2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

TraceFL: Interpretability-Driven Debugging in Federated Learning via Neuron Provenance

Waris Gill
Computer Science Department
Virginia Tech
Blacksburg, USA
waris@vt.edu

Ali Anwar
Computer Science and Engineering Department
University of Minnesota
Minneapolis, USA
aanwar@umn.edu

Muhammad Ali Gulzar
Computer Science Department
Virginia Tech
Blacksburg, USA
gulzar@cs.vt.edu

***Abstract*—In Federated Learning, clients train models on local data and send updates to a central server, which aggregates them into a global model using a fusion algorithm. This collaborative yet privacy-preserving training comes at a cost. FL developers face significant challenges in attributing global model predictions to specific clients. Localizing responsible clients is a crucial step towards (a) excluding clients primarily responsible for incorrect predictions and (b) encouraging clients who contributed high-quality models to continue participating in the future. Existing ML debugging approaches are inherently inapplicable as they are designed for single-model, centralized training.**

We introduce TraceFL, a fine-grained *neuron provenance* capturing mechanism that identifies clients responsible for a global model's prediction by tracking the flow of information from individual clients to the global model. Since inference on different inputs activates a different set of neurons of the global model, TraceFL dynamically quantifies the significance of the global model's neurons in a given prediction, identifying the most crucial neurons in the global model. It then maps them to the corresponding neurons in every participating client to determine each client's contribution, ultimately localizing the responsible client. We evaluate TraceFL on six datasets, including two real-world medical imaging datasets and four neural networks, including advanced models such as GPT. TraceFL achieves 99% accuracy in localizing the responsible client in FL tasks spanning both image and text classification tasks. At a time when state-of-the-art ML debugging approaches are mostly domain-specific (*e.g.,* image classification only), TraceFL is the first technique to enable highly accurate automated reasoning across a wide range of FL applications.

***Index Terms*—Interpretability, Explainability, Debugging, Machine Learning, Federated Learning, Transformer**

I. Introduction

Federated Learning (FL) offers distributed training that enables multiple clients to collaboratively train a global model without sharing raw data [1]–[5]. In a typical FL setup, individual clients, such as healthcare institutions, train models on their local data. These local models are then aggregated on a central server to form a comprehensive global model, all without transferring sensitive client data. The resulting global model, a fusion of all clients' models, is then used in production to make predictions on unseen data.

The complexity of FL systems, however, introduces unique debugging challenges. When a global model makes a prediction, whether correct or incorrect, a key question arises: *which client(s) is primarily responsible for a global model's output?* This question is akin to debugging software, where understanding the impact of each input and the line of code on the software's output is crucial. Addressing this debugging question is vital for the effective deployment, maintenance, and accountability of FL applications. For example, FL developers face challenges in identifying and rewarding clients responsible for successful classifications. This recognition is crucial to encourage their continued participation in future incentivized FL rounds [6]. There is mature evidence that such practice significantly improves the FL model's quality [7]. Similar debugging is key in localizing *faulty clients* that may transfer an inaccurate model for aggregation, which can result in a dangerously low-quality global model [8]–[12].

Problem. *In federated learning, the client(s) most responsible for a global model's prediction are the ones trained on data that contains the predicted labels.* This is analogous to finding influential training samples in classical machine learning [13]. However, the two domains, single model-based centralized ML and FL, are fundamentally different. Existing influence-based debugging approaches in ML [14]–[17] and regular software [18]–[20] require transparent access to data including all data manipulation operations applied on the input data. When applied to FL, these approaches will require end-to-end monitoring of clients' training (*i.e.,* require access to clients' data), which is prohibited in FL. More broadly, ML influence and interpretability-based debugging approaches target a single model in which the debugging is restricted to identifying the training data. In contrast, debugging in FL entails isolating a client's model among many. This paper addresses the following debugging problem in FL: *Given the global model inference on an input in FL, how can we identify the client(s) most responsible for the inference?*

Challenges. Determining a client's influence on the global model is challenging. Clients are randomly sampled in each round, each possessing unique data and contributing differently to the global model. Thus, the influence of a client on the global model is dynamic, non-uniform, and changes across rounds, making it difficult to link the global model's behavior to a specific client. The FL protocol restricts access to client-side training, turning FL configuration into a nearly black-box setting. Additionally, clients' models are collections of neuron weights that are individually uninterpretable. Static

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00128

analysis of models' weights to measure clients' influence is ineffective because clients' models are intrinsically different in terms of weights. Furthermore, neural networks today comprise millions of neurons (*e.g.,* GPT-3 has 175 billion parameters [21]). Considering all neurons equally, in such cases, would lead to imprecise and incorrect debugging.

FL is increasingly used for domains other than vision using various neural networks, such as transformer and convolutional neural networks (CNNs). Designing a generic FL debugging approach is a major challenge. For instance, transformers contain a self-attention mechanism that allows the model to focus on different parts of the input sequence. This mechanism is usually not seen in CNNs; instead, it uses a convolutional layer to detect the special patterns in the input data. Additionally, these architectures use different activation functions such as Rectified Linear Unit (ReLU) [22] and Gaussian Error Linear Unit (GELU) [23], introducing another source of complications.

Our Contribution. We present the concepts of ***neuron provenance***, a fine-grained lineage-capturing mechanism that formulates the flow of information in the fusion algorithm from multiple clients' models into a global FL model, ultimately influencing the predictions of the global model. Using neuron provenance, we determine the precise magnitude of contributions of participating clients towards the global model's prediction. We materialize the idea of neuron provenance in TraceFL, which runs at the aggregator (*i.e.,* central server) and requires no instrumentation on the client side.

TraceFL is designed with the following insights. Since a global model consists of millions of neurons, we observe that a dynamic subset of neurons activates in response to a given input, and not all neurons contribute equally to a prediction [24], [25]. Using this insight, TraceFL quantifies the contribution of these neurons in the global model's prediction by computing the gradient of the neurons *w.r.t.* to the prediction. Such neuron-level gradients reveal the neuron's output impact on the global model's prediction and thus reduce the scope of important neurons.

TraceFL then maps the global model's important neurons to the corresponding neurons in each client's model and computes the contribution of each client's neuron to the corresponding global model's neuron. At this stage, TraceFL computes the end-to-end **neuron provenance** of the global model prediction with the magnitude of the contribution of each client's neurons. Finally, TraceFL aggregates the contributions of each client. The client with the highest contribution is deemed the most responsible for the given prediction.

Evaluations. We demonstrate TraceFL's effectiveness, generalizability, and robustness by evaluating its client localization accuracy on both image and language models under various commercial data distributions and differential privacy methods. We evaluate TraceFL on four state-of-the-art neural networks: ResNet [26], DenseNet [27], BERT [28], and GPT [29] and using six datasets including two real-world medical imaging datasets [30]–[32]. TraceFL achieves an average accuracy of 99% in localizing the responsible client across 30 unique FL settings, spanning both correct and misprediction scenarios. For fault localization in FL, TraceFL achieves an average accuracy of 99%, compared to 32% by the existing technique [33], demonstrating TraceFL's effectiveness in real-world FL deployments. Additionally, we test TraceFL's robustness against varying data distributions and differential privacy settings and find that TraceFL remains robust and effective. We also vary the number of clients, increasing it up to 1000, and find that TraceFL is both scalable and efficient. Overall, we evaluate TraceFL on 20,600 trained client models. These experiments exceed prior research's evaluation complexity and fully represent commercial FL usage [34]–[38]. TraceFL is implemented in Flower FL [39] and compatible with GPU for parallel processing of **neuron provenance** for compute-intensive models (*e.g.,* GPT).

TraceFL advances the state of FL debugging with the following core contributions:

- TraceFL localizes the responsible clients for a given prediction without modifying the underlying fusion algorithm. Moreover, it does not require access to clients' training and can solely determine clients' contributions at the central aggregator.
- TraceFL introduces a unique concept of ***neuron provenance*** for FL applications to capture the dynamic contribution of each client, which helps rank clients based on the contribution to a given prediction. TraceFL efficiently tracks the contribution of clients in large models like GPT containing millions of parameters.
- TraceFL achieves 99% localization accuracy in localizing the responsible client in FL. TraceFL's localization accuracy remains high during localizing a faulty client where existing baseline [33] achieves 32%.
- TraceFL is the first approach that is equally effective on transformers and CNNs. Even the most sophisticated ML debugging approaches work on single model architecture and data domains, *i.e.,* either CNNs or transformers.
- TraceFL significantly advances the field of debugging and interpretability in FL, addressing open challenges in FL [33], [40] and work with differential privacy and real-world data distributions among FL clients.

