FLAMES2Graph: An Interpretable Federated Multivariate Time Series Classification Framework

Raneen Younis
L3S Research Center
Hannover, Germany
younis@l3s.de

Zahra Ahmadi*
L3S Research Center
Hannover, Germany
ahmadi@l3s.de

Abdul Hakmeh
Universität Hildesheim
Hildesheim, Germany
hakmeh@bwl.uni-hildesheim.de

Marco Fisichella
L3S Research Center
Hannover, Germany
mfisichella@l3s.de

ABSTRACT

Increasing privacy concerns have led to decentralized and federated machine learning techniques that allow individual clients to consult and train models collaboratively without sharing private information. Some of these applications, such as medical and healthcare, require the final decisions to be interpretable. One common form of data in these applications is multivariate time series, where deep neural networks, especially convolutional neural networks based approaches, have established excellent performance in their classification tasks. However, promising results and performance of deep learning models are a black box, and their decisions cannot always be guaranteed and trusted. While several approaches address the interpretability of deep learning models for multivariate time series data in a centralized environment, less effort has been made in a federated setting. In this work, we introduce FLAMES2Graph, a new horizontal federated learning framework designed to interpret the deep learning decisions of each client. FLAMES2Graph extracts and visualizes those input subsequences that are highly activated by a convolutional neural network. Besides, an evolution graph is created to capture the temporal dependencies between the extracted distinct subsequences. The federated learning clients only share this temporal evolution graph with the centralized server instead of trained model weights to create a global evolution graph. Our extensive experiments on various datasets from well-known multivariate benchmarks indicate that the FLAMES2Graph framework significantly outperforms other state-of-the-art federated methods while keeping privacy and augmenting network decision interpretation.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Spatial-temporal systems**; • **Computing methodologies** → *Supervised learning by classification.*

*Corresponding author

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599354

KEYWORDS

Federated learning, Interpretability, Neural networks, Time series

ACM Reference Format:
Raneen Younis, Zahra Ahmadi, Abdul Hakmeh, and Marco Fisichella. 2023. FLAMES2Graph: An Interpretable Federated Multivariate Time Series Classification Framework. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3580305.3599354

1 INTRODUCTION

Today, mobile devices and IoT devices are the primary computing platforms and generate a massive amount of valuable data while providing hidden insights into the human world. Data evaluation and analysis can be greatly enhanced by artificial intelligence. However, most machine learning models are trained in a centralized setting and on centrally stored data to receive a satisfactory model. This conventional learning approach, in which the clients share their data to a central server, has several drawbacks, such as privacy issues in sensitive applications and limited bandwidth on mobile devices. Moreover, training a single machine learning model on potentially enormous-sized training data on a centralized server or cloud can become absolutely expensive. A practical method of training machine learning models on edge can eliminate the need to train them on the cloud. Federated Learning (FL) [16] was introduced to address these issues by only sharing clients' trained model weights with a server in each round of communication. In its original form, the server aggregates the weights using the Federated Averaging Algorithm (FedAvg) algorithm and returns the global model weights to the clients to further use in the next training round. However, training under the FL setting still has several challenges that require further study, such as dealing with non-independent identically distributed (non-IID) data among clients [27, 56], reducing communication costs [26, 29], handling adversarial attacks [31, 54], and protecting user data privacy [11, 24]. Furthermore, the lack of transparency in the federated learning process diminishes trust and limits its adoption by more clients. It is, therefore, imperative that federated learning approaches be interpretable.

Many recent applications with distributed sensitive data, such as economics forecasting, activity recognition, and healthcare, deal with data collected from sensors [9, 45, 48, 51]. This form of data

with simultaneous multiple values is called multivariate time series (MTS). Various methods have been developed to classify multivariate time series data in a centralized scheme: Distance-based algorithms, such as 1-nearest neighbor and dynamic time warping, have shown reliable performance in case of the correct choice of distance measure for a specific application [34, 44]. On the other hand, feature-based methods heavily rely on hand-crafted features [8, 53, 55]. The use of deep learning methods, especially Convolutional Neural Networks (CNN), has shown promising results for time series data [14] and removes the heavy feature engineering requirement. However, despite their outstanding performance, deep learning models' inner functioning and decisions are ambiguous to users and even their designers. Lately, few studies have explored the interpretations of deep networks decisions on time series data in a centralized fashion [6, 57].

In this paper, we focus on two major challenges in the federated learning of multivariate time series: The communication efficiency between clients and the server, where the clients usually connect to the FL server over slow connections [18]. Another less investigated challenge is designing interpretable strategies for deep neural networks in a federated setup. We design a new graph-based approach called Federated LeArning Multivariate timE Series to Graph (FLAMES2Graph) to address these challenges. To this end, our framework relies on convolutional neural networks trained on multivariate time series data to extract representative patterns that activate neurons in CNNs. We call these representatives Multivariate Highly Activated Period (MHAP) and group their similar patterns into more general representatives via a shape-based clustering method. Instead of merely considering their occurrence, we create a graph where its nodes represent the extracted MHAPs cluster medians, and the edges are their occurrence order in an MTS sample. After every few CNN training epochs (that happens inside one federated communication round), the client shares its current local graph with the server representing the influential subsequences found in the local MTS samples. The server then merges the clients' local graphs into a global graph and returns it back to the clients for use in the following communication round. Figure 1 shows a general overview of our framework. The main contributions of the FLAMES2Graph framework are as follows:

- FLAMES2Graph is the first framework that offers a federated interpretable deep learning solution for the multivariate time series classification problem. It extracts and visualizes the representative patterns of the input data and constructs an MHAP evolution graph that captures the temporal relationship between the extracted representative patterns.
- Instead of sharing the learning weights, the clients only share their generated local graph with the central server, which aggregates those local graphs into a global one to improve the generalization capacity of local clients.
- In contrast to post-hoc interpretable approaches in a centralized setting, such as Multi-VISION [57], which create insights into the model only after the end of the training phase, FLAMES2Graph iteratively extracts the representative patterns during the training. Then, it benefits from them in the model update and prediction.
- We verify the FLAMES2Graph framework's performance and interpretability through extensive experiments on five selected datasets from Baydogan's archive [3], along with Human Activity Recognition (HAR) [2] and PAMAP2 Physical Activity Monitoring (PAM) datasets [39].

2 RELATED WORK

In this section, we review the related work: First, we discuss the existing studies related to interpretable time series deep learning methods. Then, we give an overview of the state-of-the-art federated learning techniques.

2.1 Interpretable Models for time series data

Various approaches have been developed to interpret deep learning models, especially for images and text data. Current research mainly focuses on post-model interpretability strategies, such as Local Interpretable Model-agnostic Explanations (LIME) [40], Layer Relevance Propagation (LRP) [30], Class Activation Map (CAM) [60]. CAM was the first post-hoc method that provided interpretation of the predictions by visualizing the region of raw input data that activates the network neurons for a given label [52]. Clustered Pattern of Highly Activated Period (CPHAP) [6] is another post-hoc method that extracts the representative input patterns activated by a trained CNN neurons. However, these methods were introduced only for univariate time series data. The multi-VISION framework [57] introduces a method for interpreting multivariate time series data by extracting the input data that highly activate neurons in a CNN. We use a similar approach to extract the representative input data from multivariate time series.

