Received: 29 September 2025 | Revised: 30 March 2026 | Accepted: 9 July 2026 | Published online: 27 August 2026

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2026, Vol. 00(00) 1–14
DOI: 10.47852/bonviewJCCE62027792

BON VIEW PUBLISHING

Federated Learning with SVMs: Dynamic SGD for Efficient Hyperparameter Optimization

Deebakkarthi Chinnasame Rani[1], Gowtham Ramesh[1], Sountharrajan Sehar[2,*], Varun Aiyaswamy Kannan[1], Elambharathi Padmavathi Thangavel[1] and Bharath Kumar Kanapareddy[1]

[1] *Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Coimbatore, India*

[2] *Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Chennai, India*

Abstract: In the current digital landscape, organizations are actively looking for user data to make informed decisions at the edge. This raises a need for a solution that prioritizes user privacy while leveraging user's data. Federated learning (FL) emerges as a viable solution to address this issue. However, these methods are dominated by complex, resource-intensive neural networks (NN). This necessitates the development of an FL technique that protects user privacy while employing lightweight models. Support vector machines (SVMs) provide a lightweight alternative, but their traditional training techniques, such as quadratic programming and sequential minimal optimization, are inefficient and incompatible with FL. Existing stochastic gradient descent (SGD)-based SVM variants also depend on fixed or heuristic learning rates, which restrict convergence under non-IID data and client heterogeneity. In this paper, we propose a novel meta-learned dynamic learning rate controller for SGD-trained SVMs in federated settings. Unlike standard adaptive optimizers, the controller adapts learning rates over each epoch and clients. This process enables faster convergence and lower communication overhead to heterogeneous data. Empirical evaluations on benchmark datasets show that federated SVM framework provides effective results comparable to NN-based FL approaches while significantly less computational and communication overhead.

Keywords: support vector machines, dynamic hyperparameter tuning, privacy-preserving machine learning, federated learning, meta-learning

1. Introduction

With so much data stored within the users' hands, the organizations with large amounts of user data have a huge and some say unmatched advantage over its competitors in today's digital environment. Machine learning, with algorithms that always require vast amounts of data, is leading the way in technological innovation. In conventional machine learning workflows perform training in a centralized manner. Data are gathered at a central location, and model is trained on a single machine/computing cluster. But this may involve a lot of privacy problems if the information in the data is sensitive. This challenge has resulted in a need for methods that are able to exploit user data without revealing the data itself. With federated learning (FL), the role of the central server is to only share the model parameters being updated or gradients, but not sharing the raw data itself with the clients, and clients do not communicate with each other directly. Every client trains the model locally, and only sends the changes in the parameters, minimizing privacy concerns from a centralized training approach. With such a distributed paradigm, FL allows the knowledge to be aggregated from a number of different participants while protecting the confidentiality of the data.

In most FL systems neural network (NN) is the default model. However, many federate implementations in practice have hard constraints on resources. The edge devices—mobile phones, wearable sensors, and Internet of Things (IoT) nodes—typically have limited computing power, memory, network bandwidth, and operating power. For this purpose, it is costly or even impossible to deploy NN models with a large number of parameters or communication overheads [1]. Healthcare, financial, and industrial monitoring applications are largely characterized by structured or tabular data in many FL applications. For these types of applications, deep architectures can easily be outperformed by simpler topologies. In such settings, due to resource and/or lack of benefit from data-driven deep models, compact, communication efficient alternatives are encouraged. These classifiers, like support vector machines (SVMs), are known for their low model complexity and strong generalization capabilities, which are ideal for practical FL deployments, minimizing computational burdens on resource-constrained devices and fostering responsible resource usage and sustainable practices.

However, most of the SVM implementations, such as sklearn.svm. They are derived from LIBSVM library developed by

***Corresponding author:** Sountharrajan Sehar, Department of Computer Science and Engineering, Amrita Vishwa Vidyapeetham-Chennai, India. Email: s_sountharrajan@ch.amrita.edu