Source Code. TraceFL's artifact is available at https://github.com/SEED-VT/TraceFL.

II. Background and Motivation

A. Federated Learning

Federated Learning enables multiple clients (*e.g.,* mobile devices, organizations) to train a shared model without sharing their data. This allows the model to be trained using distributed data, which can be useful in cases where data is distributed across multiple devices or organizations and cannot be easily collected and centralized. One algorithm of FL is Federated Averaging (FedAvg) [1], which uses the following equation to update the global model at each round of the training process:

$$W_{global}^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} W_k^{(t)} \tag{1}$$

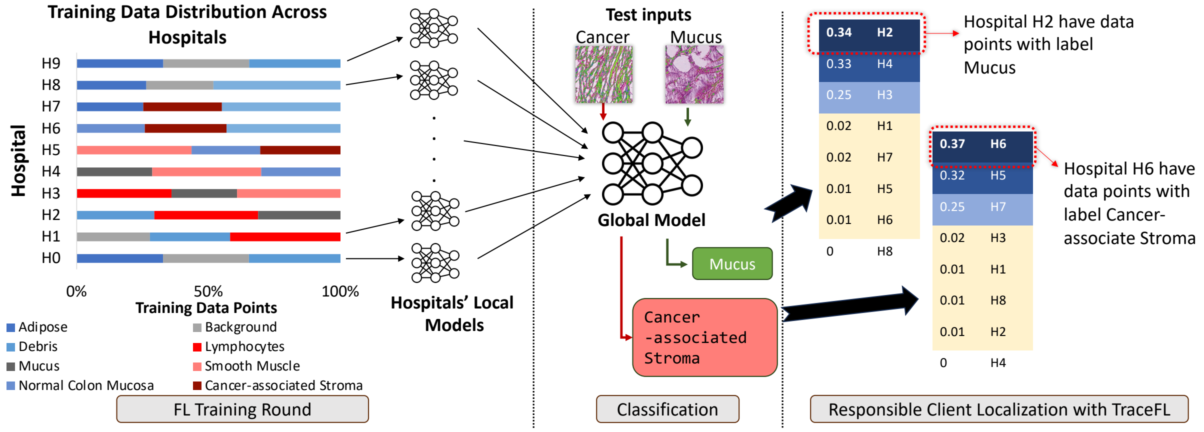

Fig. 1: Illustration of training, testing, and localization phases of the real-world motivating example. The FL global model correctly classifies two colon pathology images (original labels 'Cacner-associated Stroma' and 'Mucus'). During responsible client localization, TraceFL accurately identifies the client most responsible for the prediction, i.e., clients trained on data points with labels Mucus (Hospital H2) and 'Cancer-associated Stroma' (Hospital H6).

where $W_k^{(t)}$ and n_k represent received weights and size of training data of client k in each round t, respectively. The variable n represents the total number of data points from all clients, and it is calculated as $n = \sum_{k=1}^{K} n_k$. The equation states that the global model W^{t+1} at the next round is the average of the local models from all participating clients at the current round. In each round, the clients first train their local models using their own data, then send the parameters (*e.g.,* $W_k^{(t)}$, n_k) to the central server. The central server averages the model parameters to produce a global model, which is then sent back to the participating devices. This process is repeated for multiple rounds (*e.g.,* t from 1 to 100), with each client updating its local model using the global model from the previous round. The final global model is the result of the federated averaging process.

FL has variations such as Vertical FL [41] and Personalized FL [42]. TraceFL primarily focuses on horizontal FL [1], similar to previous work on fault localization in FL [10], [33]. A typical FL setup involves a few to thousands of clients, such as mobile devices, healthcare institutions, or enterprises. FL clients exhibit diversity in data distribution and computational resources. Data is often non-independently and identically distributed (Non-IID), with varying sizes across clients, such as hospitals specializing in different medical conditions (Figure 1). All participating clients use the same neural network architecture, ensuring compatibility during model aggregation. Additionally, clients have heterogeneous hardware and network capabilities (e.g., smartphones to powerful servers), impacting their participation and training consistency [43].

B. Motivation

Suppose a developer deploys an FL system to diagnose colon diseases based on colon pathology images, as shown in Figure 1. In this FL system, ten hospitals, identified as H0 to H9, collaborate to train a global FL model. Each hospital trains its local model, which is then aggregated to form a global model. The classification stage in Figure 1 shows a scenario where the global model makes a correct prediction on new test colon pathology images (*e.g.,* 'Cancer-associated Stroma' and 'Mucus'). Since these are correct predictions, the FL developer aims to determine which hospital is most responsible for these correct predictions so that it can be encouraged to participate in future rounds. Since the training data is protected under the privacy of medical records and inaccessible to the developer, it is challenging to identify the responsible hospital by inspecting the raw model weights shared by the hospitals.

To address this issue, the developer decides to use TraceFL to identify the most responsible client behind the correct predictions. When enabled during the global model's prediction, TraceFL localizes the hospital with H2 as the one responsible for the prediction of 'Mucus', and the hospital H6 as the one responsible for the prediction of 'Cancer-associated Stroma'. More specifically, as shown on the right in Figure 1, TraceFL ranks hospitals (*i.e.,* clients) based on their contributions to each prediction. The score associated with each hospital quantifies how responsible a hospital is for that prediction. The training data distribution on the left shows that H2's training data include data points labeled 'Mucus', whereas H6's training data include data points labeled 'Cancer-associated Stroma'.

Conversely, hospital H1's contribution is 0.02 and 0.01 in the two predictions because it does not have any data points with the labels 'Mucus' or 'Cancer-associated Stroma', respectively. Detailed evaluations on two real-world medical imaging datasets are presented in Section V-A. Localizing the clients responsible for the prediction with TraceFL serves as a basis for designing advanced incentivization approaches.

III. Challenges in Debugging FL

Federated Learning poses several challenges in designing a debugging technique that reasons about a global model's prediction on an input. Unlike traditional ML training, where training data can be easily analyzed, the FL global model (W_{global}^{t+1}) is not directly trained on the data. Instead, the global model is generated by fusing clients' models together across many rounds using popular fusion algorithms. With no insight into the training, it is challenging to identify how different clients influence the global model's behavior.

State-of-the-art neural networks, such as Transformers (BERT, GPT) and CNNs (ResNet, DenseNet), have varying structures, activation functions, and numbers of parameters. For example, GPT [29] is a 37-layer (12 block) Transformer architecture with GELU [23] activation function and 117 million parameters. DenseNet [27] has 121 layers, 8 million parameters, and uses the ReLU [22] activation function. The fusion algorithm or the global model does not inherently provide any information about individual clients' contributions. Thus, tracking a client's contribution among millions of parameters is a significantly challenging task.

Moreover, the data distribution across FL training rounds is non-identical; clients rarely have the same data point. Not all clients participate in every round, and some clients may contain only a few data points to train their local model. The class label distribution also varies across clients. Such variability causes more hurdles in precisely reasoning about a global model's behavior. Simply tracking the static weights of the global model is inadequate, as different sets of neurons are activated on different inputs, and not all neurons have equal importance. Inspecting individual clients' models does not help understand the client's contribution to the global model prediction, as it will not capture the cumulative behavior of the global model.

IV. Design of TraceFL

TraceFL addresses the aforementioned challenges using ***neuron provenance***. At a high level, TraceFL dynamically tracks the lineage of the global model at the neuron level and identifies the most influential clients against a given prediction by the global model (W_{global}^{t+1}) on an input. Enabling provenance at the neuron level solves the complexities of different neural networks architectural design (*e.g.,* number of layers and parameters, different activation functions) and enables TraceFL to work in cross domains such as image and text classification tasks.