Apart from post-model strategies, Dual-stage Attention-Based Recurrent Neural Network method (DA-RNN) offers an in-model strategy via an input-attention mechanism, captures long-term temporal dependencies, and selects relevant series for model prediction [38]. The Time2Graph framework [5] is another in-model approach with an intrinsically interpretable strategy that extracts dynamic shapelets from univariate input data through a handcrafted feature engineering procedure and builds an evolution graph based on the temporal sequence of the extracted shapelets. Gated Transformer Networks (GTN) were introduced for multivariate time series by adding gating to transformer networks to explore the GTN attention map [25]. An interpretable CNN-based method was developed by [12] for multivariate time series, which identifies both signals and time intervals that determine the classifier's output by adding an attention mechanism to the extracted features. Our approach differs from previous studies since we extract the relevant time series subsequences for multivariate time series and construct an evolution graph of the extracted sequences over time. The benefit of this approach is that no handcraft feature engineering or domain knowledge is required. In addition, our approach can manage horizontally distributed data in a federated learning setup instead of in a central system.

2.2 Federated Learning

Federated Learning is one of the most widely used decentralized machine learning solutions today to train data on clients' edge devices and share only the trained weight [1, 10, 35]. FedAvg [28] is the most commonly used optimization algorithm in federated settings. A FedAvg model consists of a global model that is first downloaded

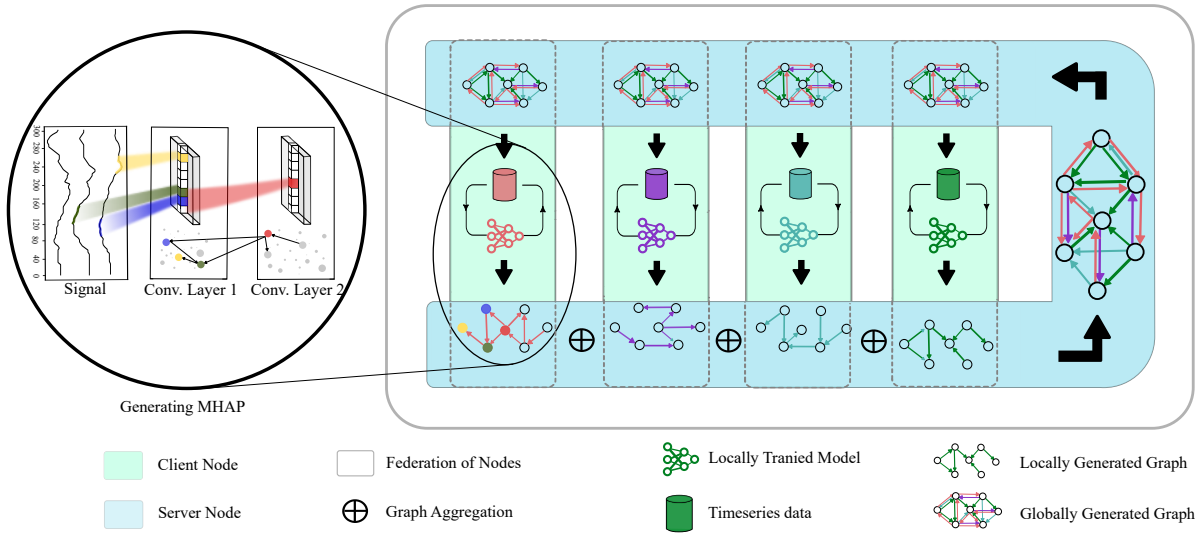


Figure 1: The general framework of FLAMES2Graph, an interpretable federated convolutional neural network for multivariate time series data. The figure illustrates four clients; each trains a CNN network and extracts the highly activated periods (MHAP) from their data to create an MHAP evolution graph in round n. The graphs are then shared with the server to be aggregated into one global graph. Finally, the clients receive the aggregated graph and use it in the following training round $n + 1$.

and then trained using stochastic gradient descent (SGD) on the local dataset of each participating client. Clients then share their trained model weights with the server, which uses the FedAvg algorithm to average the clients' model weights. The communication efficiency of FedAvg is one of its main advantages, as training performance can be maintained while reducing the total number of bytes transferred. However, FedAvg has the disadvantage of degrading convergence when applied to non-IID data [20]. Different researches were proposed to overcome the FedAvg problem on non-IID data by adding a regularization term to the local optimization or making the data distribution of the clients IID [15, 21, 50].

In this paper, we present a personalized federated learning approach. Existing personalized federated learning methods can be categorized into transfer learning, meta-learning, personalization layers, and multi-task learning [17, 46]. Tu *et al.* [47] proposed FedDL, a dynamic layer-sharing scheme that merges clients' local models to speed up convergence while maintaining high accuracy. Meta-HAR [19] is a framework for federated representation learning that trains a shared embedding network through a Model-Agnostic Meta-learning framework. These frameworks address the heterogeneous FL environments using representation learning. However, they work under the assumption that raw data from multiple devices can be shared flexibly, which adds additional communication costs and exposes data privacy. FetchSGD [42] is a federated communication-efficient optimization algorithm that compresses gradients in each communication round with count sketches and sent to a central aggregator. The aggregator maintains a momentum sketch and an error accumulation sketch, and the weight update in each round is computed based on the error accumulation sketch. The proposed technique reduces the communication requirements while fulfilling the quality requirements of federated training. Other research has also addressed representation learning in federated learning, such as [7, 22, 33]. However, none of the above methods addresses the interpretability of models in the FL setting.

Despite the community's attention to interpretable models in a centralized environment, the problem in a decentralized setting is still understudied. Methods such as LIME [40] cannot be directly applied to the federated setup. SHAP-values [49, 59] were introduced for tree-based models and vertically partitioned data; however, in horizontally partitioned data, this method is not applicable. Another federated interpretable method shares the generated dimensionally-reduced intermediate representations of each participating client data [13]. In this method, neither the models nor the client data are shared. Recently, the IFedAvg method was proposed to detect and solve interoperability problems in federated learning based on tabular datasets [41]. FedST [23] studies interpretable federated time series classification. However, it is introduced for univariate time series and requires adaptations for multivariate time series. This is challenging because the potential shapelets may come from different variables with different lengths and, therefore, cannot be directly compared. None of these methods concentrate on an interpretable federated setting for multivariate time series, the subject of our current research.

3 FLAMES2GRAPH FRAMEWORK

The overall architecture of our proposed framework is shown in Figure 1. First, each client extracts those subsequences of the multivariate input data that highly activate the neurons in sofar trained CNN (Generate MHAP). Then, each client ("Client Node" in the figure) constructs a graph with nodes representing these time series representative patterns and edges showing their temporal dependency in each sample (Section. 3.1). The federated learning server

("Server Node") then aggregates the clients' local graph into a global graph and shares it with the clients (Section. 3.2). The clients then operate on this global graph in the next round (Section. 3.3).