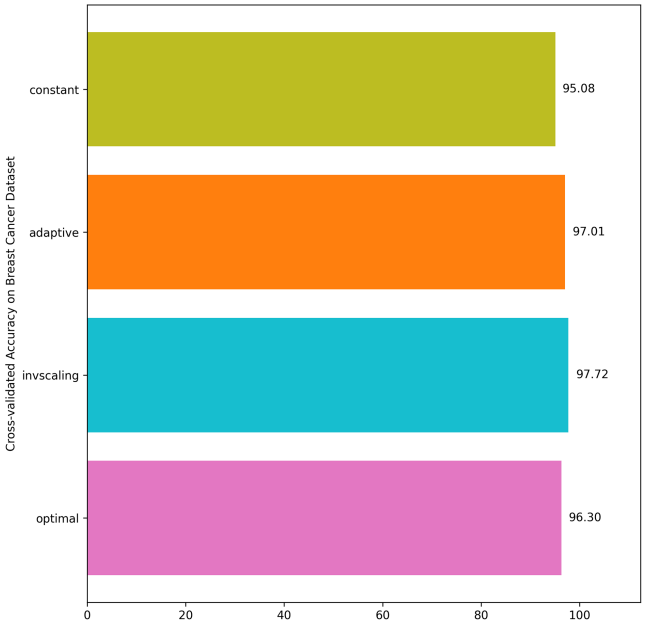
Chang and Lin [2]. One downside to this method is its O(N2) runtime. This is the reason that conventional libraries become hard to process the dataset of more than 10,000 samples [3]. Therefore, it is crucial to directly replace SVMs with NNs in an FL framework and optimize the process accordingly. Linear models trained with stochastic gradient descent (SGD), on the other hand, scale more efficiently and are more appropriate for use in resource-limited devices for FL applications. SGD initially begun as a method for optimizing linear regression, has grown into a popular optimization tool across machine learning, particularly in deep learning. The small size of datasets allows efficient training of the model on a scale of the SDG's strategy to incrementally update parameters on small sets of data. But the hyperparameters have to be tuned carefully, in particular the learning rate. This parameter directly affects the convergence speed and stability.

Apart from the experiments conducted with SVM, we tried out some learning rate adjustment techniques on the Wisconsin Breast Cancer dataset and depicted the results in Figure 1. All of the adaptive learning rate schedules outperformed the optimal fixed rate discovered using SMAC3, a Bayesian optimization tool [4]. The observed, although small, performance improvement is enough to demonstrate the clear gain of dynamic learning rate strategies. From these results, an SVM model trained by using SGD with dynamic learning rates especially for FL is proposed. This will enable high-level learning on poor computation devices with efficient and privacy-preserving methods, which will help build strong digital infrastructure.

The following are the summary of the primary contributions:

1) Propose a novel meta-learned controller for dynamically optimizing learning rate of SGD in support vector machines (SVMs), specifically designed for FL scenarios.
2) Develop a federated SVM framework that integrates the proposed dynamic learning rate mechanism, enabling efficient and scalable training for resource-constrained devices.
3) Demonstrate improved convergence and robustness of the proposed method under non-IID data distributions. That reduces communication overhead compared to conventional NN-based FL approaches.

Figure 1
Comparison of various learning rate settings



The structure of the paper is as follows: Section 2 gives an overview of related works. Section 3 outlines the system architecture, and Sections 4 and 5 describe the client node design and coordinator node design specifications, respectively. The results and analysis are shown in Section 6, and Section 7 provides a discussion of the limitations and future research areas of the proposed work. Finally, the paper concludes in Section 8.

2. Related Work

This section focuses on some of the most salient research works in the major research focuses of dynamic algorithm configuration (DAC) and FL with the aim of building adaptive and resource-efficient models for decentralized learning applications.

2.1. Dynamic algorithm configuration

Biedenkapp et al. [5] presented the idea of dynamic algorithm configuration (DAC), formalizing the problem of dynamically tuning the parameters of an algorithm over sets of multiple instances. They were able to model this problem as a Markov decision process (MDP) to help evaluate the various DAC approaches. The latter, Eimer et al. [6] extended this by providing a set of standardized benchmarks and discussing the common problem of the nonreproducibility in DAC research. In addition to the benchmarks, the work also proposed a standard template for the development of DAC algorithms. The template is in principle designed for reinforcement learning systems but can be used for other methodological contexts.

Daniel et al. [7] laid a theoretical groundwork for the development of adaptive learning rate controllers. The authors suggested to cut out certain feature at each step of SGD and based it to calculate an updated learning rate. The controller was trained and tested with Modified National Institute of Standards and Technology (MNIST). Interestingly, the use of the same controller on the Canadian Institute for Advanced Research (CIFAR-10) dataset resulted in even better accuracy scores, thus showing the meta-learning capability of the approach.

The extended approach was proposed by Adriaensen et al. [8] who used the framework developed by Eimer et al. [6] and trained using the Sequential model-based Algorithm Configuration (SMAC). Thérien et al. [9] proposed a learned optimizer framework, called µLO, by incorporating maximal update parametrization (µP) for enhancing meta-generalization on unseen tasks. Their approach allows them to achieve high performance on very large, very deep, and very long training networks, whereas standard learned and hand-designed optimizers fall short. It is still constrained by its scope of evaluation in the context of MLP tasks only and nonevaluated in different architectures.

