5(a) shows that our scheme outperforms VeriFL in terms of overhead as the number of iterations increases, especially when the convolutional kernel parameter, denoted as α, is set to 75%. At 100 iterations, the overhead for VeriFL, denoted as $\mathcal{O}_{\text{VeriFL}}(100)$, is approximately 85.645s, compared to our scheme's overhead, denoted as $\mathcal{O}_{\text{LaVFL}}(100)$, which is 60.616s. The higher overhead observed when the convolution kernel parameter α is below 50%, similar to earlier analyses, is due to the insufficiently high gradient dimension, denoted as d_{grad}, which does not allow the convolution operation's time consumption to be disregarded. However, Figure 5(b) shows that as the gradient dimension increases, our scheme's efficiency significantly surpasses that of VeriFL. Finally, Figure 5(c) compares the total computational overhead depending on the number of iteration rounds with the other parts. However, this is not the final computational overhead, and the PGS method will be added in Section V-D to further reduce the overall overhead of the process.

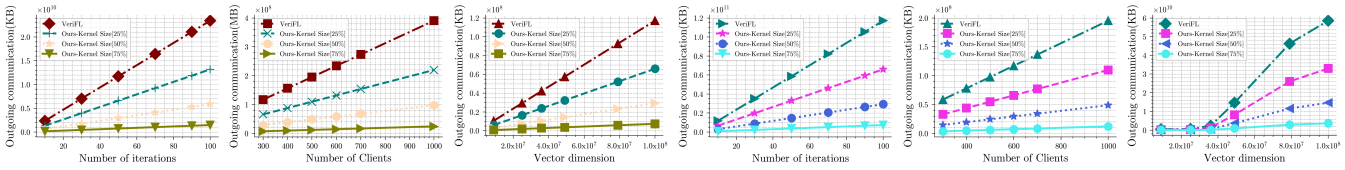
C. Communication Overhead

Communication overhead can be categorized into two main types: participant-to-participant and participant-to-server overheads. Since our scheme effectively eliminates communication overhead during the verification phase, we focused our evaluation on the performance implications of communication overhead for local training and random layer parameter phase. Specifically, we analyze scenarios in which the gradient dimension d, is set to 1×10^8.

1) Local Training Overhead: This segment presents a comparative analysis of communication overhead across varying numbers of iterations, participant counts, and gradient dimensions. Specifically, an increase in the convolution kernel dimension inversely correlates with the communication overhead incurred in each iteration cycle. As illustrated in Figure. 6(a), 6(b), 6(c), a significant difference is observed between our scheme and VeriFL when the convolution kernel

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728



(a) (b) (c) (d) (e) (f)

Fig. 6. Comparison of communication overhead between our method and VeriFL during the local training and random layer parameter processes. (a)-(c) Overhead variations in local training with respect to different iterations, number of clients N, and gradient dimensions d. (d)-(f) Overhead variations in parameter with respect to different iterations, number of clients N, and gradient dimensions d.

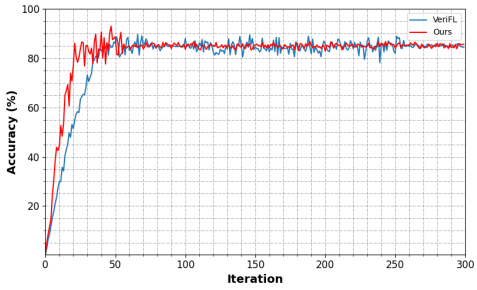


Fig. 7. Accuracy comparison between VeriFL and our LaVFL.

parameter is set at 75%, indicating a substantial reduction in the communication overhead among clients. At a convolution kernel parameter of 100% and with the number of iterations fixed at 100, the communication overhead is approximately 71.239 MB, which is only 3.112×10^{-6} times that of VeriFL. Moreover, despite the increase in both the number of clients and the gradient dimensions, which concurrently escalates the total communication overhead, our scheme consistently outperforms VeriFL.

