Privacy-Preserving Distributed Learning for Residential Short-Term Load Forecasting

Yi Dong*§, Yingjie Wang*†, Mariana Gama‡, Mustafa A. Mustafa¶‡, Geert Deconinck†, and Xiaowei Huang§

***Abstract*—In the realm of power systems, the increasing involvement of residential users in load forecasting applications has heightened concerns about data privacy. Specifically, the load data can inadvertently reveal the daily routines of residential users, thereby posing a risk to their property security. While federated learning (FL) has been employed to safeguard user privacy by enabling model training without the exchange of raw data, these FL models have shown vulnerabilities to emerging attack techniques, such as Deep Leakage from Gradients and poisoning attacks. To counteract these, we initially employ a Secure-Aggregation (SecAgg) algorithm that leverages multiparty computation cryptographic techniques to mitigate the risk of gradient leakage. However, the introduction of SecAgg necessitates the deployment of additional sub-center servers for executing the multiparty computation protocol, thereby escalating computational complexity and reducing system robustness, especially in scenarios where one or more sub-centers are unavailable. To address these challenges, we introduce a Markovian Switching-based distributed training framework, the convergence of which is substantiated through rigorous theoretical analysis. The Distributed Markovian Switching (DMS) topology shows strong robustness towards the poisoning attacks as well. Case studies employing real-world power system load data validate the efficacy of our proposed algorithm. It not only significantly minimizes communication complexity but also maintains accuracy levels comparable to traditional FL methods, thereby enhancing the scalability of our load forecasting algorithm.**

***Index Terms*—Load Forecasting, Data Privacy, Distributed Learning, Federated Learning, Secure Aggregation, Collaborative Work**

I. Introduction

ELECTRIC load forecasting plays an essential role in power scheduling, planning, operating and management [1], [2]. The stability of the power system is under threat due to the intermittence of renewable energy generations and the complex nature of utility-customer interactions and dynamic behaviors. To overcome this, residential Short-term Load Forecasting (STLF) has been widely studied to facilitate the power system operations [3], [4]. However, it is evident that residential-user privacy is at risk when residential-user load data is collected and mined [5]. For example, the residential user's daily routines and presence at home can be detected with a high probability from its electric load data, which will directly affect the residential user's property safety. Therefore, how to accurately forecast residential power load while ensuring data privacy becomes an open challenge.

The Federated Learning (FL) method has been introduced in recent years to overcome the challenges of residential-user privacy. It can decouple the data storage from the training process [6], while reaching a desirable accuracy compared to the centrally trained model [7], [8]. There are already efforts to apply FL to power system forecasting to preserve the sensitive individual consumption profiles [9]–[14]. Yang *et al.* integrate variational mode decomposition, federated k-means clustering algorithm, and SecureBoost together for STLF with data privacy protection [9]. Jun *et al.* propose a novel method for disaggregating community-level behind-the-meters solar generation using a federated learning-based Bayesian neural network, which can preserve the confidentiality of utilities' data [11]. Yong *et al.* propose a verifiable and oblivious secure aggregation for FL [15]. Their algorithm could tolerate the high drop-up rate of clients during large-scale FL training. Asad *et al.* highlight recent FL algorithms and evaluate their communication efficiency in detailed comparisons. The experimental results indicate that training data-driven models using FL not only enhances security and privacy, but also reduces communication costs [16].

The above-mentioned FL based algorithms claim the advantage of privacy in terms of not sharing the original data. With the gradual success of various privacy attack technologies in different applications, the load data can also be reverted if the attacker can access the gradients from agents [17]. Here, we use a simple example to show the threat of the attack algorithm to the artificial intelligence-based load forecasting algorithms, shown in Fig. 1.

In Fig. 1, the two figures on the left are the original consumption data and the original image after conversion. The sub-figures on the right are the regenerated images from the attacker after they obtained the original gradients. As we can see, the attacker can accurately regenerate the original data only after 10 iterations. Therefore, any leakage of the gradient can cause a significant threat to customers' data. To prevent information leakage from agents' gradients, Bonawitz *et al.* [18] proposed using Multi-Party Computation (MPC) to aggregate the gradients privately. Multiparty computation is a cryptographic technique that allows a group of mistrustful

* Both authors contributed equally to this research.

§ Department of Computer Science, University of Liverpool, UK, {yi.dong, xiaowei.huang}@liverpool.ac.uk

† Electa, Department of Electrical Engineering (ESAT), KU Leuven and EnergyVille, Belgium, {tony.wang, geert.deconinck}@kuleuven.be

‡ COSIC, Department of Electrical Engineering (ESAT), KU Leuven, Belgium, mariana.botelhodagama@kuleuven.be

¶ Department of Computer Science, The University of Manchester, UK, mustafa.mustafa@manchester.ac.uk

This work is supported by the UK EPSRC (End-to-End Conceptual Guarding of Neural Architectures [EP/T026995/1]) and the FWO SBO project SNIPPET (Secure and Privacy-Friendly Peer-to-Peer Electricity Trading [S007619N]). This project has received funding from the European Union's Horizon 2020 (FOCETA: grant agreement No 956123) and UKRI (SPACE: project No 10046257).

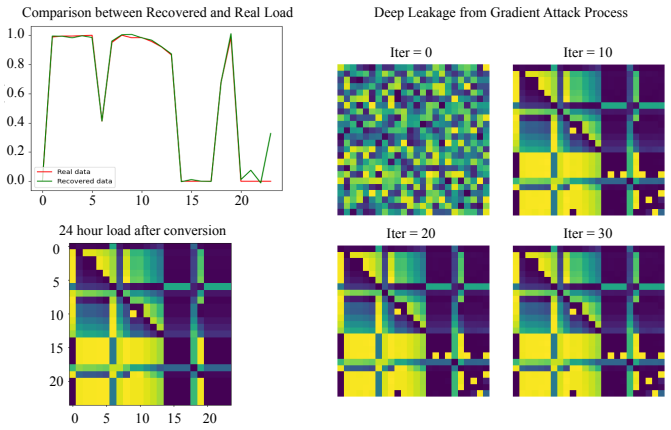


Fig. 1: Deep leakage from gradient attack.

parties to perform a computation over secret input data. By using MPC, a group of agents can obtain the aggregated value of all their gradients without ever revealing any individual gradient. The ideas presented in [18] were extensively used and developed in further work related to secure aggregation for privacy-preserving federated learning, such as [19]–[24]. Since these works focus on the federated learning setting, the secure aggregation mechanisms rely on the existence of at least one central server. For the works using additive masking [18], [22], [23], a single server is sufficient. In [21], [24], the central aggregator is distributed, meaning that at least two servers are required.

Multiparty computation achieves privacy by distributing the computation between different parties, which then need to communicate with each other according to the chosen MPC protocol to obtain the result of the desired calculation without compromising private input data. The parties running the MPC protocol can either be the agents themselves, or external servers that receive the data from the agents already in encrypted form. However, there are two possible issues:

1) *Scalability*: performing computations over private data leads to an increase in communication. Setting the MPC parties as external servers helps diminish this overhead, but still results in an expanded network, with each agent needing to connect to every MPC party.

2) *Single-point failures*: if the central server in a federated learning system experiences a single-point failure, then the entire system may become unavailable until the server is restored. The MPC protocol we use does not solve the issue, as it needs all connected parties. However, due to the nature of our proposed network topology, no single party (or server) will need to remain online during the full training process.

Besides the Deep Leakage from Gradients (DLG) attacks, the poisoning attack is another common threat towards FL models. A poisoning attack involves one or a few malicious clients injecting extreme outliers or tailored data into the training dataset or model updates to undermine or mislead the global model performance [25]. Some researchers developed a Byzantine-robust federated learning model against the poisoning attacks in various ways, such as detecting malicious clients based on their impact on the model performance and constructing new loss functions [26]–[29]. However, the common Byzantine-robust models cannot always defend the poisoning attacks, particularly when some of the clients conspire together [30]. Among the more advanced defenses, there are two significant approaches. “Romoa” is a robust model aggregation solution to both solo and collusive attacks. With a hybrid similarity measurement method, it can detect the attacks precisely whether it is targeted or untargeted attacks [31]. “FLTrust” reserved a small clean dataset as a reference at the server. By cross-validation, the central server gives a trust score to each collected update. It provides a more precise and less computationally costly way to detect malicious clients [32].

To overcome the above-mentioned challenges, we borrow the idea from consensus-based distributed optimization [33]–[35], which has flexibility in connected network graphs and initialization-free advantages. In this paper, we propose a fully decentralized Markovian Switching short-term residential-user load forecasting algorithm. The proposed distributed algorithm employs Markovian Switching topologies [36], which can randomly select sets of agents to co-train the model, and therefore reduce the communication complexity in theory and accelerate encryption speed. Furthermore, the Distributed Markovian Switching (DMS) topology is instinct-resilient against poisoning attacks. In the centralized FL training, the central server updates the global model with the average updates across the whole client group each round. Even though the malicious clients take a small part of the whole group, the accumulated strewing effect can significantly undermine the global model over multiple rounds. Unlike the centralized FL setting, the DMS takes a random set of clients each round, which reduces the chance of malicious clients poisoning attack for every round. The above-mentioned cumulative effect will not happen in the DMS setting.