3.1 FL Client Evolution Graph

With the intuition that decisions of deep networks are influenced by their highly activated neurons, Cho *et al.* [6] and Younis *et al.* [57] proposed approaches to extract input time series periods that highly activate neurons in a trained CNN. We extend their initial idea to the federated setting. Note that in this work, the CNN models were chosen because they showed the highest accuracy for the UEA dataset [14]. Moreover, our proposed framework is designed to work with all convolutional networks that do not contain local pooling layers. Each FL client extracts its respective multivariate highly activated periods (MHAP) in three steps:

- First, a multi-layer 1-d convolutional neural network is trained on the multivariate input data for a few epochs. Signals of the multivariate time series data are fed as a whole to the network in the training process. In parallel, an extended set that contains all possible signal combinations of the input sample is created. Each potential combination is created via zero padding of the excluded dimensions.
- Second, each extended set is fed to the trained model to extract MHAPs. An MHAP is extracted by obtaining the trained CNN's Highly Activated Neurons (HAN) and extracting their corresponding signal receptive field from these neurons. Figure 1 (in the "Get MHAP" part) illustrates an example of MHAP extraction. Suppose a neuron in layer l and channel c gets activated (blue neuron). In that case, the input receptive field of this neuron can be extracted from the time series data with respect to channel c and the layer l kernel size.
- Finally, the MHAPs are clustered to merge similar subsequences, and a cluster median is appointed to each cluster.

After extracting the MHAPs and cluster centers, each client generates an MHAP evolution graph. The generation of the MHAP evolution graph is inspired by the shapelet evolution graph in [5]. However, our MHAP evolution graph differs in nature and construction procedure from the shapelet evolution graph. As the MHAPs are extracted separately from each CNN layer, we deal with more than one graph, each representing a CNN layer. The time-dependent graphs are constructed after obtaining the MHAPs from each network layer. These graphs maintain the occurrence order of the MHAPs. Then, all of these graphs are merged into a single time-aware MHAP evolution graph. In the following, we explain the details.

3.1.1 Generation of MHAP time-aware graph. Figure 1 (in the "Federation of Nodes" part, the first client node in the left) shows an example of the generated MHAP time-aware graph. An MHAP graph in each network layer is built as follows: the first MHAP is extracted from the input data and assigned to the corresponding cluster center. This cluster center represents a node n_b (blue node) in the graph. Then the subsequent MHAP is extracted and assigned to its cluster center n_g (green node), respectively. Now, a directed edge e_{bg} is created between two consecutive nodes, representing the temporal order of the MHAPs. The process is repeated for each multivariate sample in the training set to generate a network layer graph G ultimately.

Algorithm 1 Generate MHAP Evolution Graph

Input: X_{train}: local multivariate time series data
Output: evolution graph G, MHAP cluster medians CM_{list}

1: Model ⟵ Update CNN(X_{train}, $epochs$)
2: **for** x `in` X_{train} **do**
3: $signal$:= InputSet(x)
4: **for** l `in Model_layers` **do**
5: **for** c `in` l **do** ▷ channels in a layer
6: $MHAP_{layer}.add(extract\_MHAP(signal))$
7: $MHAP_{cnn}.add(MHAP_{layer})$
8: $\text{CM}_{list} := K_{shape}(MHAP_{cnn}).Median$ ▷ list of layers' centroids
9: $[n] := CM_{list}.size$ ▷ vector of layer centroids size
10: **for** l in `Model_layers` **do**
11: Initialize a graph g with $n[l]$ nodes
12: **for** x `in` X_{train} **do**
13: **for** all consecutive MHAPs in x **do**
14: Add a directed edge $e_{t,t+1}$
15: $G_{list}.add(g)$
16: $G := merge(G_{list})$
17: **return** G, CM_{list}

Each node in the resulted graph represents an MHAP of the current layer, e.g., in the second layer, an MHAP is generated from k neurons of the first layer. Note that the size of k depends on the kernel size of the layer. After obtaining the respective graphs of each layer, a merging method compatible with convolutional layers' functioning is applied to the graphs' layers to create a single graph for the client. Thus, we connect each node in the graph with a node in the previous layer. For example, a node n_r in the second layer graph (red node in Generating MHAP in Figure 1) represents three neurons in the first CNN layer (green, blue, and white nodes), two of which are among the MHAPs of the first layer, n_b, n_g (the blue and green nodes, respectively), and hence, available in the first layer graph. Accordingly, we create two edges e_{rg} and e_{rb} between the node n_r in the second layer graph and the nodes n_g and n_b in the first layer graph.

Algorithm 1 describes the process of training a model on the FL client node. The algorithm receives the input training data and returns the multivariate time series graph and cluster medians in a client, which are sent to the server node. First, the client trains a CNN model for a few epochs and extracts the MHAPs from training data (lines 1-7), and then a k-shape clustering [36] is applied to each layer's MHAP list to extract cluster medians (line 8-9). K-shape clustering algorithm considers the unique characteristics of multivariate data, such as phase shift. Similar to k-means, it contains iterative and refining procedures. A shape-based distance measure accounts for phase shift, and a cross-correlation measure identifies centroids. The first step in generating a layer graph is to initialize a graph for each CNN layer with a node size equal to the number of cluster medians for that layer (line 11). Each MHAP is then processed and assigned to its respective clusters, and two consecutive MHAPs are connected by a directed edge (lines 12-14).

The generated layer graph is added to the $graph_{list}$ (line 15). In the final stage, the algorithm merges the layers graphs into a single graph, as explained earlier (line 16).

3.2 FL Server Graph Aggregation

In each round of communication, the server receives the clients' local graph and cluster medians to aggregate them into a global one. For this purpose, the server first determines the similarities between the nodes of the local graphs using their cluster medians and then merges the local graphs based on the node similarities. Finally, the edges of all graphs are augmented into one overall graph. Figure 2 shows an example of two clients' local graphs with three nodes and the merged graph in the server node. At first, the clients train their local model and generate local MHAP graphs (Figure 2, step 1), then they share their graphs with the server node (step 2). The server then aggregates the received graphs (step 3) such that the most similar cluster centers are merged and the final graph has of the same number of nodes as each client local graph, e.g., in Figure 2, the green node from the first graph, n_{a_1}, is similar to n_{b_3} in the second graph. The global graph represents this node as n_A with three edges augmented from the first and second local graphs, i.e., e_{AC}, e_{AB}, and e_{BA}. Note that the edges are weighted. For example, node B in the global graph receives two similar edges from the local graphs ($e_{b_1b_2}$ and $e_{a_3a_2}$); therefore, the global graph increases the weight of edge e_{BC} (shown in a thicker arrow in the figure). After merging the client's local graphs into a global one, the server returns the global graph to all the clients (step 4). They will update their local graph into the global one (step 5) and use it to test their local data in the subsequent communication round.