In essence, TraceFL first identifies the influential neurons by jointly analyzing neuron activations, the layers the neurons are in, and gradients with respect to individual neurons in the global model for a given test input. Next, TraceFL precisely quantifies the individual contribution of each corresponding neuron from every participating client to the neurons in the global model. Finally, TraceFL computes the total contribution of each client to the global model. These steps collectively construct a comprehensive end-to-end provenance graph, which is used to debug the contributions of clients in the given prediction of the global model. Algorithm 1 outlines the design of TraceFL.

A. Determining Influential Neurons

TraceFL first aims to identify neurons that actively participate in an FL global model's prediction. Traditional data provenance approaches must trace the participation of all input data records in the operation for completeness, eventually mapping them to individual outputs of the operation. However, tracking the provenance of all neurons with equal importance is wasteful because not all neurons participate equally in a model's prediction. Therefore, tracking the behavior of neurons with the same importance in a model may lead to over-approximation (*i.e.,* more than expected clients are classified as contributors) when provenance is used to identify the contributing clients.

Algorithm 1: TraceFL's Approach

```
Input: Let $clients$ be the list of clients' models participating in the
       FL training round.
Input: Let $global\_model$ be the aggregated global model of
       $clients$ models after the end of a training round
Input: Let $test\_input$ be an input
Output: $client2norm\_contribution$ contains the contribution of
        each client in the prediction of $test\_input$
activated_neurons = [ ];
// Section IV-A (Equation 2)
$y$ = global_model(test_input);
for each neuron in global_model do
    activated_neurons.append(neuron);
neuron2grad = $y$.backward();
// Section IV-B
neuron2prov = {};
for neuron in activated_neurons do
    neuron2prov[neuron] = {};
    for client in clients do
        cont = 0;
        for feature in neuron.input_features do
            cont+ = client.weight(neuron, feature) $\times$ feature.value;
        neuron2prov[neuron][client] = cont $\times$ neuron2grad[neuron];
// Section IV-C
client2contribution = {};
for client in clients do
    // Equation 6
    contribution = 0;
    for neuron in neuron2prov do
        contribution+ = neuron2prov[neuron][client];
    client2contribution[client] = contribution;
// Equation 7
client2norm_contribution = Softmax(client2contribution.values);
return client2norm_contribution;
```

The behavior of a neural network on a given input is determined by the set of activated neurons in the network, and different sets of neurons are activated on different inputs. We leverage this insight and apply TraceFL's neuron provenance to dynamically quantify the influence of global model neurons on each prediction for the given input. This reduces the likelihood of over-approximation by minimizing the contribution of neurons that may distort the outcome when the lineage of a specific neuron is used to localize the influential client.

Mathematically, the output of a neuron is $z = \sigma(\mathbf{w} \cdot \mathbf{z})$, where $\mathbf{w}$ is the set of weights of the neuron, $\mathbf{z}$ is the input to a neuron, and σ is the activation function. One of the commonly used activation functions (σ) is ReLU [22]. The output of σ is called the activation or output of the neuron. A neuron with ReLU function is considered active if $z > 0$. Note that the output of a neuron (z) is part of the input to the neurons of the next layer. Next, TraceFL computes the activation of each neuron in the network. Suppose that n_j represents the j-th neuron in a neural network and the set of all the outputs (*i.e.,* activations) of all the neurons in a neural network can be represented as $\{z_{n_1}, z_{n_2}, ..., z_{n_j}\}$, which captures the complete dynamic behavior of the network on a given test input $\mathbf{x}$. Note that for the first layer, the input $\mathbf{z}$ to neurons will be the input $\mathbf{x}$ to the model *i.e.,* $\mathbf{z} = \mathbf{x}$ for the first layer of the neural network.

After computing the global model neurons activations, TraceFL's goal is to find their measurable contribution towards the global model's prediction (y) on an input. In the output (y) of the global model, not all the neurons carry equal importance. For instance, neurons in the last layers learn better and more rigorous features than neurons in the initial layers of the network [24]. Since TraceFL aims to localize the client that contributed the most towards a prediction, assigning equal importance to all neurons will again cause over-approximation or even wrong client localization. To enable precise and accurate provenance, we must measure the individual influence of a neuron on the final prediction.

TraceFL quantifies the impact of the output of a neuron on the global model's prediction by computing the gradient *w.r.t.* every activated neuron on a given input to W_{global}^{t+1}. Similar to taint analysis in program analysis, gradients are sophisticated taints that encapsulate the impact of a neuron output on the output (y) of the global model. The intuition behind this is that the neurons with a higher gradient will likely cause a bigger change in prediction. Thus, such neurons are likely to be more influential to a model prediction. We use the aforementioned insight to find the influence of a neuron in the prediction (y) of the global model. The influence, denoted by c_{n_j}, of a neuron n_j in the output (y) is the partial derivative of y with respect to z_{n_j}, which measures how much y changes when z_{n_j} changes slightly. Mathematically, we write it as:

$$c_{n_j} = \frac{\partial y}{\partial z_{n_j}} \tag{2}$$

TraceFL computes the gradients using the automatic differentiation engine of PyTorch [44]. TraceFL starts from the output layer and goes back to the input layer, using the chain rule of differentiation at each step. By the end of this phase, TraceFL determines the gradient (influence) of global model neurons on its output (y). For instance, in the presence of a disease in a medical imaging input (*e.g.,* predicting colorectal cancer (CRC) from histological slides of tumor tissue), the fused neurons of the global model that have learned the representation of that particular disease during FL training will significantly influence the model's output (y). These gradients are essential in mapping neurons of clients' models to the most influential ones in the global model.

B. Neuron Provenance Across Fusion

In this step, TraceFL accurately determines the individual contribution of each corresponding neuron from every participating client to the neurons of the global model. In essence, TraceFL maps the outputs of the global model neurons to clients' neurons during prediction. Finding such a mapping and its magnitude has two challenges. First, FL uses fusion algorithms to merge clients' neurons statically. Instrumenting the fusion algorithms to trace the flow of weights across fusion is prohibitively expensive, as numerous clients participate in a round where each model may have millions of neurons. Second, the influence of clients' neurons on the neurons of the global model (W_{global}^{t+1}) is directly impacted by the output of the preceding layer in the global model, *i.e.,* the output of the neuron in the global model's previous layer is the combined output of the corresponding neurons of each client in that layer. Consequently, attempting to determine clients' neurons' contributions by feeding input to the clients' model in isolation will lead to incorrect neuron provenance, as it cannot capture the overall impact of other clients.

TraceFL leverages the insight that the set of weights of a single neuron in the global model is determined by the corresponding weights of the neurons in the clients' models. Mathematically, the weights of a single neuron in the global model, represented as $\mathbf{w_g} = [w_g^1, w_g^2, \cdots, w_g^i]$, are given by the following equation:

$$\begin{aligned} w_g^i &= \sum_{k=1}^{K} p_k * w_k^i \\ &= p_1 * w_1^i + p_2 * w_2^i + \cdots + p_k * w_k^i \end{aligned} \tag{3}$$

Here, w_k^i is the i-th weight of the neuron in the k-th client model. The variable p_k is n_k/n, where n_k represents the size of training data of client k, and n represents the total number of data points from all clients, and it is calculated as $n = \sum_{k=1}^{K} n_k$ (Equation 1). Given an input $\mathbf{z}$ to the neuron $\mathbf{w_g}$ of W_{global}^{t+1}, a client's contribution can be calculated as follows:

$$\begin{aligned} z_{out} &= \mathbf{w_g} * \mathbf{z} \\ &= [w_g^1, w_g^2, \cdots, w_g^i] * [z^1, z^2, \cdots, z^i] \\ &= w_g^1 * z^1 + w_g^2 * z^2 + \cdots + w_g^i * z^i \\ &= [p_1 * w_1^1 + p_2 * w_2^1 + \cdots + p_k * w_k^1] * z^1 \\ &+ [p_1 * w_1^2 + p_2 * w_2^2 + \cdots + p_k * w_k^2] * z^2 \\ &+ \cdots \\ &+ [p_1 * w_k^i + p_2 * w_2^i + \cdots + p_k * w_k^i] * z^i \end{aligned} \tag{4}$$

Here, z^i is the i-th input feature to the neuron and z_{out} is the output of the neuron. Thus, the contribution of a client k, denoted by $[t_k]$, in a neuron n_j of the global model (W_{global}^{t+1}) is given by the following equation:

$$\begin{aligned} [t_k]_{n_j} &= (p_k * w_k^1 * z^1 + p_k * w_k^2 * z^2 + \cdots \\ &+ p_k * w_k^i * z^i) * c_{n_j} \\ &= (p_k * [w_k^1 * z^1 + w_k^2 * z^2 + \cdots + w_k^i * z^i]) * c_{n_j} \\ &= c_{n_j} * p_k * \sum_{i=1} w_k^i * z^i \end{aligned} \tag{5}$$

In the above equation, $p_k * \sum_{i=1} w_k^i * z^i$ is the exact contribution of a client k in a neuron n_j of the global model.