The major contributions of this work are summarized as follows:

1) A secure and safe distributed algorithm is proposed for short-term load forecasting and its convergence has been theoretically proved. Different from most existing studies, e.g., [5], [9] and references therein, which are extensively concentrated on federated algorithms, the proposed algorithm in this paper is based on fully Peer-to-Peer (P2P) distributed consensus learning without a central server.
2) The DMS is developed for the proposed framework. Different from the traditional FL, the DMS has inherited robustness towards both poisoning attacks and DLG attacks. Besides that, the DMS topology shows advantages in efficiency compared with other distributed learning topologies.
3) We apply the Secure Aggregation (SecAgg) to the proposed distributed learning topologies. The SecAgg ensures no original gradients are shared. Therefore it guarantees data privacy from DLG attacks by design. Additionally, we analyzed its impact on the original complexity as an extra add-up layer.
4) The proposed algorithm has been successfully validated on a real power system load forecasting dataset. Furthermore, the reproducibility and replicability of our work are

guaranteed since all source code is available on GitHub for open access[1].

The rest of this paper is organized as follows. The mathematical preliminaries used in this paper are summarized and the researched problem is formulated in Section II. A secure-aggregated and Markovian switching distributed framework for STLF is proposed and analyzed in Section III. Simulation results and corresponding analysis are presented in Section IV. Finally, Section V concludes this paper.

II. Preliminaries

In this section, we recall some basic concepts (federated learning and distributed learning) and preliminaries related to the proposed STLF model, including graph theory and secure aggregation.

A. Federated Learning

FL is a distributed training framework for neural networks. It decouples the training process and the requirement for centralized data storage. Instead of collecting raw data from the individual agent, the raw data stays locally with agents. The agents use their own data to train the global model, M_j. Then, the central server collects the weights updates from the agents. Based on the selected algorithm, for example the Federated Average (FedAvg), the server will average the submitted weights and use them to update the global model to M_{j+1}. In this way, agents can participate in the model training without revealing their raw data. Figure 2 shows the procedure of FL.

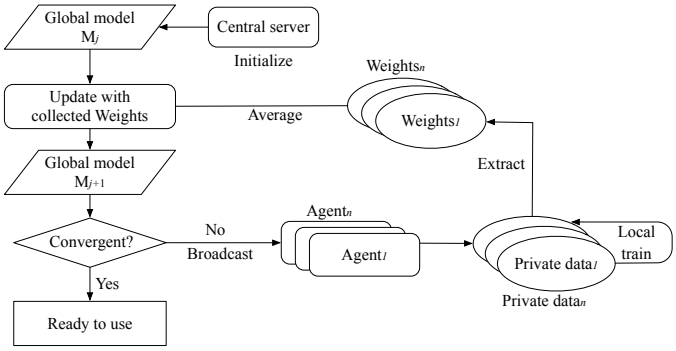


Fig. 2: Training procedure of federated learning.

In the FL framework, the submitted local model update can be "gradients" or "weights". It depends on the communication bandwidth, but they are practically the same. In the privacy analysis, we use the term "weights" for consistency.

B. Graph Theory

In this paper, it is assumed that the information can be communicated between different computing agents. Therefore, the distributed load forecasting conforms to the characteristics of the undirected graph, which can enhance the stability of communicated graphs and reduce the convergence time [37].

[1]https://github.com/YingjieWangTony/FL-DL.git

Let $V = \{1, ..., N\}$ be a set of N local agents. Assume that, at any time t, the agents constitute a graph $\mathcal{G}^t = (V^t, E^t)$, where $V^t \subseteq V$ is a set of agents and $E^t \subseteq V^t \times V^t$ is a set of edges. Whenever $(i, j) \in E^t$, we say that they are neighbors. We write $\mathbf{L}^t$ as the Laplace matrix of $\mathcal{G}^t$.

$$\mathbf{L}_{ij}^t = \begin{cases} -a_{ij}, & i \neq j \\ \sum_{i \neq j} a_{ij}, & i = j \end{cases} \tag{1}$$

To enhance privacy and safety during the training process, the communication graph in this paper is considered as a time-variance topology, called Markovian switching network topology, which means that the graph $\mathcal{G}^t$ might change over time due to the addition and deletion of agents, which forms a temporal graph $\mathcal{G} = \{\mathcal{G}^t\}_{t=0,1,...}$. The Markovian switching network has q substructures. The transition probability matrix is $\mathbb{T} = [\mathcal{T}_{ij}] \in \mathbb{R}^{q \times q}$, where $\mathcal{T}_{ij} = P\{\mathcal{G}^{t+1} = j | \mathcal{G}^t = i\}, \forall i, j \in Q$ with Q being the graph structure topology set.

C. Distributed Learning

Different from the FL framework, the distributed learning algorithm in this paper is designed in a fully decentralized setting, where the agents can only communicate with their adjacent neighbors.

From [38], the training process of the distributed learning algorithm can be summarized as:

$$\phi_i(k) = \theta_i(k) - \gamma \nabla_{\theta_i} \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i(k))) \tag{2}$$

$$\theta_i(k+1) = \alpha \sum_{j \in \mathcal{N}} a_{ij} \phi_j(k) \tag{3}$$

where $\theta_i(k)$ and $\phi_i(k)$ denote the weights of the neural network before and after the kth training iteration, $\gamma, \alpha > 0$ are the learning rate, and $\nabla(\cdot)$ is the gradient term regarding empirical risk $\mathcal{L}$, which is a measure of how well a machine learning model performs on a given dataset. It is defined as the average of the loss function (i.e., the discrepancy between predicted values and actual values) over the entire dataset. $\mathcal{D}_i$ and $M_{PF,i}$ denote the dataset and local model of ith agent, respectively.

Notice that the equations (2) and (3) are also called an Learning-Then-Consensus (LTC) algorithm, where we can easily swap the equations (2) and (3) to get the twinned Consensus-Then-Learning (CTL) algorithm.

D. Secure Aggregation

Secure aggregation is achieved through a cryptographic technique, called MPC, which allows us to perform calculations over encrypted data. There are different types of MPC protocols, with different security guarantees. In this work, we use SCALE-MAMBA [39], a software framework that implements MPC protocols based on secret sharing which have active security, meaning that the protocol remains secure even if up to a certain percentage of the parties performing the computation behaves maliciously and deviates arbitrarily from the protocol. Additionally, the protocols in SCALE-MAMBA are in the preprocessing model. In this model, there is an offline phase, where input-independent data is generated. This

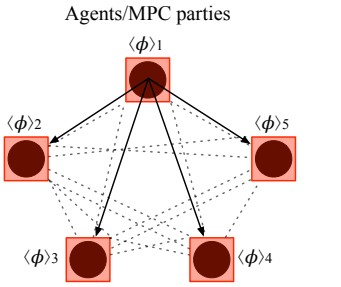
can be done at any time, including before the inputs for the desired computation are even available. Later, during the online phase, we can perform our computation faster by using the preprocessed data.

In this paper, we consider an MPC protocol based on the Shamir secret sharing scheme. In this scheme, in order to share a secret value s with a set of ν parties, the person holding s must generate a secret polynomial $f(X)$ of degree h such that $f(0) = s$. By randomly generating the other polynomial coefficients $f_1, ..., f_h$, the following polynomial is obtained:

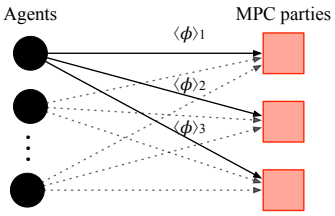
$$f(X) = s + f_1 \cdot X + ... + f_h \cdot X^h.$$

Then, each party i is given a secret share $s_i = f(i)$. Note that each secret share by itself looks random, but when at least $h+1$ parties combine their shares then the original secret value can be retrieved via Lagrange interpolation. The choice of degree h will determine how many dishonest parties we allow without compromising the secret since any set of h or fewer parties cannot reconstruct the polynomial. However, to be able to perform all the operations correctly with this scheme, it is also required that a majority of parties is honest.

We write $\langle x \rangle$ when we refer to a secret shared value x, and each secret share i is represented as $\langle x \rangle_i$. All these values are elements of a finite field $\mathbb{F}_p$, where p is a prime number, and operations between secret shared values also take place in $\mathbb{F}_p$.



(a) MPC parties played by the agents.



(b) MPC parties played by external servers.