3.3 FL Client Graph Embedding

After the clients receive the updated global graph from the server, they apply the DeepWalk embedding algorithm [37] to extract the graph node representation vectors $\Psi \in \mathbb{R}^D$, where D is the embedding size. DeepWalk is a graph neural network algorithm that operates directly on graph structures and applies a random path traversal technique to identify localized structures within a network. DeepWalk identifies latent network patterns through random paths in graphs, which are then learned and encoded by neural networks to generate the final embedding.

The clients use the graph node embedding, Ψ, to convert the test data into segments and make predictions via the graph. For that, each multivariate sample is divided into segments, with the corresponding MHAP assigned to each segment. Each MHAP corresponds to one of the graph nodes n_i, therefore, we retrieve the MHAP representation vector, $\Psi(n_i)$, and sum them over the segment. In the next step, all these vectors are merged into a representation vector $\bar{\Psi}$ for the time series sample. The new vector can then be used as an input to any classifier. Because of the promising performance of the XGBoost classifier [4], the clients use XGBoost to classify the new multivariate representations of time series.

Algorithm 2 describes an overall view of a client training process. In each communication round (line 1), the client calls Algorithm 1 on the local training data (line 2) to update or start training (in the first round) a CNN model for a few epochs. A local graph and the cluster centers are retrieved from the client, which are then sent to the server (line 3). After receiving the updated global graph from the server (line 4), the client can test the new graph on its local test data. To this end, the client first creates a new representation of the training data by dividing it into segments (lines 7-15). For each segment s, the client extracts the MHAPs inside the segment and the respective MHAP representation vector Ψ. Note that this representation is accumulated over the MHAPs in a segment (lines 10-13). The client then uses the new representation to create an XGBoost classifier (line 16), which could then be used the same way as trained on the test data to evaluate the model's performance.

Algorithm 2 Training Process in a Client Node

Input: X_{train}: local MTS train set
Output: trained CNN, evolution graph G and its embedding vector Ψ, and trained XGBoost

1: **for** *round* `in` *communication_rounds* **do**
2: (G, CM_{list}) := Generate MHAP Evolution Graph(X_{train})
3: Send (CM_{list}, G) to server
4: G := Get global graph from server
5: Ψ := Embed graph G with vector size D
6: $\bar{\Psi}$:= [] ▷ calculate embedding for train set
7: **for** x `in` X_{train} **do**
8: Divide x into S segments
9: $\bar{\Psi}_x := [0]_D$ ▷ zeros vector of size D
10: **for** s `in` S **do**
11: $\bar{\Psi}_s := [0]_D$
12: **for** all $MHAP$ in s **do**
13: $\bar{\Psi}_s$ += $\Psi(MHAP)$
14: $\bar{\Psi}_x$.add($\bar{\Psi}_s$)
15: $\bar{\Psi}.add(\bar{\Psi}_x)$
16: Train XGBoost on $\bar{\Psi}$
17: **return** G, Ψ, CNN, and XGBoost model

4 RESULTS AND DISCUSSION

We report the FLAMES2Graph frameworkresults on five well-known time-series benchmarks from the Baydogan archive, and HAR and PAM dataset. Our experiments are conducted on a intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz server with 70GB of RAM.

4.1 Datasets

We conducted experiments on several datasets described in the following: and their general statistics are reported in Table 1.

- The Baydogan's archive [3], contains 13 multivariate time series classification datasets. In this work, we choose five datasets from this archive (UWave, ECG, AUSLAN, NetFlow, and Wafer).
- Human Activity Recognition (HAR) [2] dataset recorded daily activities from 30 volunteers and produced six different labels of these activities (walking, walking upstairs, walking downstairs, standing, sitting, and lying).
- PAMAP2 Physical Activity Monitoring (PAM) [39] dataset recorded 18 different daily activities using 17 sensors. We use eight activities with more than 500 recorded samples in this work.

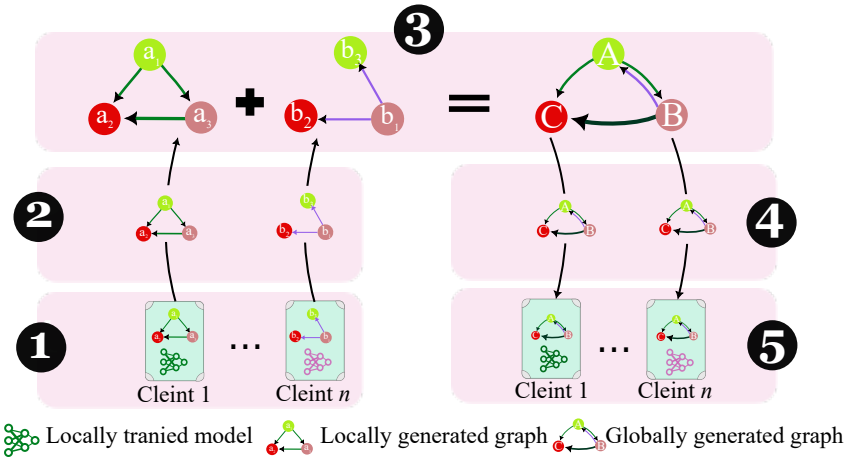


Figure 2: The Client's node in step 1 trains a model and generates the MHAP evolution graphs. In step 2, the clients send the graphs to the server. The server aggregates the local client's graphs in step 3. The global graph is forwarded to the clients in step 4. The clients replace their local graphs with the updated global graph in step 5.

Table 1: Overall statistics of the selected datasets.

Dataset	TS-length	Classes	Dimensions	Samples
ECG	152	2	2	200
Wafer	198	2	6	1, 194
NetFlow	997	2	4	1, 337
AUSLAN	136	95	22	2, 565
UWave	315	8	3	4, 478
PAM	600	8	17	5, 333
HAR	206	6	3	10, 299

4.2 Evaluation Metrics

Classification tasks are typically evaluated based on accuracy. However, accuracy could be biased in favor of the majority class in skewed datasets. Therefore, we report the results for other evaluation metrics, including balance accuracy, sensitivity, specificity, Precision, and F1-score.

4.3 Experiments Setup

In order to simulate federated learning as it occurs in a real-world scenario, we employ NVIDIA FLARE[1]. We set up a server with four clients for each experiment. We test our proposed FLAMES2Graph framework and compare it to other state-of-the-art methods on all seven chosen datasets. Note that, the data is randomly partitioned into four equal size subsets and the equal distribution of the labels for each client is ensured. For all datasets, we run 5-fold cross-validation with 80% for training, 10% for validation, and 10% for test data. Each experiment is run for 100 communication rounds. The client trains the CNN network for five epochs in each round. The graph is tested with an embedding of size 100 and a segment length of size 10. The CNN network has three convolutional blocks for all the datasets with 32, 64, and 128 filters, a batch normalization, and a Rectified Linear Unit (ReLU) activation function.