2.2. Federated learning

Yurdem et al. [10] summarized FL, which includes optimization methods, system design, and practical applications. The paper identifies privacy, scalability, and data heterogeneity as some of the challenges. According to Zhang et al. [11], there is a gap in the current research of FL. Each iteration, the authors recommend reducing the size of data by using model compression. They also point out that there is ample room for further research in determining the optimal tradeoff between the two costs—communication and computational cost.

Uddin et al. [12] provide a comprehensive systematic literature review (SLR) of robust FL, following the PRISMA framework, analyzing 244 papers. The paper outlines 8 general

themes for robustness techniques: objective regularization, optimization technique changes, differential privacy, selection, and aggregation strategies for clients. The work pinpoints research gaps and challenges including noisy updates, label flipping, adversarial attacks, and non-IID data. Future work, such as hybrid, ensemble, and adaptive mechanisms are also discussed. The study provides a systematic framework to design secure and resilient FL systems.

The experiments by Li et al. [13] were the first practical experiments on the convergence of FedAvg with non-iid datasets, a theme that was previously limited mainly to theoretical derivations. Numerical results of experiment conducted are presented in the literature. Furthermore, the research shows that the larger K is, the faster the convergence will be. Li et al. [14] advance upon the work of Li et al. [13] introducing FedProx a generalization and reparameterization of FedAvg. The distinguishing feature of FedProx from FedAvg is to combine partial solutions provided by stragglers and include a Proximal term. This is related to statistical heterogeneity and obviates the need for any manual input of the number of local epochs, with local updates now being restricted to be equal to that of the original global model. Karimireddy et al. [15] outlined some of the difficulties that arise in the context of FL, identifying Client Drift as their main problem. In order to solve this a novel algorithm called Controlled Averaging for Federated Learning (SCAFFOLD) was introduced. The authors pointed out the potential difficulties associated with FL and identified client drift as the major difficulty. Wang et al. [16] proposed a method that improves significantly from FedAvg and FedProx—Federated Normalized Averaging (FedNova). This strategy requires normalized averaging to achieve fast convergence and compensate for inconsistencies between the objectives. Reddi et al. [17] showed the integration of adaptive optimizers into FL. They proposed federated versions of adaptive optimizers such as Adagrad, Adam, and Yogi. It was designed to restore the convergence that has been suboptimal in FedAveraging. They show the great advantage of adaptive optimizers over nonadaptive optimizers in their empirical results.

FedDave [18] tackles the issue of heterogeneity in FL by using server-side aggregation method to explicitly encourage client divergence to break out of local minima. FedDive enhances robustness in high non-IID data distributions by amplifying updates that are at odds with a momentum-guided consensus. Instead, we aim at client-side optimization, by which a meta-learned controller dynamically learns the learning rates during local training. These are free strategies, in the sense that they tackle the heterogeneity at different stages of the FL pipeline.

Khan et al. [19] introduced an FL framework named FedEmo, which identifies emotions from handwriting features. Using a Wacom-like device, they extract numerous features from a patient's handwriting. They are using a hybrid cloud-edge approach that takes heavier workloads to the cloud and lighter workloads to the edge. The various clinical institutions can use FL to develop a more effective model. This works well, however, it uses deep models that are very expensive and can restrict use of the model in resource-constrained devices.

Jiang et al. [20] suggest a fuzzy ensemble-based FL system for EEG emotion identification by IoMT. The authors combine a fuzzy ranking mechanism with a variety of different deep learning models (TCN, LSTM, and GRU). To decrease the staleness of the gradients, an asynchronous dropout strategy is added. Several experiments are performed on different EEG data to verify its accuracy and robustness. But the combined network of deep models adds to the complexity of computation and communication making it problematic for edge devices with limited power capabilities.

Apart from emotion recognition, Bai et al. [21] investigate FL for network intrusion detection systems in IoT scenarios. Their results indicate that classical machine learning models, especially random forests, can reach similar performance as centralized training while keeping data privacy and limiting the communication cost. While designed for IoT security, this paper shows that lightweight models are a good way forward in resource-constrained federated environments and suggests that efficient alternatives to deep learning are worthy of further study. To compare the existing FL optimization methods with the proposed method as well as the applicability of the learning rate adaptation, the results are tabulated in Table 1.