The global gradient of neuron n_j is c_{n_j} which is multiplied with client contribution to find its actual contribution (*i.e.,* influence) towards the prediction of the global model. For instance, if the contribution of a client k is high in a neuron n_j but globally the neuron n_j has minimal influence on the global model prediction then c_{n_j} will scale down the contribution of the client in the given neuron n_j. Note that z^i represents the i-th output of the previous layer in the global model during prediction. At the end of this stage, TraceFL constructs a **neuron provenance** graph that traces a global model's prediction to influential neurons in the global model (W^{t+1}_{global}), which are further traced back to individual neurons in the clients' models.

C. Measuring Client's Contribution

To find the end-to-end contribution, we must accumulate neuron-level provenance, $c_{n_j} * p_k * \sum_{i=1} w^i_k * z^i$, of a given client's model to derive its complete contributions toward the global model's prediction. A client's overall contribution to the global model prediction is determined by the sum of the client's contribution to the neurons of the global model. Specifically, if the set of neurons of the global model is denoted by $n_1, n_2, \cdots, n_j$, then the total contribution (T_k) of the client k can be calculated using Equation 5 as follows:

$$\begin{aligned} T_k &= \beta_{n_1} * [t_k]_{n_1} + \beta_{n_2} * [t_k]_{n_2} + \cdots + \beta_{n_j} * [t_k]_{n_j} \\ &= ([c_{n_1} * \sum_{i=1} w^i_{k\_n_1} * z^i_{n_1}]_{n_1} + [c_{n_2} * \sum_{i=1} w^i_{k\_n_2} * \\ &\quad z^i_{n_2}]_{n_2} + \cdots + [c_{n_j} * \sum_{i=1} w^i_{k\_n_j} * z^i_{n_j}]_{n_j}) * p_k \end{aligned} \tag{6}$$

β is an importance factor that TraceFL computes using an exponential decay method for each neuron based on its position in the neural network. Specifically, TraceFL assigns higher importance to the last layers and lower importance to the earlier layers to minimize the noisy contributions, based on the evidence presented elsewhere [24]. $[t_k]_{n_j}$ is the contribution of the client k in neuron n_j, $z^i_{n_j}$ is the i-th input feature to neuron n_j, and $w^i_{k\_n_j}$ is the i-th weight of neuron n_j in the client k model. Using Equation 6 we can compute, for each client k, the total contribution towards the global model prediction. Thus, the client with max contribution is the client that has the most influence on the global model prediction. To make the client contribution more interpretable, we normalize the client contribution by using the softmax function as follows:

$$\tilde{T_k} = \frac{e^{T_k}}{\sum_{i=1}^{K} e^{T_i}} \tag{7}$$

$\tilde{T_k}$ is the normalized contribution of the k-th client, which is now a probability value between 0 and 1, representing the relative influence of client-k on the global model output y for a given input.

TraceFL concludes its **neuron provenance** capturing technique by listing the total contribution of each participating client in an FL round towards a global model's prediction on a given input. The magnitude of the contributions can be interpreted as a confidence level of TraceFL in identifying the source of the global model's prediction. Given that the total confidence scores of all clients cannot exceed 1, if a client has a contribution score of 0.6, it implies that no other client can surpass a score of 0.4. This makes the client most influential in determining the global model prediction and most likely responsible for the prediction.

Enable TraceFL to Use GPU. By design, TraceFL is compatible with hardware accelerators and can fully harness their parallelizability. The primary dependency of TraceFL is capturing the output of previous layer neurons in the global model for input to the next layer neurons, which inherently exists in inference as well. Additionally, TraceFL computes gradients using Equation 2, leveraging the chain rule of differentiation that the hardware accelerators can parallelize. Next, Equation 5 dissects the global model neuron and computes the contribution based on the previous layer neurons' outputs ($\mathbf{z}$) of the global model, which is the cumulative output of all clients' neurons in that previous layer. This is the only dependency in TraceFL. Once TraceFL has the cumulative output from the previous layer neurons, it parallelizes the process to find the contribution of a client in each neuron of the global model in the current neuron layer and ultimately the total contribution using Equations 6 and 7. These optimizations in TraceFL enable neuron-level provenance for neural networks primarily deployed on GPUs, such as GPT.

V. Experimental Evaluations

We design experiments to evaluate TraceFL's accuracy in localizing the client responsible for a global model's prediction on an input. We ask the following research questions.

- How accurate is TraceFL in identifying the client(s) responsible for a global model's prediction?
- Is TraceFL equally accurate on FL of different models and architectures such as CNNs and transformers (GPT)?
- How accurate TraceFL is in localizing clients responsible for mispredictions by global model?
- Does TraceFL remain effective with varying data distributions and differential privacy?
- Can TraceFL scale to a large number of clients?
- What is the runtime performance of TraceFL?

Models and Datasets. We evaluate TraceFL on state-of-the-art and commercially used CNNs, including ResNet-18 [26] and DenseNet-121 [27], as well as the two most popular transformer models, BERT [28] and GPT [29] to demonstrate the wide applicability of TraceFL. We train ResNet and DenseNet on CIFAR-10 [45] and MNIST [46]. These network-dataset combinations are widely used and serve as standardized benchmarks in practice [1], [37]. We also evaluate TraceFL on real-world medical imaging datasets, including the Colon Pathology dataset [30] and Abdominal CT dataset [31], [32], to demonstrate its usability in complex real-world FL systems. The Colon Pathology dataset contains 107,180 biomedical images representing nine classes of colon pathology, while the Abdominal CT dataset contains 58,830 images of abdominal CT scans representing 11 classes. More details about these datasets can be found in [47], [48]. For NLP tasks, we evaluate TraceFL on BERT and GPT models trained on the DBpedia and Yahoo Answers datasets [49]. The DBpedia dataset contains 560,000 training samples and 70,000 testing samples, while the Yahoo Answers dataset contains 1,400,000 training samples and 60,000 testing samples, representing 14 and 10 classes, respectively.

Data Distribution Among Clients. We use Dirichlet distribution in FL to distribute non-overlapping data points among clients in each round. This is the standard FL data distribution method proven to produce real-world distribution [37], [38], [50], [51]. The parameter (α) in Dirichlet ranges from [0, ∞), determining the level of Non-IID in experiments. For instance, when α equals 100, it replicates uniform local data distributions, while smaller α values increase the probability that clients possess samples from a single class [51]. A value of 0.5 is a common practice in prior work [37], [50]. We use an even stricter parameter value of 0.3 to stress test TraceFL and demonstrate its usability in more challenging cases. Nevertheless, Section V-C1 performs sensitivity analysis by varying α from 0.1 to 1. These settings inherently simulate varying degrees of label overlap among clients. To explicitly manage overlapping labels, pathological data distributions can be employed [37], as shown in Figure 1. The pathological data distribution is available in TraceFL's artifact. Furthermore, TraceFL's artifact contains configurable data distributions among clients and allows evaluations on varying numbers of test inputs.

Experimental Environment. To resemble real-world FL, we deploy our experiments in Flower FL [39], running on an enterprise-level cluster of six NVIDIA DGX A100 [52] nodes. Each node is equipped with 2048 GB of memory, at least 128 cores, and an A100 GPU with 80 GB of memory.