[1]FLARE Documentation: https://nvflare.readthedocs.io/en/main/flare_overview.html

FLAMES2Graph is compared with various benchmark approaches. We used their proposed experiment setting (network architecture and hyperparameters) for the comparison methods to reproduce similar performance. The benchmark approaches are:

- **Centralized** model of our proposed FLAMES2Graph is the first method to compare. We test all the data as if collected on a central server. The server trains the CNN model on the entire data, extracts the MHAPs, and creates the global graph.
- **FedAvg [28]** is the well-known federated learning strategy that Google introduced. FedAvg averages the clients' trained weights in the central server. In this work, we use a three-layer 1d-CNN network with filters equal to 32, 64, and 128. Note that the CNN used in the FedAvg model has a similar structure and parameters to the one used in FLAMES2Graph.
- **FedRep [7]** learns a shared data representation across clients and a personalized, low-dimensional classifier (local head) for each client. FedRep performs many local updates for every representation update based on the low-dimensional local parameters distributed across clients. In this manner, clients collaborate to learn the global model using all their data, while learning their personalized heads using their local information.
- **FetchSGD [42]** uses a Count Sketch to compress model updates and benefits from the mergeable nature of sketches to combine and update models from several clients. At each communication round, FetchSGD clients compute a gradient based on their local data, compress it using Count Sketch, and send it to an aggregator. The central aggregator keeps momentum and error accumulation of sketches and extracts the weights update from the error accumulation sketch.

Using the default network architecture proposed in the original FedRep and FetchSGD papers resulted in extremely inadequate performance on time series data. Therefore, we replaced their proposed deep neural network with a 1d CNN similar to the one used in our proposed framework. This improved the performance of the models significantly.

Table 2: The average evaluation metrics for FedAvg, FedRep, FetchSGD, and FLAMES2Graph on different benchmark datasets. The numbers in the parenthesis indicate the measurement rank among all the five comparing methods. The best performance among all federated learning models is highlighted.

			Federated Learning			
Dataset	Measure	Centralized	FedAvg	FedRep	FetchSGD	FLAMES2Graph
ECG	Accuracy	0.92 (1)	0.83 (3)	0.83 (3)	0.66 (5)	**0.91 (2)**
	BalanceACC	0.95 (1)	0.63 (3)	0.55 (4)	0.50 (5)	**0.91 (2)**
	F1-Score	0.95 (1)	0.67 (3)	0.58 (4)	0.52 (5)	**0.91 (2)**
	Precision	0.95 (1)	0.64 (3)	0.58 (4)	0.43 (5)	**0.87 (2)**
Wafer	Accuracy	0.98 (1)	0.94 (3)	0.94 (3)	0.11 (5)	**0.98 (1)**
	BalanceACC	0.99 (1)	0.50 (3)	0.50 (3)	0.50 (3)	**0.94 (2)**
	F1-Score	0.98 (1)	0.85 (4)	0.95 (3)	0.11 (5)	**0.96 (2)**
	Precision	0.98 (2)	0.81 (4)	0.95 (3)	0.11 (5)	**1.00 (1)**
NetFlow	Accuracy	0.99 (1)	0.78 (3)	0.78 (3)	0.22 (5)	**0.88 (2)**
	BalanceACC	0.99 (1)	0.67 (3)	0.44 (5)	0.50 (4)	**0.89 (2)**
	F1-Score	0.98 (1)	0.65 (4)	0.89 (3)	0.18 (5)	**0.91 (2)**
	Precision	0.98 (1)	0.79 (4)	0.89 (3)	0.15 (5)	**0.93 (2)**
AUSLAN	Accuracy	0.95 (1)	0.80 (3)	0.65 (4)	0.10 (5)	**0.92 (2)**
	BalanceACC	0.96 (1)	0.65 (3)	0.52 (4)	0.10 (5)	**0.94 (2)**
	F1-Score	0.95 (1)	0.66 (3)	0.65 (4)	0.11 (5)	**0.93 (2)**
	Precision	0.95 (2)	0.66 (3)	0.65 (4)	0.11 (5)	**1.00 (1)**
UWave	Accuracy	0.93 (1)	0.46 (3)	0.30 (4)	0.12 (5)	**0.87 (2)**
	BalanceACC	0.90 (1)	0.46 (3)	0.16 (4)	0.12 (5)	**0.88 (2)**
	F1-Score	0.90 (1)	0.41 (3)	0.15 (4)	0.13 (5)	**0.87 (2)**
	Precision	0.90 (1)	0.64 (3)	0.15 (4)	0.11 (5)	**0.87 (2)**
PAM	Accuracy	0.86 (1)	0.19 (5)	0.23 (3)	0.23 (3)	**0.85 (2)**
	BalanceACC	0.85 (1)	0.22 (3)	0.12 (4)	0.12 (4)	**0.80 (2)**
	F1-Score	0.85 (2)	0.22 (3)	0.20 (4)	0.19 (5)	**0.87 (1)**
	Precision	0.85 (2)	0.22 (3)	0.20 (4)	0.15 (5)	**0.88 (1)**
HAR	Accuracy	0.97 (1)	0.23 (4)	0.41 (3)	0.16 (5)	**0.91 (2)**
	BalanceACC	0.95 (1)	0.24 (4)	0.41 (3)	0.16 (5)	**0.90 (2)**
	F1-Score	0.92 (1)	0.24 (4)	0.42 (3)	0.15 (5)	**0.90 (2)**
	Precision	0.92 (1)	0.24 (4)	0.42 (3)	0.13 (5)	**0.91 (2)**
Average Rank	Accuracy	1.00	3.43	3.29	4.71	**1.86**
	BalanceACC	1.00	3.14	3.86	4.43	**2.00**
	F1-Score	1.14	3.43	3.57	5.00	**1.86**
	Precision	1.43	3.43	3.57	5.00	**1.57**

4.4 Prediction Performance

We compared the accuracy, BalanceAccuracy, F1-score, and Precision performance of FLAMES2Graph to several comparison methods on various well-known multivariate time series benchmarks. We report the measures' mean value of 5-fold cross-validation for each client and then average the client measurements over four FL clients. The centralized setup allows the clients to share their data with a server node. The server then trains the model using the clients' data, i.e., the training takes place on the server. Table 2 reports the results of all comparison methods. Compared to the centralized method, our proposed framework indicates a slight performance drop in all datasets, except in Wafer, where it achieves the same accuracy. This is in line with the fact that the decentralized federated learning methods guarantee clients' data protection at the cost of reducing predictive performance by only receiving clients' trained weights. Therefore, we expect FLAMES2Graph to be outperformed by the centralized model. However, it shows comparable results in most datasets proving that the FLAMES2Graph framework has sufficient convergence capacity, improving issues such as overfitting.