Table 1
Comparison of existing FL optimization methods and their applicability to learning rate adaptation

Authors	Dynamic Learning Rate Adjustment	Integration of Support Vector Machines (SVM) in Federated Learning	Meta-Learning Step Size Controller	Improved Theoretical Convergence
Li et al. [13]	Static	LSTM	Learning rate found by HPO on 1 epoch	Yes, through proximity term
Karimireddy et al. [15]	Static	NN	No	Yes, through control variates
Khan et al. [19]	Static	Transformer-based model	No	No explicit convergence analysis
Jiang et al. [20]	Static	TCN, LSTM, GRU (ensemble)	No	No explicit convergence guarantees
Bai et al. [21]	Static	Random Forest	No	No explicit convergence guarantees
Wang et al. [16]	Yes. Exponential decay	LR, DNN	No	Yes, through normalized averaging

(*Continued*)

Table 1
(*Continued*)

Authors	Dynamic Learning Rate Adjustment	Integration of Support Vector Machines (SVM) in Federated Learning	Meta-Learning Step Size Controller	Improved Theoretical Convergence
Reddi et al. [17]	Static	CNN, LSTM	No, learning rate found through grid search	Yes, through adaptive optimizers
Rateria et al. [18]	No (focuses on aggregation, not LR)	Primarily deep models (CNN)	No, divergence-rewarded aggregation	Partial (improves robustness to non-IID via divergence-aware aggregation)
Proposed work	Yes	Yes	Yes, learning rate controlled by Meta-learned Controller	Yes, through dynamic learning rates