We vary training rounds between 15 to 80 with clients ranging from 100 to 1000, thus testing TraceFL on more configurations than any related work [51], [53]. Ten randomly selected clients participate in each round, reflecting a real-world scenario where not all the clients participate in the given round [54]. We evaluate TraceFL with FedAvg [1].

Localization Accuracy. To measure the performance of TraceFL, we evaluate the accuracy of TraceFL in finding the responsible clients. For brevity, we refer to this as localization accuracy, which is defined as follows: Given the z number of test inputs to the global model (W^{t+1}_{global}), if TraceFL accurately locates m times the clients responsible for the z predictions, then the localization accuracy is $\frac{m*100}{z}$.

A. TraceFL's Localization Accuracy in Correct Predictions

Identifying the clients most responsible for correct prediction is a key debugging objective that helps encourage future participation of those clients to improve the overall FL accuracy. Note that TraceFL directly does not improve the FL model accuracy. Instead, it reasons about the behavior of the FL global model which an FL developer can use to improve the FL model accuracy (*e.g.,* selecting clients which are contributing more in the FL global model predictions).

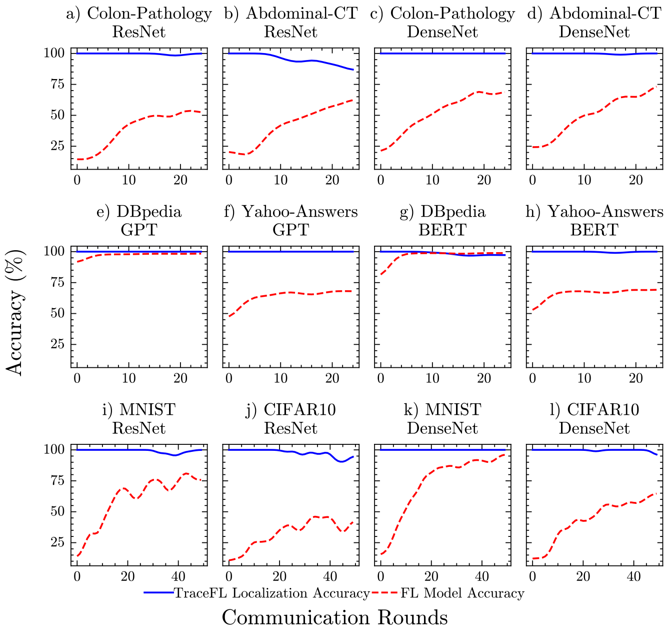

Fig. 2: TraceFL performance on multiple datasets and models both on text and image classification tasks.

TraceFL's **neuron provenance** traces predictions back to clients trained on those labels, ranking clients by their contribution. TraceFL returns a ranked list of clients in descending order of responsibility towards a prediction, where the client with the highest score is likely to be most responsible.

We evaluate TraceFL's localization accuracy on two real-world medical imaging datasets, two standardized image datasets, and two NLP classification datasets using ResNet, DenseNet, BERT, and GPT models resulting in over 12 FL configurations spanning a total 400 FL rounds and 4000 models. We verify if the most responsible client returned by TraceFL contains the data with the label that was correctly predicted by the global model. We measure the accuracy on at least 10 test inputs in each round. Figure 2 shows TraceFL's performance in localizing responsible clients. The X-axis represents training rounds, while the Y-axis shows the FL global model's classification accuracy and TraceFL's localization accuracy.

We include the FL global model's accuracy to demonstrate the training progression. Higher global model accuracy improves neuron provenance confidence, aiding TraceFL's effectiveness. Global model accuracy helps calibrate the provenance results because lower model accuracy leads to low confidence in prediction, which transitively reduces the confidence of neuron provenance, causing additional challenges for TraceFL. As training progresses and more clients with unique labels participate, the global model's accuracy improves.

Our results indicate that TraceFL consistently localizes responsible clients regardless of the global model's performance, neural network architecture, number of training rounds, or dataset. It accurately identifies contributions even from clients participating for the first time. Across different FL

Domain	Dataset	Dirichlet Distribution (α)	FedDebug Accuracy (%)	TraceFL Accuracy (%)
Image	Abdominal-CT	0.3	0.00	100
		0.7	21.5	100
		1.0	44.4	100
	Colon-Pathology	0.3	0.00	100
		0.7	54.7	100
		1.0	68.7	100
	CIFAR10	0.3	20.0	100
		0.7	11.3	100
		1.0	22.0	100
	MNIST	0.3	14.0	100
		0.7	86.0	100
		1.0	36.0	100
Text	DBpedia	0.3	NA	96.7
		0.7	NA	94.0
		1.0	NA	97.3
	Yahoo-Answers	0.3	NA	100
		0.7	NA	100
		1.0	NA	100

TABLE I: Comparison of TraceFL with FedDebug on localizing clients responsible for misprediction. FedDebug is compatible with image classification only and is effective under specific data distribution (*i.e.,* $\alpha = 1$).

settings, TraceFL's average localization accuracy on image classification tasks is 98.96%, and in text classification tasks, it is 99.59%, demonstrating its broader effectiveness and applicability to domains other than image classification.

Takeaway. On average, TraceFL achieves localization accuracy of 99.12% across all FL experiments settings.

B. *TraceFL's Localization Accuracy in Mispredictions*

FL's global model can exhibit unwanted behavior (*e.g.,* mispredictions) due to intentional or unintentional faults in the training data of clients. Mislabelling in training data may occur due to faulty sensors, human error in labeling data, or, in some cases, adversarial attacks [1]–[5]. Finding a client responsible for such behavior is a crucial debugging goal that helps FL developers exclude such clients from participating in future rounds to improve the global model's quality.

To evaluate TraceFL's localization accuracy on mispredicted labels by a global model, we design the following experiments with ten clients. Similarly to prior work on fault localization in FL, FedDebug [33], we select one client in an FL round and flip a specific label in its training data to make it faulty. The inclusion of such clients influences the global model to make mispredictions. For instance, in the medical dataset, we flip the label 'Cancer-associated Stroma' to 'Adipose' in the Colon Pathology dataset to reflect a faulty hospital containing incorrect label data that may occur due to misdiagnosis.

Table I shows the results. TraceFL outperforms FedDebug significantly and can operate in cross-domain tasks of image and text classification without any change in its approach. This is expected since FedDebug, by construction, applies to a different problem setting, i.e., debugging the model instead of the prediction, and it primarily targets a specific set of Non-IID data distributions. Even on image classification tasks, TraceFL outperforms FedDebug in terms of localization accuracy. For instance, in Abdominal-CT with $\alpha = 1$, FedDebug's average accuracy is 44.4% while TraceFL's accuracy is 100%.

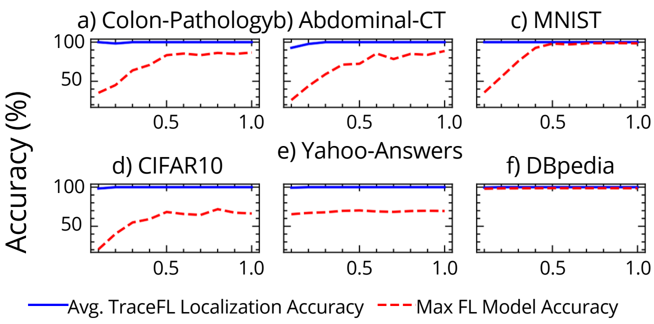

Fig. 3: TraceFL performance on different data distributions. The X-axis represents the values of Dirichlet alpha.

Takeaway. TraceFL achieves 99.3% average localization accuracy across 18 FL settings, whereas FedDebug's average localization accuracy is only 32% on image classification.

C. *TraceFL's Robustness*

Varying the client's data distribution and applying differential privacy (DP) techniques in FL pose additional hurdles to FL in achieving high model accuracy, which, in turn, may pose challenges to TraceFL in keeping its high localization accuracy. Therefore, to add rigor to our experiments, we evaluate the impact of these two additional FL settings on TraceFL localization accuracy. In this section, we only include results from the most challenging experiment setting due to space constraints.