Table 2 also shows that our proposed method outperforms FedAvg in all acquired measures on the UWave, PAM, and HAR datasets. Incidentally, FLAMES2Graph improves the accuracy by about 10% on the ECG, NetFlow, and AUSLAN datasets and about 4% on the Wafer dataset. We also observe that FedRep does not yield promising results when the number of classes increases. For example, the UWave and PAM datasets contain eight classes, and the HAR dataset has six. FedRep shows lower accuracy in these datasets, while its accuracy is acceptable in the other datasets with two classes. When compared to other methods, FetchSGD performs poorly on most datasets. This could be due to the specific sketches that are fine-tuned for the image dataset and are not intuitively scaled for time series data. Therefore, directly applying this method to time series, FetchSGD shows low performance. Improving the created sketches based on the unique nature of time series data could improve the method's performance. Overall, the experiment results show that FLAMES2Graph enhances the predictive performance compared to the state-of-the-art federated frameworks on all tested datasets. Also, it does not merely provide satisfactory classification results for multivariate time series data but also provides

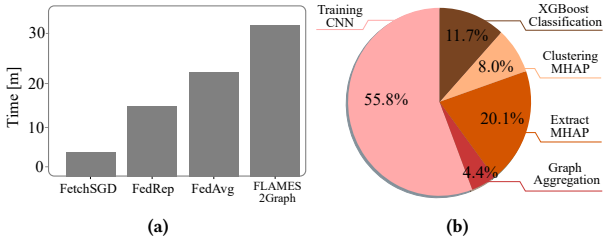
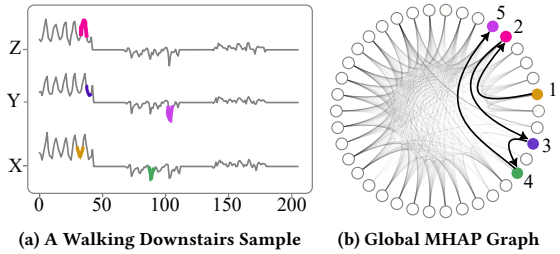


Figure 3: (a) Required runtime comparison of FetchSGD, FedRep, FedAvg, and FLAMES2Graph on the ECG dataset, (b) Time spent on each task in FLAMES2Graph.



(a) A Walking Downstairs Sample **(b) Global MHAP Graph**

Figure 4: An example of a walking downstairs sample from the HAR dataset with its MHAPs from the first CNN layer, (b) The third-round global MHAP evolution graph with colored nodes highlighting MHAPs of (a).

an interpretable network decision, which we will elaborate more in a case study in the following section.

Regarding computational complexity, Figure 3(a) shows that FetshSGD is the quickest framework among the other federated learning methods, and FLAMES2Graph is the slowest due to the traning of a CNN and MHAPs extraction, which depends on the number of input data signals. However, our framework outperforms all other federated frameworks and provides an interpretable model while the runtime is still reasonable. In Figure 3(b), we analyze the time proportion required to run each task in FLAMES2Graph. The pie chart demonstrates that training a CNN network takes the most time, about 55%, and MHAP extraction takes 22%, which is the second most time-consuming task. Note that MHAP extraction depends on the size of the data, i.e., extracting MHAPs from larger datasets requires extra time, as it should locate input data periods that highly activate neurons. Therefore, with growing data size, the search space becomes more expansive, and additional time is required to complete the task. Nonetheless, a previous study showed that extracting these highly activating input periods can clearly improve the MTS model interpretability [57]. The remaining time is divided among the other parts: 11% for XGboost training, about 8% for clustering, and 4% for graph aggregations.

4.5 Quantitative Evaluation

Various methods evaluate a method's interpretability power quantitatively. In a well-known study, Zeiler and Fergus developed a method using perturbation analysis in computer vision in which pixels are zeroed according to their importance [58]. Similarly, a method was developed to assess the quality of interpretation methods for time series data [43, 57]. Therefore, we follow the same idea to evaluate FLAMES2Graph's interpretability quantitatively. In this approach, time series data are perturbed under the assumption that perturbing the most significant feature will reduce accuracy. To evaluate the quality of our framework, we add Gaussian noise with $\mu = 0$ and $\sigma^2 = \{1, 3, 5\}$ to *MHAP*s. Then, we compare the predictive accuracy of the original test data to the perturbed MHAPs. Table 3 shows the predictive accuracy after adding different Gaussian noises to MHAPs. We observe that, on average, MHAP perturbations lead to a significant change in accuracy. This is consistent with the assumption that changing the values of MHAP will notably reduce accuracy, and these periods represent relevant features.

Table 3: The change in accuracy of datasets on perturbed MHAPs. We repeat the experiments by adding a Gaussian noise with $\mu = 0$ and $\sigma^2 : \{1, 3, 5\}$.

		Perturbed MHAPs		
Dataset	*Original*	$\sigma^2 : 1$	$\sigma^2 : 3$	$\sigma^2 : 5$
ECG	0.91	0.82	0.57	0.54
Wafer	0.98	0.95	0.88	0.83
NetFlow	0.88	0.85	0.70	0.65
AUSLAN	0.92	0.88	0.70	0.62
UWave	0.87	0.85	0.74	0.58
PAM	0.85	0.84	0.66	0.60
HAR	0.91	0.90	0.73	0.59
Avg. Change in Accuracy		**0.03**	**0.19**	**0.27**

4.6 Case Study: HAR dataset

In this section, we take a closer look at the interpretability capacity and the effectiveness of our proposed FLAMES2Graph framework by envisaging a case study from the HAR dataset. Figure 4 illustrates an example in a client node with a walking downstairs label and its extracted MHAPs (Figure 4(a)). These extracted MHAPs are represented as nodes with their temporal transitions in the global MHAP evolution graph obtained from the server (Figure 4(b)). We observe that not only does the client's model performance improve by using the FLAMES2Graph framework (i.e., in Table 2), but the client also visualizes the important input subsequences (MHAP) that guide the model to perform well. This graph visualizes the MHAPs and their temporal order in the input data, which is crucial in understanding network decisions on time series data.

Privacy leakage is an important concern in federated learning. Figure 5 empirically elaborates on the FLAMES2Graph effectiveness on this matter. Figure 5(a) presents a random sample from the HAR dataset with its respective MHAPs. Its MHAP clusters from the first CNN layer with their cluster medians are then displayed in Figure 5(b). Figure 5(c) shows the percentage of the extracted MHAPs over the total input data subsequences for each CNN layer. In FLAMES2Graph, the federated clients only share their local graph and cluster centers with the server. This keeps all critical information about the client's private data confidential. The length of cluster centers corresponds to the first CNN kernel size, which is usually very small and not comparable to the original time series

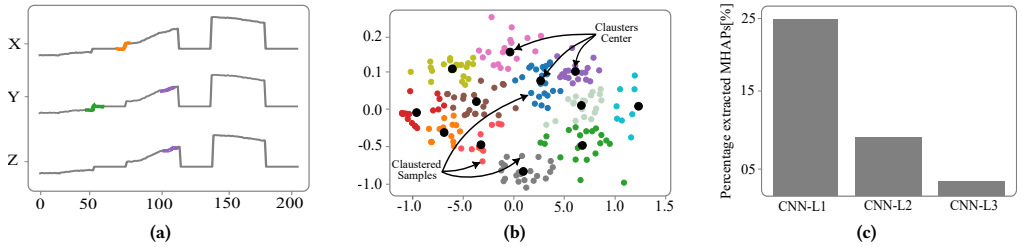


Figure 5: (a) Another example of walking downstairs sample from the HAR dataset with three dimensions and its MHAPs from the first CNN layer, (b) The clustered MHAPs with their centroid (black points) from the HAR dataset. We used PCA for visualization purposes to display the clustered samples in 2D representation, (c) the percentage of extracted MHAPs from input subsequences in each layer.