2) Random Layer Parameter Overhead: The incorporation of random layer parameter necessitates an l-fold increase in communication overhead relative to the training phase. As depicted in Figure. 6, the overarching trend in overhead observed during the training phase is evident, with our scheme markedly surpassing VeriFL when the convolutional kernel parameter is set at 75%. Figure. 6(f) illustrates that the communication overhead for all methods intensifies in tandem with the vector dimension. The VeriFL approach, in particular, demonstrates a steep, almost linear surge in overhead as the dimension enlarges. Irrespective of the kernel size, our methodology maintains a minimal overhead compared to VeriFL—a disparity that becomes increasingly pronounced as the dimension approaches the 1×10^8 threshold.

D. PGS Verification Probability Assessment

To demonstrate probabilistic feasibility, we establish a lower bound on the probability P_{dc}, critical for the final verification's success. Our approach during convolution and encryption phases involves sampling varying gradient dimensions from each local client's l-th layer gradient matrix W_l in every iteration. This sampling detection strategy aims to increase the probability of validating each iteration, thus enhancing the overall success rate of the final verification phase.

We hypothesize that the quantity of W_l sampled in each iteration is proportional to r_{Er_G}, where d_{Er_G} represents the gradient dimensions sampled per iteration. The total dimensions across all clients, denoted as ct_{Er_D}, and the unattainable dimensions per iteration, denoted as f_v, are also considered. The sampling strategy ensures that even if some dimensions are non-validatable in a given iteration, the process can still advance efficiently by leveraging the available data, thus maintaining continual verification successes.

With the established lower bound for P_{dc}, we aim to show that this method consistently secures a high probability of successful verification, as depicted in Figure 8(a). The probability P_{dc} can be calculated based on the fraction of inaccessible dimensions relative to the total dimensions. As the fraction of unusable dimensions or the absolute number increases, P_{dc} decreases, reflecting how unavailable data dimensions negatively impact the success probability of the verification.

However, by carefully selecting the sample dimension, it is possible to maintain P_{dc} at a favorable level, thereby enhancing the likelihood of successful verification outcomes. This balance ensures that even with some dimensions being inaccessible, the overall verification process remains robust and effective.

E. Accuracy

The accuracy comparison between the VeriFL and our LaVFL is illustrated in the Figure 7. As shown, both methods achieve similar accuracy levels after sufficient iterations. Initially, our method converges more rapidly, reaching higher accuracy faster than VeriFL. This rapid convergence demonstrates the effectiveness of our gradient sampling and aggregation techniques in maintaining model performance. These evaluations were conducted on the S2ORC domain subset dataset, which is a comprehensive collection of academic papers used to benchmark the performance of language models.

After around 50 iterations, both methods stabilize, achieving accuracy levels above 80%. This indicates that despite the reduced computational and communication overhead, our method does not compromise the model's accuracy. The fluctuations observed in the accuracy of both methods can be attributed to the inherent variability in federated learning scenarios, but overall, our method consistently matches or exceeds the performance of the conventional method.

F. Performance Evaluation with LLMs Parameters

This analysis compares the computational and communication overhead between original methods and the PGS method

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

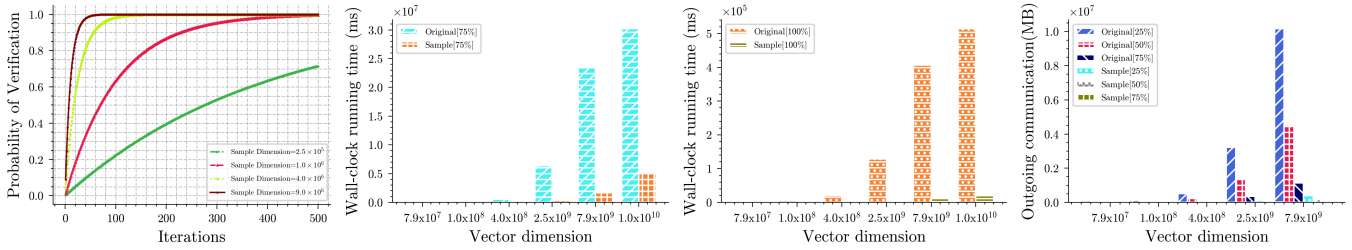


Fig. 8. (a) Verification probability and sampling iteration rounds. Comparison between our original and sampling methods in terms of computational overhead associated with the local training process. (b) Kernel Size[75%] with different dimensions d of gradients. (c) Kernel Size[100%] with different dimensions d of gradients. (d) Comparison between our original and sampling methods in terms of communication overhead associated with the local training process with different dimensions d of gradients.