Fig. 3: Agent secret sharing a gradient ϕ with the MPC parties.

To perform linear operations, i.e., adding two secret shared values or multiplying a secret shared value by a public scalar in $\mathbb{F}_p$, it is enough for each party to locally perform these operations on their own shares. E.g., to calculate $\langle z \rangle = \langle x \rangle + a \cdot \langle y \rangle$, each party i should compute $\langle z \rangle_i = \langle x \rangle_i + a \cdot \langle y \rangle_i$. Multiplying two secret shared values or performing more complex operations such as comparisons requires communication between the parties running the MPC protocol so that the correct secret shared result can be reached without revealing the inputs. Indeed, communication is generally the bottleneck for computations with MPC. However, for secure aggregation only addition is needed, and so communication is only required for the secret sharing of the inputs and reconstructing the final aggregated results.

As illustrated in Fig. 3, it is possible to have agents also act as an MPC party, in which case each agent needs to secretly share its own gradients with the other agents (who are also acting as MPC parties). In order to improve scalability, we can instead have external servers play the role of MPC parties. In this case, we can have as few as three MPC parties, to whom the agents will send secret shares of their gradients. These external servers should be run by entities with conflicting interests, in order to avoid collusion and therefore maintain the privacy of the gradients. Note that we should always aggregate more than two gradients in each round, since otherwise the two agents whose weights were aggregated will be able to derive each other's gradients from the revealed aggregated value.

Remark 1: We note that our use of Secure Aggregation is not intended to be an improvement upon previous work in the federated learning setting. Indeed, this work is in a fundamentally different setting due to our proposed network topology. Although some existing papers already analyze the use of different topologies for improving the efficiency of secure aggregation [22], [23], they always assume the existence of a central aggregation server. Additionally, Differential Privacy (DP) has also been considered in previous works to ensure that even the final aggregated result does not leak any information [19], [20]. We do not analyze the use DP since it is not directly affected by the choice of topology. Nonetheless, it would be relatively straightforward to adapt previous techniques [40] for combining MPC and DP to our use case.

III. Distributed Markovian Switching Algorithm

A. Problem Formulation

In this paper, we focus on the privacy-preserving residential short-term load forecasting problem, where every agent i holds a private dataset $\mathcal{D}_i$, and we define $\mathcal{D} = \mathcal{D}_1 \cup ... \cup \mathcal{D}_N$. The N agents will communicate and train a model M_{DL} where DL stands for distributed learning. This will be compared with a federated learning model M_{FL} (i.e., the server collects and modifies the weights from all local agents during the training) and a model M_C trained by centralized learning (i.e., the server collects all raw data from the agents before training).

B. DMS Learning

The proposed short-term load forecasting algorithm is to utilize decentralized distributed learning to collectively aggregate the local gradients. In a fully decentralized setting, the agents can only send/collect messages to/from their neighbors but have a collective objective of training a global model M_{DL}, which is required to be high-performing and privacy-preserving.

Every distributed agent is regarded as a self-training agent and cooperates with each other using distributed neural networks with a neurodynamic consensus-based approach, as shown in Fig. 4. The framework consists of a group of neural networks distributed over a connected graph, where the update of coefficients includes two stages: in the first stage, the

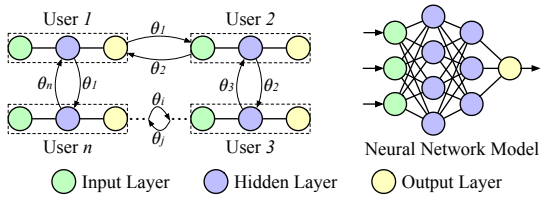


Fig. 4: An illustrative framework of distributed learning.

ith agent trains its neural network locally using the dataset $\mathcal{D}_i$, and in the second stage, the consensus of their locally trained weights is performed. With this setting, all the agents are able to obtain the same neural network if every sub-dataset is available to at least one local agent, which will be demonstrated by the convergence analysis.

Minimizing the empirical risk allows the model to generalize well to new, unseen data. The idea is that if the model is trained on a dataset that is representative of the real-world data it will encounter in the future, then minimizing the empirical risk should result in a model that can make accurate predictions on new data. Therefore, the objective of the network is to minimize the empirical risk over the entire dataset, given by

$$\min \mathcal{L}(\mathcal{D}, M_{PF}(\theta)) = \sum_{i=1}^{N} \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i)) \tag{4}$$

where $\mathcal{D}$ and $\mathcal{D}_i$ denote the entire dataset and the sub-dataset of agent i, respectively; θ and θ_i are the weights of the global and local neural networks; M_{PF} and $M_{PF,i}$ represent the models of the global neural network and the local ones, respectively. It is worth noting that the local learning process, i.e.,

$$\min \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i)) \tag{5}$$

does not necessarily yield the minimization of the global empirical risk in (4), since the models $M_{PF,i}$ are different. Consequently, this necessitates the employment of consensus tools in cooperation with the training process.

Note that the sub-dataset of each agent is assumed to be stationary. For the ith agent with n_i samples in $\mathcal{D}_i$, the gradient can be calculated by

$$\nabla_{\theta_i} \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i)) = \frac{1}{n_i} \sum_{k=1}^{n_i} \nabla_{\theta_i} \ell(s^k, \theta_i) \tag{6}$$

where s^k represents the kth data sample in $\mathcal{D}_i$, and $\ell(s^k, \theta_i)$ is the empirical risk of sample s^k on a model with weights θ_i. Similarly, the global gradient is given by

$$\nabla_{\theta} \mathcal{L}(\mathcal{D}, M_{PF}(\theta)) = \frac{1}{n} \sum_{k=1}^{n} \nabla_{\theta} \ell(s^k, \theta) \tag{7}$$

where $n = \sum_{i=1}^{N} n_i$ is the total number of samples in $\mathcal{D}$.

Gradient-based methods have been widely used in training neural networks, due to their high efficiency and flexibility [41]–[43]. In this paper, a gradient descent algorithm for agent i is formulated as

$$\phi_i(k) = \theta_i(k) - \gamma \nabla_{\theta_i} \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i(k))) \tag{8}$$

$$\langle \theta_i(k+1) \rangle = \alpha \sum_{j \in \mathcal{N}} a_{ij} \langle \phi_j(k) \rangle \tag{9}$$

where $0 \leq \alpha \leq 1$ is positive constant; a_{ij} denotes the communication link between agent i and agent j; $a_{ij} > 0$ if they can communicate, otherwise, $a_{ij} = 0$. Note that for each agent i, the consensus only requires the updated weights ϕ_j from its adjacent neighbors $j \in \mathcal{N}_i$ with $\mathcal{N}_i$ denoting the neighbors of agent i. Recall also that we add the updated weights using secure aggregation, hence the secret share notation used in Equation 9. The shares of $\langle \theta_i(k+1) \rangle$ will then be sent by the MPC parties to agent i, who can finally reconstruct the value $\theta_i(k+1)$.

Based on the above discussion, the overall structure of the fully decentralized distributed algorithm for a region can be summarized in Algorithm 1.

Algorithm 1 Distributed Markovian Switching (DMS)

Initialization:
for each agent $i \in \mathcal{V}$:
1. initialize the weights of local neural network θ_i;
2. set the required consensus speed by changing the coefficients in Equation (9).

Iteration:
3. set $k := k + 1$, for $i \in \mathcal{V}$
4. train the local neural network using the gradient descent method (8) with locally stored dataset $\mathcal{D}_i$ to obtain $\phi_i(k)$;
5. apply secure aggregation to encrypt original weights $\langle \phi_i(k) \rangle$;
6. run the consensus operation (9) to drive the globally averaged weights $\theta_i(k+1)$ by communicating with its neighbors $\mathcal{N}_i$;

Termination: termination condition is satisfied or iteration budget is reached.

C. Correctness Analysis on Performance/Convergence

We need to show that Algorithm 1 is able to converge with all agents having the same model. Considering the generality, we choose Mean Squared Error (MSE) as the loss function.

MSE is a widely used measure in statistics and machine learning for quantifying the difference between estimated values and the actual values [44]. It is calculated as the average of the squares of the errors or deviations, that is, the difference between the estimator and what is estimated. Mathematically, MSE is defined as

$$MSE = \frac{1}{n_s} \sum_{i=1}^{n_s} (y_i - \hat{y})^2 \tag{10}$$

where n_s is the number of samples, y_i and $\hat{y_i}$ are the true and predicted values, respectively.

First of all, two assumptions are needed. With these two assumptions and the mean value theorem, we can obtain the closed-loop MSE dynamics, and therefore the convergence of the MSE can be proved with a designed learning rate.

Assumption 1: The Hessian matrix of local empirical risk is bounded, i.e.,:

$$\underline{p}_i \mathbf{I}_n \leq \nabla^2 \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i)) \leq \overline{p}_i \mathbf{I}_n \tag{11}$$

where $0 \leq \underline{p}_i \leq \overline{p}_i$ are non-negative constants.