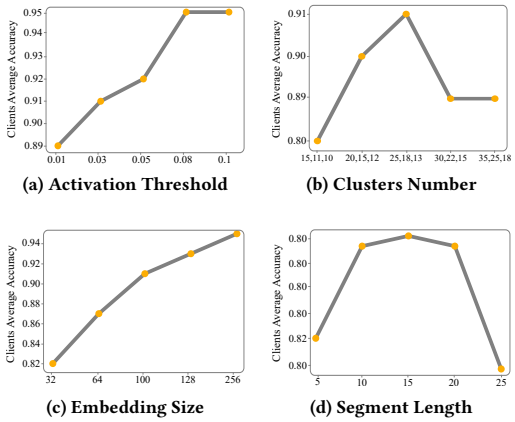


Figure 6: Parameter sensitivity analysis.

length (in this example, equals eight). This means that the center length is approximately 3.8% of the total time series length. Moreover, MHAPs usually do not comprise the whole variant and belong to only one variant of a multivariate sample. Hence, these centers usually represent a portion of data dimensions. Furthermore, the percentage of extracted MHAPs from the total input data subsequences is about 25% for the first CNN layer, 0.08% for the second CNN layer, and 0.01% for the last network layer (Figure 5(c)). This is intuitive because neurons in the upper layers are activated less than neurons in the first layers.

4.7 Sensitivity Analysis

In this section, we investigate the sensitivity of the FLAMES2Graph primary hyperparameters: the CNN activation threshold, clusters size (n in Algorithm 1), segment length (S in Algorithm 2), and graph embedding size (D in Algorithm 2). We report the results in Figure 6.

According to Figure 6(a), the higher the activation threshold, the better the predictive accuracy. However, in this case, there is a trade-off between achieving better accuracy and the increasing complexity of clustering. Since with higher thresholds, more MHAPs are generated, and the clustering method and graph generation become more complex and time-consuming.

Figure 6(b) depicts our investigations on cluster size. Since we use a three-layer CNN, there are three parameters in this regard. Figure 6(b) shows the effect of these parameters, and that the best cluster size selection is ($l_1 = 25, l_2 = 18, l_3 = 13$). Due to the reduced number of MHAPs in upper layers of CNN, it is intuitive that the cluster size in those layers should be set less than the lower layers. We chose these numbers by try and error. For any new dataset one can use a validation set and apply the elbow method [32] to find the best cluster numbers.

Figure 6(c) shows the effect of graph embedding size. We observe that smaller embedding vectors have lower representativeness and hence, less accuracy. On the other hand, high-dimensional embeddings could increase the complexity of XGBoost or any other classifier. Moreover, we observe that the impact of longer vectors is not ever-increasing, e.g., vector lengths of more than 100 do not drastically impact the accuracy in this figure.

Finally, Figure 6(d) illustrates the effect of segment length, which has a bell-shaped curve and indicates that the best segment length should be between 10 and 20. That means segments with a shorter length can discard useful MHAPs, and longer segments can extract overfitted MHAPs.

5 CONCLUSION

This work presents FLAMES2Graph, a personalized federated learning framework for interpreting deep neural networks in multivariate time series data. It visualizes essential input subsequences that activate network neurons in each client and builds a temporal evolution graph to capture time dependencies. FLAMES2Graph reduces communication complexity by sharing only the evolution graphs. Extensive experiments show its superiority over existing methods, comparable to centralized frameworks. Future work includes optimizing accuracy and runtime performance and enhancing privacy and security through graph and centroid encryption.

ACKNOWLEDGMENTS

This research was partially funded by the Federal Ministry of Education and Research (BMBF), Germany under the project LeibnizKILabor with grant No. 01DD20003. The research was performed while the first author was finically covered by the German Federal Ministry of Education and Research (BMBF), Germany under the project ProKI-Hannover with grant No.60172523.

REFERENCES

[1] Sawsan Abdulrahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. 2021. A Survey on Federated Learning: The Journey From Centralized to Distributed On-Site Learning and Beyond. *IEEE Internet of Things Journal* 8, 7 (2021), 5476–5497.

[2] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra Perez, and Jorge Luis Reyes Ortiz. 2013. A public domain dataset for human activity recognition using smartphones. In *Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning*. 437–442.

[3] Mustafa Gokce Baydogan. 2015. Multivariate time series classification datasets. http://www.mustafabaydogan.com/ [Accessed: 2022-09-23].

[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[5] Ziqiang Cheng, Yang Yang, Wei Wang, Wenjie Hu, Yueting Zhuang, and Guojie Song. 2020. Time2graph: Revisiting time series modeling with dynamic shapelets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3617–3624. Issue 04.

[6] Sohee Cho, Ginkyeng Lee, Wonjoon Chang, and Jaesik Choi. 2020. Interpretation of Deep Temporal Representations by Selective Visualization of Internally Activated Nodes. *arXiv preprint arXiv:2004.12538* (2020).

[7] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. 2021. Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning*. PMLR, 2089–2099.

[8] Pierpaolo D'Urso, Livia De Giovanni, Elizabeth Ann Maharaj, and Riccardo Massari. 2014. Wavelet-based self-organizing maps for classifying multivariate time series. *Journal of Chemometrics* 28, 1 (2014), 28–51.

[9] Marco Fisichella and Filippo Garolla. 2021. Can Deep Learning Improve Technical Analysis of Forex Data to Predict Future Price Movements? *IEEE Access* 9 (2021), 153083–153101. https://doi.org/10.1109/ACCESS.2021.3127570

[10] Marco Fisichella, Gianluca Lax, and Antonia Russo. 2022. Partially-federated learning: A new approach to achieving privacy and effectiveness. *Inf. Sci.* 614 (2022), 534–547. https://doi.org/10.1016/j.ins.2022.10.082

[11] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[12] Tsung-Yu Hsieh, Suhang Wang, Yiwei Sun, and Vasant Honavar. 2021. Explainable multivariate time series classification: A deep neural network which learns to attend to important variables as well as time intervals. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 607–615.

[13] Akira Imakura, Hiroaki Inaba, Yukihiko Okada, and Tetsuya Sakurai. 2021. Interpretable collaborative data analysis on distributed data. *Expert Systems with Applications* 177 (2021), 114891.