*1) **Varying Data Distribution**:* Different distributions of data among clients can impact the FL training process. For instance, in a highly challenging data distribution ($\alpha = 0.1$), FL training suffers from low global model accuracy. This is a known phenomenon in FL [37], where the FL fusion algorithm struggles to aggregate clients' models trained on severely heterogeneous training data. To mitigate bias towards a specific Dirichlet data distribution, we evaluate TraceFL on varying the value of α from 0.1 to 1.0, showing the impact of different data distributions on TraceFL's localization accuracy.

Figure 3 shows the results of this experiment on all six datasets. The X-axis represents the value of α in the Dirichlet distribution, while the Y-axis represents the accuracy. For a value of α, we report the maximum accuracy achieved by the global model across all the rounds as FL model accuracy and the average accuracy of TraceFL across all the rounds as localization accuracy of TraceFL.

As expected, the FL training accuracy decreases as the value of α decreases. This is because the clients have varying data both in terms of quantity and labels. For instance, when $\alpha = 0.1$ in Figure 3-(a), the maximum FL global model accuracy observed across all rounds is 35.25% and when $\alpha = 0.5$ the maximum accuracy is 83.4%. Since GPT is an advanced neural network architecture that learns better in comparison to DenseNet, the FL training accuracy is higher in GPT on lower α values as well. Overall, TraceFL localization

DP Noise	DP Sensitivity	FL Model Accuracy %	TraceFL Avg. Accuracy %
0.003	15	97.36	100
0.006	10	97.90	100
0.012	15	88.81	100

TABLE II: Results of TraceFL with DP in FL.

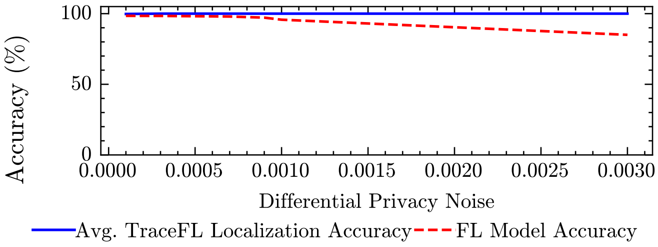

Fig. 4: Impact of DP noise on FL training accuracy.

accuracy is 99.76%, on average, across all values of α. The line plots show no significant change in TraceFL localization accuracy, demonstrating TraceFL's robustness in challenging real-world data distributions.

*2) **Differential Privacy-Enabled FL**:* Differential privacy (DP) is a privacy-preserving mechanism that ensures that the output of a model does not reveal any information about the individual data points. DP in FL [55] adds noise to the weights of a model to protect against an adversary stealing or recovering the individual training data points. However, a delicate balance is needed in DP between the noise to be added and model accuracy, as adding too much noise severely decreases the model's accuracy.

We evaluate TraceFL's robustness when DP is enabled in FL, using standard DP settings in FL that provide optimal privacy and model accuracy, as mentioned in prior work [55]. Table II presents the results of this experiment, and Figure 4 shows the impact of noise on the FL training accuracy. As expected, the FL model's accuracy decreases when the DP noise increases and vice versa.

However, TraceFL maintains its performance in DP-enabled FL. As DP adds noise to the model weights, the global model's output is still based on its neurons' activations on the given input. Thus, TraceFL's working principle remains intact, and it successfully traces back to the source of the prediction based on the global model's **neuron provenance**. We want to emphasize that *TraceFL does not recover the individual clients' data points. It only identifies the responsible clients in ranked order.* Overall, we find that TraceFL is robust against the use of differential privacy in FL where it achieves an average localization accuracy of 99% in GPT and DBpedia dataset (Figure 4 and Table II).

Takeaway. TraceFL is robust to challenging real-world data distributions and the use of differential privacy, achieving approximately 99% localization accuracy.

D. TraceFL's Scalability

We assess the scalability of TraceFL across three different dimensions: (1) by increase the total clients, (2) by increasing the client participation, and (2) by increasing the number of rounds. First, we vary the number of clients from 200 to 1000 and measure if TraceFL can still accurately localize the responsible client. We use the state-of-the-art neural network GPT and the DBpedia dataset. Table III presents the results of the scalability experiment. We observe that TraceFL's performance remains consistent, with an average localization accuracy of 99% across 200 to 1000 clients over a total of 75 FL training rounds. This experiment significantly exceeds the scale of experiments performed by prior work [33].

Total Clients	FL Model Accuracy %	TraceFL Avg. Accuracy %
200	98.49	99.76
400	98.29	99.76
600	98.39	100
800	98.10	100
1000	98.05	99.52

TABLE III: Scalability results of TraceFL with different number of clients with GPT.

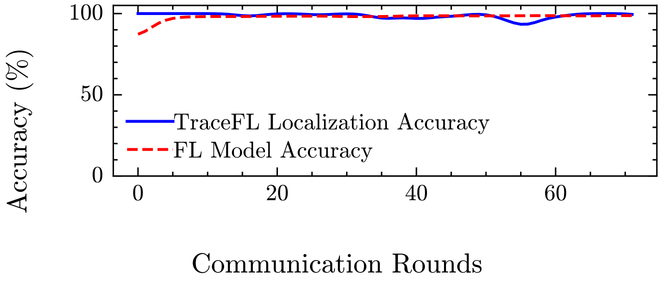

Fig. 5: TraceFL's scalability when # of rounds increase

When we vary the number of participating clients per round from 20 to 50, TraceFL's performance remains stable, achieving 100% localization accuracy across 60 FL training rounds. Prior work has shown that even at an enterprise scale, only a few clients participate in a single FL round [34], [36]. Furthermore, we evaluate the scalability of TraceFL over up to 80 rounds with 400 clients in total. Figure 5 demonstrates that TraceFL maintains consistent performance with an average localization accuracy of 98%. These results indicate that TraceFL is scalable and can handle numerous clients and rounds without compromising its performance.

Takeaway. Overall, TraceFL is capable of handling the provenance of millions of neurons in the neural network to accurately identify the most responsible client. In FL settings of up to 80 FL training rounds and 1000 clients using large models such as GPT and BERT—and 6 different datasets, TraceFL achieves an average localization accuracy of 99.20%.

E. TraceFL's Localization Time

We evaluate the runtime performance of TraceFL by measuring the time TraceFL takes to accurately localize the responsible clients in FL. As mentioned before, there is no existing method that localizes the responsible clients for both correct and incorrect predictions. The closed related work to TraceFL is FedDebug, which only localizes faulty clients. Thus, we compare TraceFL's localization time with FedDebug's faulty localization time.

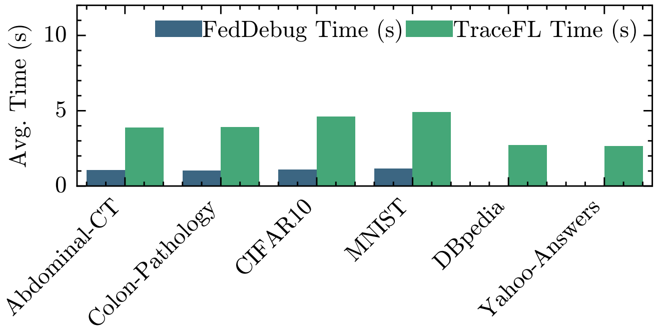

Fig. 6: Client localization of TraceFL vs. FedDebug.

Figure 6 presents the localization times per dataset for both TraceFL and FedDebug, averaged across faulty client localization settings. Note that FedDebug is not compatible with the text classification models; therefore, its localization times for the two text datasets are not available. TraceFL takes, on average, 3.7 seconds to localize the responsible client, whereas FedDebug's faulty client's localization time is 1.1 seconds on average. This is expected as TraceFL requires computing gradients of neuron outputs, whereas FedDebug compares raw neuron activations. While TraceFL's localization time is higher than FedDebug, it is almost negligible compared to the FL's per round training time (in minutes if not hours [33]).

Takeaway. TraceFL compensates for the marginally slower localization time with much broader debugging support for model architecture, text data domains, and general-purpose reasoning in FL.