Assumption 2: The gradient noise, w_i, satisfies the following properties i.e.,:

$$\mathbb{E}[w_i] = 0 \tag{12}$$

$$\mathbb{E}[\|w_i\|^2] \leq \xi_i^2 \tag{13}$$

where ξ_i is non-negative constant, $\|\cdot\|$ denotes the L2-norm of the argument.

Remark 2: Assumption 1 indicates that the network objective, $\sum_{i=1}^{N} \mathcal{L}(\mathcal{D}_i, M_{PF,i}(\theta_i))$, is strongly convex, which means that the neural network has a unique and optimal solution/weights θ^*. In distributed optimization problems, some assumptions have been made, including the use of bounded Hessian matrices for convergence analysis. Other works [45]–[47] have used bounded gradients with set constraints, which is not feasible for unconstrained problems. Our assumption is less restrictive because we can solve problems with unbounded gradients.

Assumption 2 implies that the noise, w_i, is unbiased and has uniformly bounded variance ξ_i^2, which is a stronger assumption and has been considered in many studies [42], [48].

Theorem 1: Under Assumptions 1 and 2, if the learning rates satisfy $0 \leq \gamma_i \leq \frac{2}{\overline{p}_i}$, the worst MSE, $\|\mathbb{N}\|^\infty$ in the distributed learning network converges to

$$\lim_{k\to\infty} \|\mathbb{N}(k)\|^\infty \preceq \alpha\Xi \tag{14}$$

where γ_i is the local learning rate for ith agent. $\overline{p}_i$ is the local upper bound of the Hessian matrix as shown in equation (11). $\mathbb{N}$ is the MSE vectors of the distributed neural networks, which is defined later in equation (25). $\Xi = diag(\xi_1^2, \cdots, \xi_N^2)$ is the matrix format of the uniformly bounded variances.

Remark 3: In Theorem 1, $0 \leq \gamma_i \leq \frac{2}{\overline{p}_i}$ provides an upper bound for designing the learning rates, based on which we can prove the convergence of the proposed algorithms. Equation (14) reveals how the learning rates, the bounds of the Hessian matrix, and the noise variance will affect the performance of the algorithms. It means that minimizing the learning rates can lead to a reduction in the optimality error. The designer can balance between the convergence speed and the optimality error according to the real applications by adjusting the heterogeneous learning rates. In other words, this theorem serves as a reference to select learning rates during our training process in both DL and FL frameworks. In (14), we have 3 parameters, $\overline{p}_i, \alpha$ and ξ_i. $\overline{p}_i$ can be obtained from the loss function, $\alpha \leq 1$ is a fixed parameter, and the coefficient of gradient noises ξ_i should be approximated according to the practical applications.

Proof:

To simplify the notations in the analysis process below, we rewrite the weights in a compact form by integrating all weights into a vector form as:

$$\mathbf{\Phi}(k) = \text{col}(\phi_1(k), \cdots, \phi_N(k)) \tag{15}$$

$$\mathbf{\Theta}(k) = \text{col}(\theta_1(k), \cdots, \theta_N(k)) \tag{16}$$

Then, we can rewrite the proposed distributed algorithm (8) and (9) in the following form:

$$\mathbf{\Phi}(k) = \mathbf{\Theta}(k) - (\Gamma \otimes \mathbf{I}_n)\boldsymbol{\nabla}\mathcal{L}(\mathbf{\Theta}(k)) + \mathbf{\Omega}(k) \tag{17}$$

$$\mathbf{\Theta}(k+1) = \alpha(\mathcal{A} \otimes \mathbf{I}_n)\mathbf{\Phi}(k) \tag{18}$$

where $\Gamma = diag(\eta_1, \cdots, \eta_N)$, $\boldsymbol{\nabla}\mathcal{L}(\mathbf{\Theta}) = col(\nabla\mathcal{L}_1(\mathbf{\Theta}), \cdots, \nabla\mathcal{L}_N(\mathbf{\Theta}))$, $\mathbf{\Omega} = col(\omega_1, \cdots, \omega_N)$. $\mathcal{A}$ is the Laplacian matrix, which is defined as $\mathcal{A} = [a_{ij}]_{N\times N}$. $a_{ij} > 0$ if the agent i is communicating with agent j. Note that each row of the $\mathcal{A}$ sums to 1, which is also called row-stochastic matrix [37].

To confirm that the MSE converges, we transfer the weights to an error recursion formation:

$$\hat{\mathbf{\Phi}}(k) = \mathbf{\Phi}(k) - \theta^*\mathbf{1}_N \tag{19}$$

$$\hat{\mathbf{\Theta}}(k) = \mathbf{\Theta}(k) - \theta^*\mathbf{1}_N \tag{20}$$

Furthermore, to relate the gradient term $\boldsymbol{\nabla}\mathcal{L}(\mathbf{\Theta}(k))$ with error term $\hat{\mathbf{\Theta}}(k)$, we have the mean value theorem from [49]:

$$\nabla_z g(y) = \nabla_z g(x) + [\int_0^1 \nabla_z^2 g[x + \tau(y-x)]d\tau](y-x) \tag{21}$$

Substituting the $\mathbf{\Theta}(k)$ and $\theta^*\mathbf{1}_N$ into equation (21) as y and x:

$$\boldsymbol{\nabla}_\theta\mathcal{L}(\mathbf{\Theta}(k)) = \boldsymbol{\nabla}_\theta\mathcal{L}(\theta^*\mathbf{1}_N) + [\int_0^1 \nabla_\theta\boldsymbol{\nabla}\mathcal{L}[\theta^*\mathbf{1}_N + \tau(\hat{\mathbf{\Theta}}(k))]d\tau](\hat{\mathbf{\Theta}}(k)) \tag{22}$$

Since $\boldsymbol{\nabla}_\theta\mathcal{L}(\theta^*\mathbf{1}_N) = 0$, and we let $P(k) = \int_0^1 \nabla_\theta\boldsymbol{\nabla}\mathcal{L}[\theta^*\mathbf{1}_N + \tau(\hat{\mathbf{\Theta}}(k))]d\tau$. The equation (17) can be rewritten as:

$$\hat{\mathbf{\Phi}}(k) = \hat{\mathbf{\Theta}}(k) - (\Gamma \otimes \mathbf{I}_N)P(k)\hat{\mathbf{\Theta}}(k) + \mathbf{\Omega}(k) \tag{23}$$

Combining (18) and (23) together, we can get the error term update law as:

$$\hat{\mathbf{\Theta}}(k+1) = \alpha(\mathcal{A}\otimes\mathbf{I}_n)[\mathbf{I}_n N - (\Gamma\otimes\mathbf{I}_N)P(k)]\hat{\mathbf{\Theta}}(k) + \alpha(\mathcal{A}\otimes\mathbf{I}_n)\mathbf{\Omega}(k) \tag{24}$$

We define the MSE vectors of the distributed neural networks as:

$$\mathbb{M}(k) = \text{col}(\mathbb{E}\|\hat{\boldsymbol{\phi}}_1(k)\|^2, \cdots, \mathbb{E}\|\hat{\boldsymbol{\phi}}_N(k)\|^2) \tag{25}$$

$$\mathbb{N}(k) = \text{col}(\mathbb{E}\|\hat{\boldsymbol{\theta}}_1(k)\|^2, \cdots, \mathbb{E}\|\hat{\boldsymbol{\theta}}_N(k)\|^2) \tag{26}$$

By taking the Euclidean norm for each agent's error dynamics, we have:

$$\|\hat{\boldsymbol{\theta}}_i(k+1)\|^2 = \|\alpha\sum_{j\in\mathcal{N}} a_{ij}\hat{\boldsymbol{\phi}}_j(k)\|^2 \tag{27}$$

Since the norm function $\|\cdot\|$ is a convex function, applying Jensen's inequality to (27):

$$\|\hat{\boldsymbol{\theta}}_i(k+1)\|^2 \leq \alpha\sum_{j\in\mathcal{N}} a_{ij}\|\hat{\boldsymbol{\phi}}_j(k)\|^2 \tag{28}$$

Taking the expectation of both sides of (28), we can obtain:

$$\mathbb{N}(k+1) \preceq \alpha\mathcal{A}\mathbb{M}(k) \tag{29}$$

Taking the Euclidean norm and expectation of $\hat{\mathbf{\Phi}}_i(k)$ leads to:

$$\mathbb{M}(k) = \mathbb{E}\|[\mathbf{I}_{nN} - (\Gamma\otimes\mathbf{I}_N)P(k)]\hat{\mathbf{\Theta}}(k)\|^2 + \mathbb{E}\|\mathbf{\Omega}(k)\|^2 \tag{30}$$

Based on Assumption 1, the bound of the symmetrical matrix $P(k)$ could be obtained:

$$\underline{p}_i \mathbf{I}_n \leq P(k) \leq \overline{p}_i \mathbf{I}_n \tag{31}$$

and therefore:

$$0 \leq [\mathbf{I}_{nN} - (\Gamma \otimes \mathbf{I}_N)P(k)] \leq \overline{\lambda}_i^2 \mathbf{I}_n \tag{32}$$

where $\overline{\lambda}_i = \max(1-\gamma_i \underline{p}_i)^2, (1-\gamma_i \overline{p}_i)^2$.