[14] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[15] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*. PMLR, 5132–5143.

[16] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. In *Proc. of the NIPS Workshop on Private Multi-Party Machine Learning*.

[17] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. 2020. Survey of personalization techniques for federated learning. In *Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. 794–797.

[18] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. 2012. Mobile data offloading: How much can WiFi deliver? *IEEE/ACM Transactions on networking* 21, 2 (2012), 536–550.

[19] Chenglin Li, Di Niu, Bei Jiang, Xiao Zuo, and Jianming Yang. 2021. Meta-har: Federated representation learning for human activity recognition. In *Proceedings of the Web Conference 2021*. 912–922.

[20] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. 2021. Federated learning on non-iid data silos: An experimental study. (2021). https://arxiv.org/abs/2102.02079

[21] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proceedings of Machine Learning and Systems*, Vol. 2. 429–450.

[22] Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. 2020. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523* (2020).

[23] Zhiyu Liang and Hongzhi Wang. 2023. FedST: Federated Shapelet Transformation for Interpretable Time Series Classification. *arXiv preprint arXiv:2302.10631* (2023).

[24] Bingyan Liu, Yao Guo, and Xiangqun Chen. 2021. PFA: Privacy-preserving Federated Adaptation for Effective Model Personalization. In *Proceedings of the Web Conference 2021*. 923–934.

[25] Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. 2021. Gated transformer networks for multivariate time series classification. *arXiv preprint arXiv:2103.14438* (2021).

[26] Jing Ma, Qiuchen Zhang, Jian Lou, Li Xiong, and Joyce C Ho. 2021. Communication efficient federated generalized tensor factorization for collaborative health data analytics. In *Proceedings of the Web Conference 2021*. 171–182.

[27] Xiaodong Ma, Jia Zhu, Zhihao Lin, Shanxuan Chen, and Yangjie Qin. 2022. A state-of-the-art survey on solving non-IID data in Federated Learning. *Future Generation Computer Systems* 135 (2022), 244–258. https://doi.org/10.1016/j.future.2022.05.003

[28] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[29] Jed Mills, Jia Hu, and Geyong Min. 2019. Communication-efficient federated learning for wireless edge intelligence in IoT. *IEEE Internet of Things Journal* 7, 7 (2019), 5986–5994.

[30] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition* 65 (2017), 211–222.

[31] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. 2021. A survey on security and privacy of federated learning. *Future Generation Computer Systems* 115 (2021), 619–640. https://doi.org/10.1016/j.future.2020.10.007

[32] Andrew Ng. 2012. Clustering with the k-means algorithm. *Machine Learning* (2012), 1–2.

[33] Jaehoon Oh, SangMook Kim, and Se-Young Yun. 2022. FedBABU: Toward Enhanced Representation for Federated Image Classification. In *International Conference on Learning Representations*.

[34] Carlotta Orsenigo and Carlo Vercellis. 2010. Combining discrete SVM and fixed cardinality warping distances for multivariate time series classification. *Pattern Recognition* 43, 11 (2010), 3787–3794.

[35] Junjie Pang, Yan Huang, Zhenzhen Xie, Qilong Han, and Zhipeng Cai. 2021. Realizing the Heterogeneity: A Self-Organized Federated Learning Framework for IoT. *IEEE Internet of Things Journal* 8, 5 (2021), 3088–3098.

[36] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1855–1870.

[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[38] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. 2017. A dual-stage attention-based recurrent neural network for time series prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 2627–2633.

[39] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *The 16th international symposium on wearable computers*. 108–109.

[40] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " 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*. 1135–1144.

[41] David Roschewitz, Mary-Anne Hartley, Luca Corinzia, and Martin Jaggi. 2021. IFedAvg: Interpretable Data-Interoperability for Federated Learning. *arXiv preprint arXiv:2107.06580* (2021).

[42] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. 2020. Fetchsgd: Communication-efficient federated learning with sketching. In *International Conference on Machine Learning*. 8253–8265.

[43] Udo Schlegel, Hiba Arnout, Mennatallah El-Assady, Daniela Oelke, and Daniel A Keim. 2019. Towards a rigorous evaluation of xai methods on time series. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 4197–4201.

[44] Skyler Seto, Wenyu Zhang, and Yichen Zhou. 2015. Multivariate time series classification using dynamic time warping template selection for human activity recognition. In *2015 IEEE symposium series on computational intelligence*. IEEE, 1399–1406.

[45] Anooshiravan Sharabiani, Houshang Darabi, Ashkan Rezaei, Samuel Harford, Hereford Johnson, and Fazle Karim. 2017. Efficient classification of long time series by 3-d dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47, 10 (2017), 2688–2703.

[46] Gurtaj Singh, Vincenzo Violi, and Marco Fisichella. 2023. Federated Learning to Safeguard Patients Data: A Medical Image Retrieval Case. *Big Data Cogn. Comput.* 7, 1 (2023), 18. https://doi.org/10.3390/bdcc7010018

[47] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. 2021. Feddl: Federated learning via dynamic layer sharing for human activity recognition. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 15–28.

[48] Shahid Ullah and Caroline F Finch. 2013. Applications of functional data analysis: A systematic review. *BMC medical research methodology* 13, 1 (2013), 1–12.
[49] Guan Wang. 2019. Interpret federated learning with shapley values. *arXiv preprint arXiv:1905.04519* (2019).
[50] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 7611–7623.
[51] Jane-Ling Wang, Jeng-Min Chiou, and Hans-Georg Müller. 2016. Functional data analysis. *Annual Review of Statistics and Its Application* 3 (2016), 257–295.
[52] Zhiguang Wang, Weizhong Yan, and Tim Oates. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*. IEEE, 1578–1585.
[53] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. 2010. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter* 12, 1 (2010), 40–48.
[54] Xue Yang, Yan Feng, Weijun Fang, Jun Shao, Xiaohu Tang, Shu-Tao Xia, and Rongxing Lu. 2022. An Accuracy-Lossless Perturbation Method for Defending Privacy Attacks in Federated Learning. In *Proceedings of the ACM Web Conference 2022*. 732–742.
[55] Lexiang Ye and Eamonn Keogh. 2009. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 947–956.
[56] Raneen Younis and Marco Fisichella. 2022. FLY-SMOTE: Re-Balancing the Non-IID IoT Edge Devices Data in Federated Learning System. *IEEE Access* 10 (2022), 65092–65102. https://doi.org/10.1109/ACCESS.2022.3184309
[57] Raneen Younis, Sergej Zerr, and Zahra Ahmadi. 2022. Multivariate Time Series Analysis: An Interpretable CNN-based Model. In *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–10.
[58] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.
[59] Fanglan Zheng, Kun Li, Jiang Tian, Xiaojia Xiang, et al. 2020. A vertical federated learning method for interpretable scorecard and its application in credit scoring. *arXiv preprint arXiv:2009.06218* (2020).
[60] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2921–2929.