for gradient dimensions up to 1×10^{10}. It demonstrates the PGS method's effectiveness in handling the high-dimensional gradients characteristic of LLMs such as BERT, GPT-3, and their derivatives. These models typically consist of hundreds of millions to billions of parameters, necessitating efficient methods to manage their complexity in a federated learning setup.

Figure 8(b) shows that for a 75% convolutional kernel parameter sampling, the runtime of the conventional method scales linearly with the vector dimension. In contrast, the sampling method exhibits a markedly reduced runtime, indicative of sub-linear growth. This demonstrates the substantial efficiency gains achieved by the PGS method, which is crucial for managing the training and updating processes of LLMs. Figure 8(c) further illustrates that at 100% kernel sampling, the sampling method maintains a nearly constant runtime, unlike the linearly increasing runtime of the conventional method, thereby showing significant efficiency improvements at high dimensions. For instance, at 75% kernel sampling, the conventional method's runtime is approximately three times that of the sampling method, indicating a tenfold increase in computational efficiency.

As depicted in Figure 8(d), the conventional method's communication overhead increases linearly with the vector dimension. In contrast, the sampling method's overhead grows sub-linearly, either logarithmically or with a sub-linear exponent. This sub-linear growth is particularly beneficial for LLMs, which generate large amounts of gradient data that need to be efficiently communicated between clients and servers in a federated learning environment. The percentage reduction in communication overhead is significant. For instance, at a 75% sampling rate, the communication overhead can be reduced by 90%, indicating a tenfold improvement in communication efficiency at a 50% sampling rate.

G. Discussion

While our experiments (600 clients for training, 1000 for verification) demonstrate that LaVFL's use of convolutional kernels can halve computational overhead compared to VeriFL at a 75% kernel setting, scalability to thousands of clients still poses challenges in communication and coordination. Although the convolution step reduces gradient dimensionality, transmitting even reduced-size gradients may become a bottleneck. The PGS strategy helps by sampling only a subset of gradient elements each round—thereby cutting both memory and communication costs—and could be combined with cluster- or hierarchy-based aggregation to further distribute load before global aggregation, ensuring that LaVFL remains efficient at very large client scales.

VI. Conclusions and Future Work

In this paper, we propose an Efficient Verifiable Federated Learning scheme (LaVFL) to address the verification challenges incurred by LLMs. LaVFL introduces novel methods, including layer-by-layer verification using the CGC method, a random layer parameter verification mechanism, and a PGS strategy. These methods ensure efficient and secure verification of LLM updates. Our security analysis and experimental results demonstrate the effectiveness of LaVFL in providing a secure and practical solution for verifying LLMs in FL settings.

LaVFL offers strong privacy and security in federated learning but depends on a centralized TA for parameter initialization and distribution, creating a single point of failure. To address this, future work will explore decentralized alternatives, such as blockchain and distributed key management, to enhance robustness. We also aim to extend LaVFL to defend against adaptive adversaries by optimizing the PGS strategy for dynamic threats. Moreover, we plan to investigate more efficient zero-knowledge proofs to reduce computational costs while preserving security, and integrate LaVFL with techniques like differential privacy and secure multi-party computation for broader privacy protection. These enhancements will make LaVFL more resilient, scalable, and adaptable in adversarial, privacy-sensitive settings.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] J. Konecnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, vol. 8, 2016.