Based on Assumption 2, we have:

$$\mathbb{E}\|\mathbf{\Omega}(k)\|^2 \leq \Xi^2 \tag{33}$$

Until now, the closed-loop MSE error can be derived as:

$$\mathbb{N}(k+1) \preceq \alpha \mathcal{A}\Lambda \mathbb{N}(k) + \alpha \mathcal{A}\Xi \tag{34}$$

where $\Lambda = diag(\overline{\lambda}_1, \cdots, \overline{\lambda}_i)$

Considering the worst case MSE, $\|\mathbb{N}(k+1)\|^\infty$,

$$\|\mathbb{N}(k+1)\|^\infty \preceq \alpha \|\mathcal{A}\Lambda\|^\infty \|\mathbb{N}(k)\|^\infty + \alpha \|\mathcal{A}\Xi\|^\infty \tag{35}$$

To guarantee the convergence of (35), we must have $\|\mathcal{A}\Lambda\|^\infty \leq 1$. We also have $\|\mathcal{A}\|^\infty = 1$ and $\alpha \leq 1$, so we have

$$\overline{\lambda}_i \leq 1 \tag{36}$$

which means:

$$(1-\gamma_i \underline{p}_i)^2 \leq 1 \tag{37}$$

$$(1-\gamma_i \overline{p}_i)^2 \leq 1 \tag{38}$$

Because of $\underline{p}_i \leq \overline{p}_i$, we should choose the learning rate between the range to guarantee the convergence of the proposed distributed learning algorithm:

$$0 \leq \gamma_i \leq \frac{2}{\overline{p}_i} \tag{39}$$

■

After the convergence analysis, we also provide a theoretical analysis of the proposed algorithm due to the complexity of the algorithm is not easy to visualize. We evaluate the complexity based on the communicated components' size with a normalized factor, $O(n|E|)$. The algorithm is a gradient-based algorithm, which is a widespread technique in machine learning and optimization. Gradient-based algorithms are efficient in terms of computation, as they only require the calculation of the gradient of the objective function, which can be done using simple mathematical operations. This makes the algorithm easy to implement and computationally efficient, which is a desirable feature in many real-world applications.

In terms of communication complexity, the proposed algorithm only requires the agents to communicate with their immediate neighbors. In other words, each agent only needs to exchange information with the agents that are in its direct vicinity. This limited communication requirement is beneficial as it reduces the overall complexity of the algorithm and makes it more suitable for distributed implementation. Moreover, the proposed algorithm removes the requirement of the central server, which can be a bottleneck in some systems.

In this setting, the communication complexity for each agent is $O(n|E|)$, where E is the set of communication edges and n is the number of iterations, because in every iteration the agent only needs to send one message to their neighbors and every message is of constant size $O(1)$. The computational complexity for every agent is also $O(n|E|)$, since every iteration the agent can process the received messages and update the local state in a linear way. It is noticed that due to the randomly connected characteristic of Markovian switching topology, the communication edges in each round are not a constant value, e.g. $E \in [1, 30], n = 30$.

IV. Case Studies

In our study, we propose a new training strategy called DMS learning, which utilizes a dynamic switching mechanism between different topologies during the learning process. To evaluate the performance of DMS, we compare it with four other popular training strategies: Centralized Learning, FedAvg, Distributed Learning with Fully-Connected (DFC), and Distributed Learning with Ring Topology (DRING). Here, we use Fig. 5 to illustrate the communication differences between different learning strategies. These five training strategies are implemented on four state-of-the-art deep learning neural network models: DNN [50], CNN [51], Wavenet [52], and LSTM [3]. The experiments are conducted on a dataset with a large number of samples, and the results are evaluated based on the accuracy, convergence rate, and communication cost of the trained models.

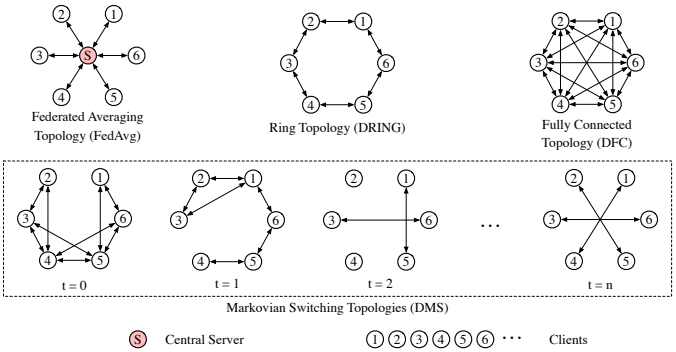


Fig. 5: Communication topologies of different strategies.

A. Data Sources

The Smart Metering Electricity Customer Behavior Trials are used as our case study [53]. It contains over 5000 Irish home and business participants during 2009 and 2010. Their electricity consumption is recorded by smart meters with 30 minutes intervals. The longest record is from January 1st, 2009 to June 30th, 2010. After data cleaning and clustering, we selected 30 households to present a virtual energy community. The selection was made among 30 houses that were clustered together with all methods and obtained a high score in each.

More specifically, in this case, we use a typical non-Independently Identically Distributed (non-IID) dataset with a large group of agents. It is not realistic and practical to directly apply FL to the raw dataset. Therefore, we performed

the K-means algorithm to cluster the dataset into small groups [54]. The clustering result can be seen as Fig. 6.

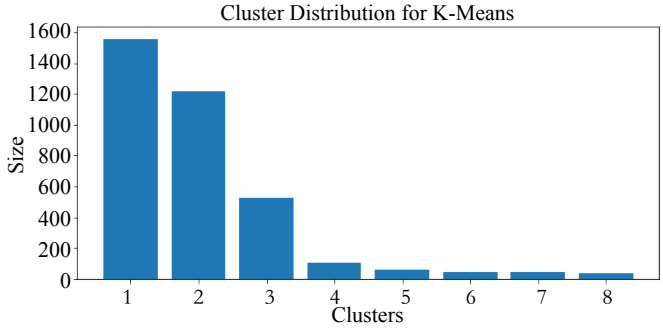


Fig. 6: The clustering result with K-means.

As we can see from Fig. 6, $Cluster_1$ contains most of the agents. It indicates this cluster represents the most common consumption behavior. Therefore, we randomly select 30 agents from $Cluster_1$ to perform our further study in this paper.

B. Experiment Setup

The evaluation is based on simulations. We programmed the forecasting code in Python and based on the machine learning framework provided by FLOWER[2](flwr), where the neural network models are written in PyTorch[3].

For the secure aggregation experiments, we use four machines with an Intel i-9900 CPU and 128GB of RAM, and four machines with an Intel Core i7-770 CPU and 32GB of RAM. For experiments with up to 4 MPC parties, only the first group of machines is used (one machine per MPC party). For experiments with more than 8 parties, each machine runs more than one party. The ping time between the machines is 1.003 ms.

C. Experiment Results and Analysis

In this section, we evaluate the performance of the proposed algorithm by assessing it against four key aspects: scalability, privacy, accuracy, and complexity. By evaluating the algorithm from these four key perspectives, we can ensure that it meets the standards required for practical use.

a) Scalability: We measure the algorithm's ability to handle increasing amounts of data and agents, ensuring that it can scale to meet the needs of large-scale applications. The evaluation method for scalability is the convergence speed. The proposed algorithm can handle a large number of agents with a relatively small number of neighbors. This allows for more efficient use of resources and makes it possible to use the algorithm in a distributed setting, where the number of agents can be large. To demonstrate the scalability of the proposed algorithm, we first analyze the converge steps in the slowest Distributed Learning with Ring Topology (DRING). Then, we conducted an experiment to work with up to 200 distributed learning agents and 3 different communication topologies (DMS, DFC [38] and DRING [55]). The experimental results are shown in Fig. 7.

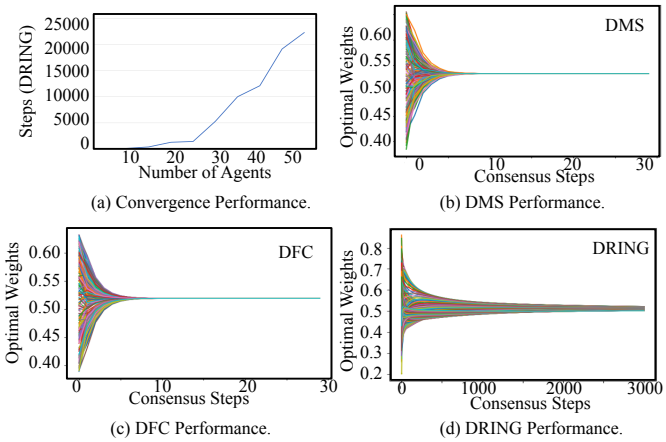


Fig. 7: Scalability of different decentralized learning topologies.