F. Threat to Validity and Limitations

There are two primary threats to the validity of the results. First, in our experiments, we select a random subset of clients to participate in every round. A different sequence of randomly selected participating clients may alter the TraceFL's accuracy. We mitigate this threat by performing responsible client localization on every round and then reporting the average localization accuracy across rounds. Second, the same Dirichlet distribution (α) may provide a different distribution of the training data across clients. Even when the value α is the same, the localization accuracy of TraceFL may vary slightly. We mitigate this threat by averaging the localization accuracy across rounds and also measuring the localization accuracy on different datasets and models. TraceFL is designed for classification tasks in FL and may not be directly applicable to non-classification tasks such as text generation [56] and embeddings generation [57].

VI. Related Work

Debugging and Interpretability in Machine Learning. As the complexity of neural network models continues to increase, the need for interpretability techniques becomes more crucial and important. Interpretability techniques are used to understand the inner workings of a neural network. These techniques try to explain the decisions made by the model, and how the model makes these decisions. This is important for many reasons, including the ability to explain which input features are important to a model's output, to understand the model's behavior, and to identify potential biases and errors in a trained model. Several approaches, such as Integrated Gradients [58], Gradient SHAP [59], DeepLIFT [60], Saliency [61], Guided GradCAM [62], Occlusion (also called sliding window method) [63], and LIME [64], exist which evaluate the contribution of each input feature to model's output. For instance, Integrated Gradients [58] evaluates the contribution of each input feature by calculating the integral of gradients *w.r.t.* input. This is done along the path from a selected baseline to the given input. Occlusion involves replacing each contiguous rectangular region with a predetermined baseline or reference point and measuring the difference in the model's output. This approach is based on perturbations and provides a way to evaluate the importance of input features by measuring the change in the model's output.

Existing debugging techniques [65]–[69] are designed to identify issues and enhance the performance of a single neural network in centralized ML. These methods typically require access to training data, which is prohibited in FL. For example, NPC [68] constructs a Decision Graph using training data. Furthermore, these approaches have not been evaluated on modern neural network architectures such as Transformers.

Almost all existing debugging and interpretability approaches are inapplicable in FL, as by design, they solve an orthogonal problem — identifying the important feature in the input responsible for a prediction instead of clients. This distinction is critical because the training data or the training process is completely inaccessible in FL. Existing approaches require access to the client's data. Furthermore, they are only designed for a single neural network, but the FL global model is a mixture of clients' models participating in the given round. Operating these techniques on FL would require us to first identify a suspicious client's mode–a problem that TraceFL solves. Even if such techniques are applied to a client's model, the resulting feedback is not immediately actionable and constructive. TraceFL is designed to address the limitations of the existing debugging approaches and added challenges of FL, such as distributed training, inaccessibility to clients, and the mixture of models.

FedDebug [33] introduces differential testing in FL to identify faulty clients by capturing each client's activations for a given input and localizing the client(s) whose behavior deviates from others. Building on FedDebug, a backdoor detection technique in FL is presented in [10]. Additionally, FedGT [11] aims to identify malicious clients in FL; however, it is limited to scaling up to 15-30 clients and has not been tested on advanced architectures like GPT.

Despite their contributions, these existing methods target a narrower problem under a specialized setting. FedDebug is limited to image classification tasks using CNNs, restricting its applicability to Transformer architectures. Additionally, it is designed primarily for faulty clients and IID distributions [37], as demonstrated in Table I. In contrast, TraceFL targets a broader debugging problem using a domain-agnostic and highly accurate client localization mechanism applicable to diverse neural network architectures, data types, and distributions through its novel fine-grained **neuron provenance**.

There has been recent work on ensuring accountability in FL systems. A vast majority of solutions leverage the blockchain to ensure accountability [70]–[74]. Some of these works (BlockFLow [73], BlockFLA [74]) design an FL system that uses the Ethereum blockchain to provide accountability and monetary rewards for good client behavior. However, all these systems require utilizing the blockchain and entail significant modifications to the existing FL system, presenting a barrier to adoption. TraceFL, in contrast, can work with any existing system without modifications.

Provenance Approaches in ML. Provenance has been extensively studied for both ML and dataflow programs [18]–[20], [75], [76]. They address various issues such as reproducibility [17], [77]–[80], provide debugging and testing granularities [76], explainability [20], and mitigating data poisoning attacks [14]–[16]. In the context of machine learning, provenance tracks the history of datasets, models, and experiments. This information is used to select the interpretability of neural network predictions and reproducibility. Provenance-based approaches are important to create ML systems that generate reproducible results [17], [77]–[80]. For instance, Ursprung [17] captures provenance and lineage by integrating with the execution environment and records information from both system and application sources of an ML pipeline. Ursprung does not require changes to the code and only adds a small overhead of up to 4%.

VII. Conclusion

We introduce the concept of neuron provenance and developed a debugging and interpretability tool, TraceFL, for FL. TraceFL accurately identifies the primary contributors to a global model's behavior. Our evaluations show that TraceFL achieves an impressive average localization accuracy of 99%. Furthermore, TraceFL also outperforms the existing fault localization technique. We provide a reusable functional artifact of TraceFL in the Flower framework to have an immediate practical impact in real-world FL deployment, addressing the open challenges of debugging and interpretability in FL.

Acknowledgement

We thank anonymous reviewers for providing valuable and constructive feedback to help improve the quality of this work. This work was supported in part by Amazon - Virginia Tech Initiative in Efficient and Robust Machine Learning, 4-VA, and the National Science Foundation award 2106420. We also thank the Advanced Research Computing Center at Virginia Tech and the Flower FL framework for their support in building and evaluating this work.

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. C. Jiang, B. Kantarci, S. Oktug, and T. Soyata, "Federated learning in smart city sensing: Challenges and opportunities," *Sensors*, vol. 20, no. 21, p. 6230, 2020.

[3] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.*, "The future of digital health with federated learning," *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[4] G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated learning for open banking," in *Federated learning*. Springer, 2020, pp. 240–254.

[5] Z. Zheng, Y. Zhou, Y. Sun, Z. Wang, B. Liu, and K. Li, "Applications of federated learning in smart cities: recent advances, taxonomy, and open challenges," *Connection Science*, pp. 1–28, 2021.

[6] Y. J. Cho, D. Jhunjhunwala, T. Li, V. Smith, and G. Joshi, "To federate or not to federate: incentivizing client participation in federated learning," in *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*, 2022.

[7] Y. Zhan, J. Zhang, Z. Hong, L. Wu, P. Li, and S. Guo, "A survey of incentive mechanism design for federated learning," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 1035–1044, 2021.

[8] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[9] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*. PMLR, 2019, pp. 634–643.

[10] W. Gill, A. Anwar, and M. A. Gulzar, "FedDefender: Backdoor Attack Defense in Federated Learning," in *Proceedings of the 1st International Workshop on Dependability and Trustworthiness of Safety-Critical Systems with Machine Learned Components*, ser. SE4SafeML 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 6–9. [Online]. Available: https://doi.org/10.1145/3617574.3617858

[11] M. Xhemrishi, J. Östman, A. Wachter-Zeh *et al.*, "Fedgt: Identification of malicious clients in federated learning with secure aggregation," *arXiv preprint arXiv:2305.05506*, 2023.

[12] H. Ali, D. Chen, M. Harrington, N. Salazar, M. Al Ameedi, A. F. Khan, A. R. Butt, and J.-H. Cho, "A survey on attacks and their countermeasures in deep learning: Applications in deep neural networks, federated, transfer, and deep reinforcement learning," *IEEE Access*, vol. 11, pp. 120 095–120 130, 2023.

[13] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International conference on machine learning*. PMLR, 2017, pp. 1885–1894.

[14] J. W. Stokes, P. England, and K. Kane, "Preventing machine learning poisoning attacks using authentication and provenance," in *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 2021, pp. 181–188.

[15] N. Baracaldo, B. Chen, H. Ludwig, A. Safavi, and R. Zhang, "Detecting Poisoning Attacks on Machine Learning in IoT Environments," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, 2018, pp. 57–64.

[16] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: A data provenance based approach," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017, pp. 103–110.

[17] L. Rupprecht, J. C. Davis, C. Arnold, Y. Gur, and D. Bhagwat, "Improving reproducibility of data science pipelines through transparent provenance capture," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3354–3368, 2020.

[18] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, "Putting Lipstick on Pig: Enabling Database-Style Workflow Provenance," *Proc. VLDB Endow.*, vol. 5, no. 4, p. 346–357, dec 2011.