[3] A. G. Roy, S. Siddiqui, S. Pölsterl, N. Navab, and C. Wachinger, "Braintorrent: A peer-to-peer environment for decentralized federated learning," *arXiv preprint arXiv:1905.06731*, 2019.

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728

[4] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 2019, pp. 954–964.

[5] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[6] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: Query processing for location services without compromising privacy," in *VLDB*, vol. 6, 2006, pp. 763–774.

[7] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018.

[8] C. Collberg, S. Martin, J. Myers, and J. Nagra, "Distributed application tamper detection via continuous software updates," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 319–328.

[9] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, pp. 1–18, 2014.

[10] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[11] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, "Large language models in medicine," *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023.

[12] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, 2020.

[13] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, vol. 78, no. 110, pp. 1–108, 1998.

[14] X. Guo, Z. Liu, J. Li, J. Gao, B. Hou, C. Dong, and T. Baker, "V eri fl: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2020.

[15] A. Patra and A. Suresh, "Blaze: blazing fast privacy-preserving machine learning," *arXiv preprint arXiv:2005.09042*, 2020.

[16] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 39–56.

[17] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 35–52.

[18] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[19] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," *arXiv preprint arXiv:2009.00349*, 2020.

[20] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 724–738.

[21] C. Dwork, "Differential privacy," in *International colloquium on automata, languages, and programming*. Springer, 2006, pp. 1–12.

[22] D. Olszewski, A. Lu, C. Stillman, K. Warren, C. Kitroser, A. Pascual, D. Ukirde, K. Butler, and P. Traynor, "" get in researchers; we're measuring reproducibility": A reproducibility study of machine learning papers in tier 1 security conferences," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 3433–3459.

[23] U. Fiege, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 210–217.

[24] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 947–964.

[25] A. L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, and P. Camacho, "{VeriZexe}: Decentralized private computation with universal setup," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4445–4462.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[27] Y. Pang, M. Sun, X. Jiang, and X. Li, "Convolution in convolution for network in network," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 5, pp. 1587–1597, 2017.

[28] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM workshop on artificial intelligence and security*, 2019, pp. 1–11.

[29] Y. Wang, A. Zhang, S. Wu, and S. Yu, "Vosa: Verifiable and oblivious secure aggregation for privacy-preserving federated learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[30] J. Zhao, H. Zhu, F. Wang, R. Lu, Z. Liu, and H. Li, "Pvd-fl: A privacy-preserving and verifiable decentralized federated learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2059–2073, 2022.

[31] S. Gao, J. Luo, J. Zhu, X. Dong, and W. Shi, "Vcd-fl: Verifiable, collusion-resistant, and dynamic federated learning," *IEEE Transactions on Information Forensics and Security*, 2023.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[33] H. L. Xin'ao Wang, K. Chen, and L. Shou, "Fedbfpt: An efficient federated learning framework for bert further pre-training," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 2023, pp. 4344–4352.

[34] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. Le, "Rethinking pre-training and self-training," *Advances in neural information processing systems*, vol. 33, pp. 3833–3845, 2020.

[35] E. Cambria and B. White, "Jumping nlp curves: A review of natural language processing research," *IEEE Computational intelligence magazine*, vol. 9, no. 2, pp. 48–57, 2014.

[36] T. Fan, Y. Kang, G. Ma, W. Chen, W. Wei, L. Fan, and Q. Yang, "Fate-llm: A industrial grade federated learning framework for large language models," *arXiv preprint arXiv:2310.10049*, 2023.

[37] D. C. Nguyen, S. Hosseinalipour, D. J. Love, P. N. Pathirana, and C. G. Brinton, "Latency optimization for blockchain-empowered federated learning in multi-server edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 12, pp. 3373–3390, 2022.

[38] V. Singh, R. Aljundi, and E. Belilovsky, "Controlling forgetting with test-time data in continual learning," *arXiv preprint arXiv:2406.13653*, 2024.

[39] D. E. Knuth, "The genesis of attribute grammars," in *Attribute Grammars and their Applications: International Conference WAGA Paris, France, September 19–21, 1990 Proceedings*. Springer, 2005, pp. 1–12.