The experiment was designed to evaluate the effect of increasing the number of agents on the algorithm's convergence performance. Figure 7(a) shows that the convergence performance increases almost linearly with the number of agents under the DRING topology, while the other three figures show the convergence of a 200-agent system with different topologies. This indicates that the proposed algorithm is scalable and can easily adapt to large-scale federated learning systems. Actually, both the communication and computational complexity of an agent do not depend on the set of agents, but rather the set of neighbors. This means that as long as the number of neighbors is kept unchanged, the algorithm can scale to as many agents as possible. This is a significant advantage over other federated learning algorithms that have a high computational and communication complexity with respect to the number of agents. From the simulation results, it can be observed that due to the random communication approach, the DMS topology significantly improves convergence speed compared to the DRING topology, and achieves convergence speed almost equivalent to that of the DFC topology. Theoretically, the DMS topology reduces 50% of the communication edges compared to the DFC topology.

b) Privacy: For secure aggregation, we used an MPC protocol based on Shamir secret sharing which remains secure as long as the majority of the parties are honest (i.e., when there are three parties, at most one can be malicious). If this is satisfied and a dishonest party deviates from the protocol, the honest parties will detect it and abort the protocol with probability $1 - 1/p$, where p is the order of the field. We choose p such that $\log_2 p = 128$, so that dishonest behavior is caught with overwhelming probability.

We implemented secure aggregation using SCALE-MAMBA and performed experiments for different communication topologies. For the FedAvg topology, we consider 3 external MPC parties, receiving secret shared gradients from the 30 agents in every round. For the DRING topology, each agent acts as an MPC party and performs the secure aggregation protocol together with the two connected neighbors. For

[2]https://flower.dev/
[3]https://pytorch.org/

the DFC topology, there are 30 MPC parties played by the agents themselves, who secretly share their gradients with all other 29 agents in every round. In the DMS topology, the MPC parties are played by the subset of agents chosen to aggregate their gradients in that round.

The runtimes obtained for one round of the different topologies are summarized in Table I. DFC clearly results

TABLE I: Runtimes in seconds for computing secure aggregation of gradients for one round of training.

FedAvg	DRING	DFC	DMS		
			3 parties	4 parties	5 parties
1.08s	0.16s	488.41s	0.16s	0.25s	0.39s

in the most expensive secure aggregation computation, taking $488.41s$ ($\approx$ 8min). This was to be expected, since using more MPC parties quickly increases the communication overhead. On the other hand, for this experiment, we ran more than one party on a single machine, with some machines running up to six parties, which also contributed to the slow runtime. In FedAvg, the fact that there are only 3 MPC parties significantly improves the performance. However, DRING and DMS allow even faster computation times. In DRING, each group of three neighboring agents is aggregating 3 gradients in each round (instead of the 3 MPC parties aggregating 30 gradients in FedAvg). If all the groups of 3 agents perform their computations in parallel, a runtime of 0.16s is achieved. Regarding DMS, the number of agents aggregating their gradients in each round can be adjusted according to the desired runtimes. Naturally, increasing this number will also result in a slower aggregation, especially since the number of MPC parties running the protocol will increase. Nonetheless, aggregation with 5 parties in DMS still achieves runtimes more than twice as fast as FedAvg.

Remark 4: Due to a limitation of the SCALE-MAMBA software, each gradient to be aggregated must initially be entered as three separate values (one for the sign and two for the value itself). These values are then put together to obtain the original gradient value before performing the aggregation itself. However, this means that we need to secret share 3 times as many values, resulting in runtimes 2 to 3 times slower than if only the value itself was secretly shared. The runtimes in Table I assume that the gradients can be read as one single secret value.

Due to the use of different network topologies, models and experimental setups, comparing the numerical benchmarks in Table I with those of previous related work is not straightforward. Instead, we look at the computation and communication complexities for performing secure aggregation for the communication topologies considered in this work. We compare them to corresponding complexities for the protocol in [21], which in turn presents lower complexities than other previous works (as is shown in [24, Table 2]).

The setting of the protocol in [21] is similar to the FedAvg topology, hence the identical complexities. With the DFC topology, the absence of a distributed central server results in high communication and computation complexities for the clients, which can become unpractical as the number of clients increases. The DRING and DMS topology allow removing

TABLE II: Computation and communication complexity per training iteration of the different topologies compared with SAFELearn [21]. m is the length of the model updates and n is the number of clients.

Approach	Computation		Communication	
	Server	Client	Server	Client
SAFELearn [21]	$\mathcal{O}(mn)$	$\mathcal{O}(m)$	$\mathcal{O}(mn)$	$\mathcal{O}(m)$
FedAvg	$\mathcal{O}(mn)$	$\mathcal{O}(m)$	$\mathcal{O}(mn)$	$\mathcal{O}(m)$
DRING	-	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$
DFC	-	$\mathcal{O}(mn)$	-	$\mathcal{O}(mn)$
DMS	-	$\mathcal{O}(m)$	-	$\mathcal{O}(m)$

the central server while maintaining low complexities on the clients' side. Note that the number of MPC parties also influences the complexities but is not considered in this table. This is because in [21], FedAvg and DRING the number of parties is small and constant throughout the protocol. In the DMS topology, as the complexity analysis in Section III-C, the worst-case scenario for DMS involves the number of interacting clients in each round equal to the total number of clients. When using DFC, the MPC parties are the clients, and hence their impact is already accounted for.

c) Robustness to attacks: In this section, we conducted experiments to evaluate the robustness of our proposed method, DMS, to two different attack algorithms, poisoning attacks, and DLG. With the poisoning attack scenario, we arbitrarily and randomly attacked 10% of the agents (3 out of 30) and attackers introduced a poisoning attack by injecting a deliberate malicious perturbation of 0.2 into their broadcasted model weights during each training round. We monitored the resulting mismatch between the model's parameters and the optimal value throughout the training process. The outcomes of these experiments are presented in Fig. 8, which illustrates a comprehensive comparison between our DMS approach and FedAvg.

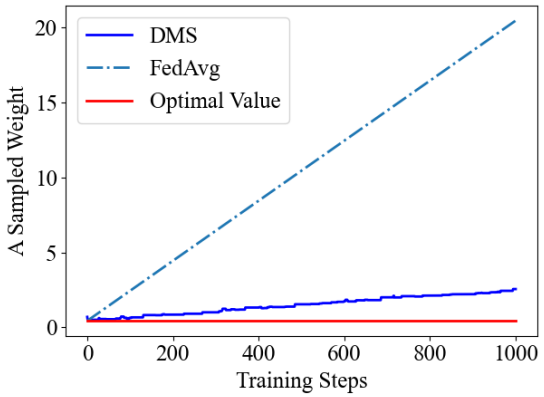


Fig. 8: Weights update under attacks.

From the observations in Fig. 8, it is evident that our DMS exhibits superior robustness compared to the FedAvg algorithm under poisoning attacks. Even under a sustained onslaught of over 1000 rounds of attacks, our method displayed only a 9% increase in error, unlike FedAvg. This robustness, which refers to maintaining accuracy under attack, can be attributed to our novel approach that adapts to

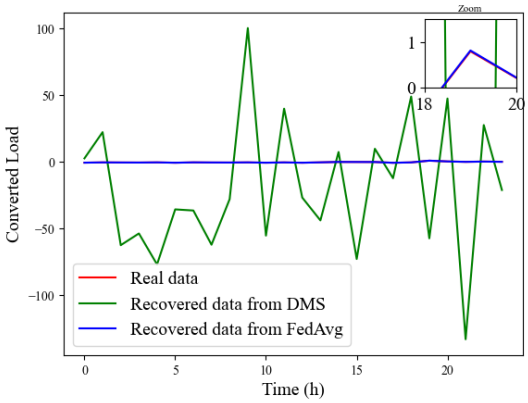


Fig. 9: Recovered data from DLG attack.