[19] H. Park, R. Ikeda, and J. Widom, "RAMP: A System for Capturing and Tracing Provenance in MapReduce Workflows," *Proc. VLDB Endow.*, vol. 4, no. 12, p. 1351–1354, jun 2020.

[20] S. Akoush, R. Sohan, and A. Hopper, "HadoopProv: Towards Provenance as a First Class Citizen in MapReduce." in *TaPP*, 2013.

[21] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate Quantization for Generative Pre-trained Transformers," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[22] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Confer-*

ence on International Conference on Machine Learning, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[23] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[24] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, vol. 3, no. 3, p. e10, 2018.

[25] S. Yu, P. Nguyen, W. Abebe, W. Qian, A. Anwar, and A. Jannesari, "Spatl: Salient parameter aggregation and transfer learning for heterogeneous federated learning," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–14.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[29] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[30] J. N. Kather, J. Krisam, P. Charoentong, T. Luedde, E. Herpel, C.-A. Weis, T. Gaiser, A. Marx, N. A. Valous, D. Ferber *et al.*, "Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study," *PLoS medicine*, vol. 16, no. 1, p. e1002730, 2019.

[31] X. Xu, F. Zhou, B. Liu, D. Fu, and X. Bai, "Efficient multiple organ localization in ct image using 3d region proposal network," *IEEE transactions on medical imaging*, vol. 38, no. 8, pp. 1885–1898, 2019.

[32] P. Bilic, P. Christ, H. B. Li, E. Vorontsov, A. Ben-Cohen, G. Kaissis, A. Szeskin, C. Jacobs, G. E. H. Mamani, G. Chartrand *et al.*, "The liver tumor segmentation benchmark (lits)," *Medical Image Analysis*, vol. 84, p. 102680, 2023.

[33] W. Gill, A. Anwar, and M. A. Gulzar, "FedDebug: Systematic Debugging for Federated Learning Applications," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 512–523. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00053

[34] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NeurIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[35] D. Avdiukhin and S. P. Kasiviswanathan, "Federated Learning under Arbitrary Communication Patterns," in *International Conference on Machine Learning*, 2021.

[36] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, B. McMahan *et al.*, "Towards federated learning at scale: System design," *Proceedings of machine learning and systems*, vol. 1, pp. 374–388, 2019.

[37] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated Learning on Non-IID Data Silos: An Experimental Study," in *IEEE International Conference on Data Engineering*, 2022.

[38] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *Advances in neural information processing systems*, vol. 33, pp. 7611–7623, 2020.

[39] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[40] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. Nitin Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and Open Problems in Federated Learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[41] Y. Liu, Y. Kang, L. Li, X. Zhang, Y. Cheng, T. Chen, M. Hong, and Q. Yang, "A communication efficient vertical federated learning framework," *Scanning Electron Microsc Meet at*, 2019.

[42] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 394–21 405, 2020.

[43] A. F. Khan, A. A. Khan, A. M. Abdelmoniem, S. Fountain, A. R. Butt, and A. Anwar, "FLOAT: Federated Learning Optimizations with Automated Tuning," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 200–218.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[45] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[46] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[47] J. Yang, R. Shi, and B. Ni, "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis," in *IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, 2021, pp. 191–195.

[48] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[49] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, 2015.

[50] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated Learning with Matched Averaging," in *International Conference on Learning Representations*, 2020.

[51] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2351–2363, 2020.

[52] "Nvidia dgx a100 — the universal system for ai infrastructure," https://images.nvidia.com/aem-dam/Solutions/Data-Center/nvidia-dgx-a100-datasheet.pdf.

[53] J. Liu, J. Lou, L. Xiong, J. Liu, and X. Meng, "Projected federated averaging with heterogeneous differential privacy," *Proceedings of the VLDB Endowment*, vol. 15, no. 4, pp. 828–840, 2021.

[54] T. Li, M. Sanjabi, A. Beirami, and V. Smith, "Fair Resource Allocation in Federated Learning," in *International Conference on Learning Representations*, 2020.

[55] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning Differentially Private Recurrent Language Models," in *International Conference on Learning Representations*, 2018.

[56] L. Sani, A. Iacob, Z. Cao, B. Marino, Y. Gao, T. Paulik, W. Zhao, W. F. Shen, P. Aleksandrov, X. Qiu *et al.*, "The Future of Large Language Model Pre-training is Federated," *arXiv preprint arXiv:2405.10853*, 2024.

[57] W. Gill, M. Elidrisi, P. Kalapatapu, A. Ahmed, A. Anwar, and M. A. Gulzar, "MeanCache: User-Centric Semantic Cache for Large Language Model Based Web Services," 2024. [Online]. Available: https://arxiv.org/abs/2403.02694

[58] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[59] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[60] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*. PMLR, 2017, pp. 3145–3153.

[61] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[62] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[63] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 2014, pp. 818–833.

[64] M. T. Ribeiro, S. Singh, and C. Guestrin, “” why should i trust you?” explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[65] B. Sun, J. Sun, L. H. Pham, and J. Shi, “Causality-based neural network repair,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 338–349.

[66] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. S. Păsăreanu, “Nnrepair: Constraint-based repair of neural network classifiers,” in *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*. Springer, 2021, pp. 3–25.

[67] S. Gerasimou, H. F. Eniser, A. Sen, and A. Cakan, “Importance-driven deep learning system testing,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 702–713.

[68] X. Xie, T. Li, J. Wang, L. Ma, Q. Guo, F. Juefei-Xu, and Y. Liu, “NPC: Neuron Path Coverage via Characterizing Decision Logic of Deep Neural Networks,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, Apr. 2022.

[69] C. Tao, Y. Tao, H. Guo, Z. Huang, and X. Sun, “Dlregion: coverage-guided fuzz testing of deep neural networks with region-based neuron selection strategies,” *Information and Software Technology*, vol. 162, p. 107266, 2023.

[70] X. Bao, C. Su, Y. Xiong, W. Huang, and Y. Hu, “FLChain: A Blockchain for Auditable Federated Learning with Trust and Incentive,” in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, 2019, pp. 151–159.

[71] W. Zhang, Q. Lu, Q. Yu, Z. Li, Y. Liu, S. K. Lo, S. Chen, X. Xu, and L. Zhu, “Blockchain-based federated learning for device failure detection in industrial iot,” *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5926–5937, 2020.

[72] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, “Reliable federated learning for mobile networks,” *IEEE Wireless Communications*, vol. 27, no. 2, pp. 72–80, 2020.

[73] V. Mugunthan, R. Rahman, and L. Kagal, “Blockflow: An accountable and privacy-preserving solution for federated learning,” *arXiv preprint arXiv:2007.03856*, 2020.

[74] H. B. Desai, M. S. Ozdayi, and M. Kantarcioglu, “Blockfla: Accountable federated learning via hybrid blockchain architecture,” in *Proceedings of the eleventh ACM conference on data and application security and privacy*, 2021, pp. 101–112.

[75] D. Logothetis, S. De, and K. Yocum, “Scalable Lineage Capture for Debugging DISC Analytics,” in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC ’13. New York, NY, USA: Association for Computing Machinery, 2013.

[76] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie, “Titian: Data provenance support in spark,” in *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, vol. 9, no. 3. NIH Public Access, 2015, p. 216.

[77] M. Paganini and J. Z. Forde, “dagger: A python framework for reproducible machine learning experiment orchestration,” *CoRR*, vol. abs/2006.07484, 2020.

[78] R. Souza, L. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandao, D. Civitarese, E. Brazil, M. Moreno, P. Valduriez *et al.*, “Provenance data in the machine learning lifecycle in computational science and engineering,” in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2019, pp. 1–10.

[79] S. Samuel, F. Löffler, and B. König-Ries, “Machine learning pipelines: Provenance, reproducibility and fair data principles,” in *Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020+ IPAW 2021, Virtual Event, July 19–22, 2021, Proceedings 8*. Springer, 2021, pp. 226–230.

[80] D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, “Production machine learning pipelines: Empirical analysis and optimization opportunities,” in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2639–2652.