[40] S. Berndt and M. Liśkiewicz, "Algorithm substitution attacks from a steganographic perspective," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1649–1660.

[41] W. Xiong and R. Lagerström, "Threat modeling–a systematic literature review," *Computers & security*, vol. 84, pp. 53–69, 2019.

[42] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," *Advances in neural information processing systems*, vol. 30, 2017.

[43] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International conference on machine learning*. Pmlr, 2018, pp. 5650–5659.

[44] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

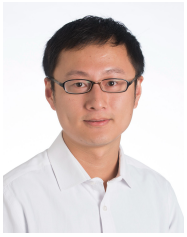
[45] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[46] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. S. Weld, "S2orc: The semantic scholar open research corpus," *arXiv preprint arXiv:1911.02782*, 2019.

Tianyou Zhang (S'23) received the B.S. degree in Computer Science and Technology from JiangXi University of Science and Technology, Jiangxi, China, in 2023. He is currently pursuing the Master's degree of Computer Science and Technology at Beijing University of Technology, Beijing, 100124, China. His research interests include cloud storage security, Internet of things security, privacy-preserving computing and data privacy. He is a student member of IEEE.

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

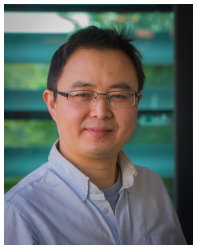
This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2025.3581728



Haiyang Yu (M'21) received the PhD degree in computer science and technology from Beijing University of Technology in 2019. From 2017 to 2018, he visited the University of Melbourne in Australia. He has authored or coauthored more than 20 papers in highly ranked journals and top conference proceedings. He is currently an assistant professor of computer science at Beijing University of Technology. His research interests include cloud storage auditing, blockchain, vehicular network and data mining. He is a member of IEEE and ACM.

Zhen Yang (M'15) received the Ph.D. degree in signal processing from the Beijing University of Posts and Telecommunications, Beijing, China. He is currently the Full Professor of computer science and engineering with the Beijing University of Technology, Beijing, China. He has authored or coauthored more than 30 papers in highly ranked journals and top conference proceedings. His research interests include data mining, machine learning, trusted computing, and content security. He is the Senior Member of the Chinese Institute of Electronics.

Yuwen Chen received the M.S. degree in computer software and theory from Zhengzhou University, Zhengzhou, China, in 2015, and the Ph.D. degree in telematic engineering form the Technical University of Madrid, Madrid, Spain, in 2019. He is a Lecturer of Computer Science and Engineering with the Beijing University of Technology, Beijing, China. His research interests include IoT security and privacy, and smart grid privacy and security.



Shui Yu (F'23) received the Ph.D. degree from Deakin University, Melbourne, VIC, Australia, in 2004. He is currently a Professor with the School of Computer Science, University of Technology Sydney, Sydney, NSW, Australia. He has authored or coauthored four monographs and edited two books, and more than 400 technical papers, including top journals and top conferences, such as IEEE TPDS, TC, TIFS, TMC, TKDE, TETC, ToN, and INFOCOM. His research interests include big data, security and privacy, networking, and mathematical modeling. Dr. Yu initiated the research field of networking for big data in 2013, and his research outputs have been widely adopted by industrial systems, such as Amazon cloud security. He was an associate editor for the IEEE Transactions on Parallel and Distributed Systems. He is currently on a number of prestigious editorial boards, including IEEE Communications Surveys and Tutorials (Area Editor), IEEE Communications Magazine, and IEEE Internet of Things Journal. He was a Distinguished Lecturer of IEEE Communications Society during 2018–2021. He is a Distinguished Visitor of IEEE Computer Society, a voting Member of IEEE ComSoc Educational Services board, and an elected Member of the Board of Governor of IEEE Vehicular Technology Society.

Authorized licensed use limited to: Centro Universitário Fei. Downloaded on June 23,2025 at 19:07:24 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.