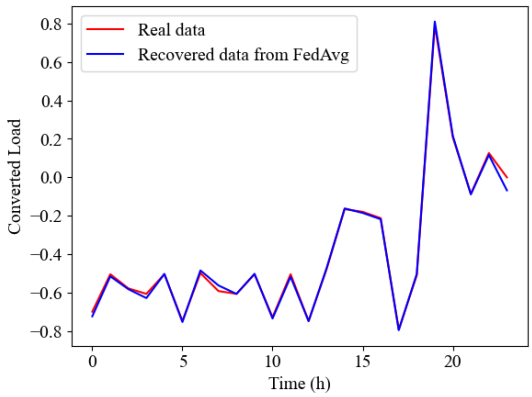


Fig. 10: Comparison between recovered and real converted Load under FedAvg setting.

changing communication topologies. In our implementation of the Markovian switching mechanism, only a subset of agents participate in weight updates each round, hindering attackers from consistently propagating their malicious weights. This inconsistency significantly enhances the system's robustness against adversarial influences. These results underscore the effectiveness of our DMS method in defending against poisoning attacks, suggesting its potential to significantly enhance the security of learning systems.

The second scenario is under the attack of DLG. DLG is a strong attack against gradients-sharing-based NN training. Once it hijacks the real gradients, the attacker can recover the original training set. In this experiment, an attacker randomly selects a communication line and gains continuous access to the transmitted information. By doing so, the attacker can obtain the change in weights between two communications, effectively capturing gradient information. This data is then used to recover load data. For benchmarking purposes, we compare our results against the FedAvg algorithm in this experiment. The experiment results are shown in Fig. 9.

Fig. 9 shows the recovered data by employing the DLG algorithm. It can be seen that our DMS algorithm performs better than the FedAvg algorithm in terms of protecting the information leakage from the gradient. In FedAvg, the central server initializes and broadcasts the global model to clients. Therefore, each client holds an identical NN model with the same randomly initialized weights. Meanwhile, for every round of training, each client needs to share their updates with the central server. In this case, the attacker can succeed if they have access to different stages of weights, for example, the initialized weights and any weights from later updates. The comparison of recovered data under FedAvg setting can be clearly seen as 10. By contrast, the attacker faces more difficulties in the DMS setting. First, the clients' models are not unified initialized. Second, the communicating clients in each round of training are randomly selected. Therefore, it is way more difficult for the attacker to hijack weights in different stages for a specific client.

d) Accuracy: Here, we measure the accuracy of the algorithm's predictions. The evaluation metric we used is MSE. We conduct an experiment to forecast short-term residential load using four different state-of-the-art models: DNN, CNN, LSTM, and WaveNet. For each model, we implement four different algorithms: FedAvg, DRING, DFC and DMS. Firstly, the forecasting results of three randomly selected residentials are shown in Fig. 11, for the full forecasting results, please refer to our GitHub[4].

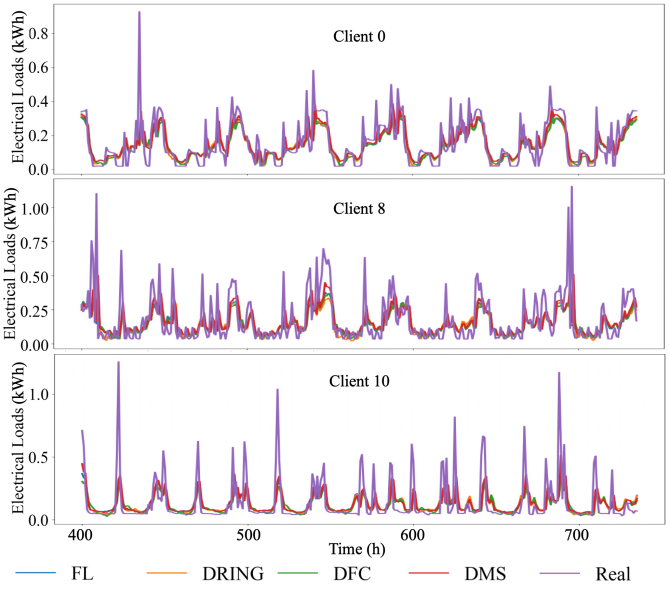


Fig. 11: Forecasting performance of different algorithms on DNN models.

From Fig. 11, we can see that the proposed DMS algorithm performs well and is able to capture the trends and fluctuations in the actual electricity consumption values quite well. It is slightly better than other algorithms in most time slots. It is noticed that there are some deviations between the actual and every predicted peak value in Fig. 11. That is largely due to the loss function, MSE. Using MSE in regression problems is common, but it has some limitations when the data is imbalanced. First, MSE treats all errors equally, regardless if their values are high or low. In other words, MSE will not prioritize peak value accuracy. It leads to an underestimation of peak values. Second, when the data is imbalanced, the peak

[4]https://github.com/YingjieWangTony/FL-DL.git

values are often considered as outliers during optimization. While MSE is sensitive to outliers, it aims to minimize the error across all samples uniformly. This can be problematic when peaks are outliers but are very important. Thirdly, global optimization could be a reason, too. The optimization process aims to minimize the global error. Peak values are rare in normal consumption profiles, the optimization process may choose to "sacrifice" accuracy on these points in favor of better global accuracy. We propose the following solutions to this situation, custom loss function, data resampling, feature engineering, and ensemble methods. These solutions could be a good help towards peak value prediction. However, this is beyond the current scope of the paper.

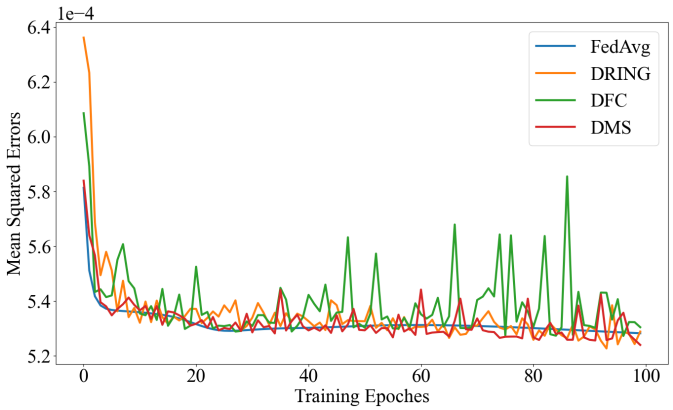


Fig. 12: Training performance of different algorithms on DNN models.

In Fig. 12, we show the training performance among four different algorithms. Mean squared error is applied to evaluate the performance of the models. To further illustrate the comparison results among different models and algorithms, we summarize the overall mean square error values for each model and algorithm on the testing dataset, as shown in Table. III.

TABLE III: Mean square error (kWh) of different algorithms and models.

	DNN	CNN	LSTM	WAVENET
FedAvg	0.05263	0.06495	**0.0537**	0.4965
DRING	0.05289	0.06498	0.0556	**0.3659**
DFC	0.05305	0.06493	0.0552	0.5043
DMS	**0.05236**	**0.06388**	0.0550	0.5156

From the results given in Fig. 12 and Table III, the proposed DMS algorithm meets the requirements and has better accuracy than other models. For the dataset used in this case, the DMS algorithm proposed obtains better performance and fewer prediction errors in most cases. It is noticed that some fluctuations during the training process, even after 100 training epochs. These fluctuations are primarily caused by the heterogeneity in local data distributions. Each node trains on a distinct dataset, leading to models that are finely tuned to their specific data characteristics. When these models are aggregated, as in Federated Learning, the process attempts to reconcile these divergent updates, causing fluctuations in the global model. This phenomenon is exacerbated in environments where the local datasets are not independently and identically distributed. While theoretically, a centralized training approach using a powerful computing center and all available data would yield an optimal model, practical constraints like geographical limitations and privacy concerns make this approach infeasible. Thus, distributed training, despite its inherent fluctuations, becomes a necessary compromise, offering a balance between model performance and adherence to logistical and privacy constraints. Over time, as more rounds of aggregation occur, the global model tends to stabilize, gradually adapting to the diversity of local updates.

V. Conclusion

In this paper, we developed a Markovian Switching distributed learning framework for residential short-term load forecasting. Moreover, a secure aggregation approach, MPC has been employed to address the threat of deep leakage from the gradient. We analyzed DMS from several perspectives, such as accuracy, scalability, complexity and privacy. The DMS is compared with traditional centralized, FL, DRING and DFC models. The simulation shows that the DMS model not only secures residential-user privacy but also shows equivalent or even superior accuracy than the other models. Particularly, it significantly reduces computational complexity and enhances scalability compared to FL models.

References

[1] O. Abedinia, N. Amjady, and H. Zareipour, "A new feature selection technique for load and price forecast of electrical power systems," *IEEE Trans. Power Syst.*, vol. 32, no. 1, pp. 62–74, 2016.
[2] M. Ali, M. Adnan, and M. Tariq, "Optimum control strategies for short term load forecasting in smart grids," *Int. J. Electr. Power Energy Syst.*, vol. 113, pp. 792–806, 2019.
[3] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.
[4] W. Lin, D. Wu, and B. Boulet, "Spatial-temporal residential short-term load forecasting via graph neural networks," *IEEE Trans. Smart Grid*, vol. 12, no. 6, pp. 5373–5384, 2021.
[5] J. S. Nightingale, Y. Wang, F. Zobiri, and M. A. Mustafa, "Effect of clustering in federated learning on non-iid electricity consumption prediction," in *2022 IEEE PES Innov. Smart Grid Technol. Conf. Eur.*, 2022.
[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal process. mag.*, vol. 37, no. 3, pp. 50–60, 2020.
[7] N. Gholizadeh and P. Musilek, "Federated learning with hyperparameter-based clustering for electrical load forecasting," *Internet Things*, vol. 17, p. 100470, 2022.
[8] J. D. Fernández, S. P. Menci, C. M. Lee, A. Rieger, and G. Fridgen, "Privacy-preserving federated learning for residential short-term load forecasting," *Appl. Energy*, vol. 326, p. 119915, 2022.
[9] Y. Yang, Z. Wang, S. Zhao, and J. Wu, "An integrated federated learning algorithm for short-term load forecasting," *Electr. Power Syst. Res.*, vol. 214, p. 108830, 2023.
[10] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards personalized federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, 2022.
[11] J. Lin, J. Ma, and J. Zhu, "A privacy-preserving federated learning method for probabilistic community-level behind-the-meter solar generation disaggregation," *IEEE Trans. Smart Grid*, vol. 13, no. 1, pp. 268–279, 2021.
[12] J. Gao, W. Wang, Z. Liu, M. F. R. M. Billah, and B. Campbell, "Decentralized federated learning framework for the neighborhood: a case study on residential building load forecasting," in *Proc. 19th ACM conf. embed. networked sens. syst.*, 2021, pp. 453–459.
[13] M. Savi and F. Olivadese, "Short-term energy consumption forecasting at the edge: A federated learning approach," *IEEE Access*, vol. 9, pp. 95 949–95 969, 2021.

[14] M. N. Fekri, K. Grolinger, and S. Mir, "Distributed load forecasting using smart meter data: Federated learning with Recurrent Neural Networks," *Int. J. Electr. Power Energy Syst.*, vol. 137, p. 107669, 2022.

[15] Y. Wang, A. Zhang, S. Wu, and S. Yu, "VOSA: Verifiable and oblivious secure aggregation for privacy-preserving federated learning," *IEEE Trans. Dependable Secure Comput.*, pp. 1–17, 2022.

[16] M. Asad, A. Moustafa, T. Ito, and M. Aslam, "Evaluating the communication efficiency in federated learning algorithms," in *2021 IEEE 24th int. conf. comput. support. coop. work des.*, 2021, pp. 552–557.

[17] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Adv. neural inf. process. syst.*, vol. 32, 2019.

[18] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. 2017 ACM SIGSAC conf. comput. commun. secur.*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191.

[19] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM workshop artif. intell. secur.*, ser. AISec'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–11.

[20] D. Byrd and A. Polychroniadou, "Differentially private secure multi-party computation for federated learning in financial applications," in *Proc. first ACM int. conf. AI finance*, ser. ICAIF '20. New York, NY, USA: Association for Computing Machinery, 2021.

[21] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Mollering, T. Nguyen, P. Rieger, A. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "Safelearn: Secure aggregation for private federated learning," in *2021 IEEE secur. priv. workshops*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 56–62.

[22] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *Proc. 2020 ACM SIGSAC conf. comput. commun. secur.*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1253–1269. [Online]. Available: https://doi.org/10.1145/3372297.3417885

[23] J. So, B. Guler, and A. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. PP, pp. 1–1, Jan. 2021.

[24] T. Gehlhar, F. Marx, T. Schneider, A. Suresh, T. Wehrle, and H. Yalame, "Safefl: Mpc-friendly framework for private and robust federated learning," in *2023 IEEE secur. priv. workshops*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 69–76. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SPW59333.2023.00012

[25] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *25th eur. symp. res. comput. secur.* Springer, 2020, pp. 480–501.

[26] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors," in *2008 IEEE Symp. Secur. Priv.* IEEE, 2008, pp. 81–95.

[27] J. Feng, H. Xu, S. Mannor, and S. Yan, "Robust logistic regression and classification," *Adv. neural inf. process. syst.*, vol. 27, 2014.

[28] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *2018 IEEE symp. secur. priv.* IEEE, 2018, pp. 19–35.

[29] J. Steinhardt, P. W. W. Koh, and P. S. Liang, "Certified defenses for data poisoning attacks," *Adv. neural inf. process. syst.*, vol. 30, 2017.

[30] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in *29th USENIX secur. symp.*, 2020, pp. 1605–1622.

[31] Y. Mao, X. Yuan, X. Zhao, and S. Zhong, "Romoa: Ro bust mo del a ggregation for the resistance of federated learning to model poisoning attacks," in *26th eur. symp. res. comput. secur.* Springer, 2021, pp. 476–496.

[32] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *arXiv prepr. arXiv:2012,13995*, 2020.

[33] Z. Ding, "Consensus output regulation of a class of heterogeneous nonlinear systems," *IEEE Trans. Autom. Control*, vol. 58, no. 10, pp. 2648–2653, 2013.

[34] T. Zhao and Z. Ding, "Distributed initialization-free cost-optimal charging control of plug-in electric vehicles for demand management," *IEEE Trans. Ind. Inform.*, vol. 13, no. 6, pp. 2791–2801, 2017.

[35] Y. Dong, T. Zhao, and Z. Ding, "Demand-side management using a distributed initialisation-free optimisation in a smart grid," *IET renew. power gener.*, vol. 13, no. 9, pp. 1533–1543, 2019.

[36] Y. Wang, L. Cheng, W. Ren, Z.-G. Hou, and M. Tan, "Seeking consensus in networks of linear agents: Communication noises and Markovian switching topologies," *IEEE Trans. Autom. Control*, vol. 60, no. 5, pp. 1374–1379, 2014.

[37] W. Ren and R. W. Beard, *Distributed Consensus in Multi-Vehicle Cooperative Control*. Springer, 2008, vol. 27.

[38] Z. Li, B. Liu, and Z. Ding, "Consensus-based cooperative algorithms for training over distributed data sets using stochastic gradients," *IEEE Trans. Neural Netw. Learn. Syst.*, 2021.

[39] A. Aly, M. Keller, E. Orsini, D. Rotaru, P. Scholl, N. P. Smart, and T. Wood, "SCALE and MAMBA documentation," 2018.

[40] J. Böhler and F. Kerschbaum, "Secure multi-party computation of differentially private heavy hitters," in *Proc. 2021 ACM SIGSAC conf. comput. commun. secur.*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2361–2377. [Online]. Available: https://doi.org/10.1145/3460120.3484557

[41] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," *Adv. neural inf. process. syst.*, vol. 20, 2007.

[42] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "d^2: Decentralized training over decentralized data," in *Int. Conf. Mach. Learn.*, 2018, pp. 4848–4856.

[43] D. Saad, "Online algorithms and stochastic approximations," *Online Learn.*, vol. 5, no. 3, p. 6, 1998.

[44] P. J. Bickel and K. A. Doksum, *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. CRC Press, 2015.

[45] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, 2009.

[46] Z. Deng, S. Liang, and Y. Hong, "Distributed continuous-time algorithms for resource allocation problems over weight-balanced digraphs," *IEEE trans. cybern.*, vol. 48, no. 11, pp. 3116–3125, 2017.

[47] Y. Zhu, W. Yu, G. Wen, G. Chen, and W. Ren, "Continuous-time distributed subgradient algorithm for convex optimization with general constraints," *IEEE Trans. Autom. Control*, vol. 64, no. 4, pp. 1694–1701, 2018.

[48] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Trans. Autom. control*, vol. 57, no. 3, pp. 592–606, 2011.

[49] W. Rudin *et al.*, *Principles of Mathematical Analysis*. McGraw-hill New York, 1976, vol. 3.

[50] H. S. Hippert, C. E. Pedreira, and R. C. Souza, "Neural networks for short-term load forecasting: A review and evaluation," *IEEE Trans. power syst.*, vol. 16, no. 1, pp. 44–55, 2001.

[51] C.-L. Liu, W.-H. Hsaio, and Y.-C. Tu, "Time series classification with multivariate convolutional neural network," *IEEE Trans. ind. electron.*, vol. 66, no. 6, pp. 4788–4797, 2018.

[52] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv prepr. arXiv:1609,03499*, 2016.

[53] I. Commission for Energy Regulation (CER), "CER smart metering project - gas customer behaviour Trial,2009-2010," 2012.

[54] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, "Three approaches for personalization with applications to federated learning," *arXiv prepr. arXiv:2002,10619*, 2020.

[55] T. Zhao, Z. Li, and Z. Ding, "Consensus-based distributed optimal energy management with less communication in a microgrid," *IEEE Trans, Ind. Inform.*, vol. 15, no. 6, pp. 3356–3367, 2018.