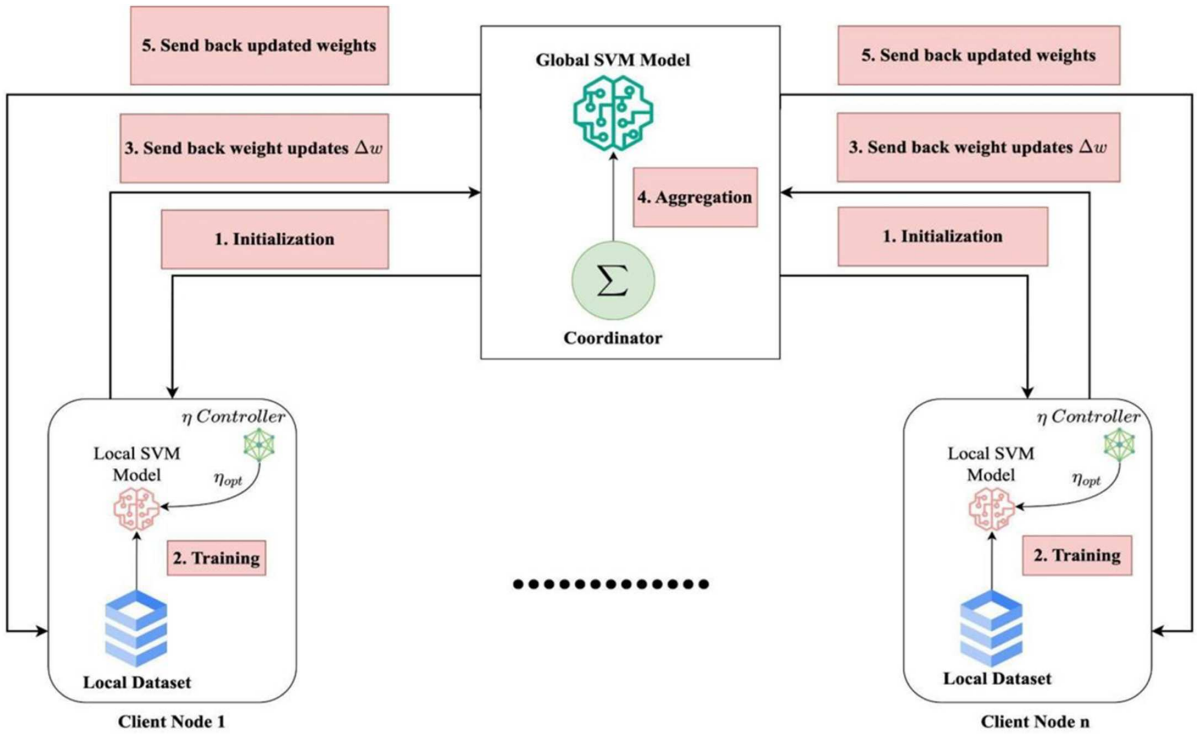
3. System Design

The overview of the proposed system is shown in Figure 2. The system consists of a number of client nodes and a coordinator node. Local SVM models are trained with global weights sent by coordinator. The training is started by the coordinator node, which will generate a random weight vector and send it to all client nodes. Each client node has its own controller to help it with its FL role. The client nodes run a set of SGD steps with a learning rate dynamically updated by the controller for each batch. After the local training is done, the client nodes send the updated weights to the coordinator node. The coordinator keeps the list of the client nodes that respond with updated weights and starts the process of aggregation once it receives data from the number of nodes it needs to get the update. In the aggregation, a weight is assigned to each client based on the number of data samples in their local training algorithm, and the contribution of each client is scaled by that weight. One training round consists of the mentioned sequence of client sampling, local training, and weighted aggregation process, which iterates until the convergence criteria are met.

4. Client Node

This section describes the internal components and how they work of a single client node. The client node is the most complex component of the system and is a decentralized federation of other clients that work to train the global model. It has three basic

Figure 2
Adaptive FL coordinator–client architecture



parts: the FL Client Agent, the Step-Size Controller, and the SGD module. The FL Client Agent is the one that handles the communication between the client node and coordinator node. Once the agent receives the weights from a coordinator, local training is started. After local training, the weight updates will be sent back to the coordinator via the agent.

4.1. Step-size controller

The step-size controllers are used to automatically tune the learning rate for every epoch. It aims to alter the learning rate without relying on the tasks and previous knowledge of the learning problem. Each learning algorithm, like SGD, returns a vector of features to the controller, and the controller uses Equation (1) to determine the optimum learning rate.

$$\eta = g\left(\hat{\phi}, \theta\right) = exp\ \left(\ \hat{\phi}.\theta\ \right) \tag{1}$$

where

1) η is the learning rate.
2) $\hat{\phi}$ is the vector of features received from training algorithms such as SGD.
3) θ is the vector of learned model parameters.

Equation (1) shows the calculation of the learning rate. The learning rate is determined as the exponential dot product of the two vectors $\hat{\phi}$ and θ. The vector $\hat{\phi}$ consists of four features: the discounted average of predictive change in function value ($\widehat{\phi_1}$), the uncertainty estimate of predictive change in function value ($\widehat{\phi_2}$), the discounted average of disagreement in function value ($\widehat{\phi_3}$), and the Uncertainty estimate of disagreement in function value ($\widehat{\phi_4}$). The vector θ consists of corresponding four feature values generated using a Bayesian optimizer SMAC. This feature vector is further modified to enhance the learning process, thereby improving the decision-making of the controllers. For instance, in the RMSProp training method, the vector is augmented with an additional feature representing a bias term with a constant value of 1, resulting in the final vector $(1, \widehat{\phi_1}, \widehat{\phi_2}, \widehat{\phi_3}, \widehat{\phi_4})$. Conversely, for SGD-momentum, the feature vector is modified to $(1, \widehat{\phi_1} - \widehat{\phi_2}, \widehat{\phi_3} + \widehat{\phi_4})$ with the aim of reducing the search space. These findings, along with empirical evidence, are derived from the work of Daniel et al. [7]. These modifications significantly impact the training process, as only three parameters need to be learned for momentum, whereas five parameters are required for RMSProp.

4.2. Stochastic gradient descent

The proposed method is based on SGD to minimize the hinge loss of SVM to obtain an optimal model that can be used with FL. In this process, a meta-trained step-size controller is used that dynamically adjusts the learning rate for each epoch as opposed to the static one for conventional SGD.

SGD control flow is summarized in Figure 3 and discusses how it relates to the step-size controller. There are two outer loops in this process: An outer loop that mainly processes a mini batch of the dataset in each of its steps, and an outer loop that has a number of steps that depends on the size of the dataset and is denoted by the constant E. Inside the inner loop, the loss and the gradient for each and every sample from the mini batch are calculated using the right functions. The results of these computations are used in extracting four key features from the samples, and the computation of these features is discussed in this section.

4.2.1. Formal feature representation

In the machine learning world, we know that quality of data equals quality of resulting controller. Likewise, the working of the controller is greatly dependent on the quality of the features provided to it. The features should indicate the current training state, and they should be general in such a way that they can be used for other datasets and model architectures.

But there are some limitations on in extraction of features. The feature extraction should not be more expensive than the training algorithm. Based on these considerations, we have used the following features: predictive change discounted average, predictive change uncertainty measure, disagreement discounted average, and disagreement uncertainty measure. All these properties are easy to compute, and all of them are just "side effects" of the gradients computed in the SGD loop. These features were mentioned by Daniel et al. [7]. These features have been verified by their use in Adriaensen et al. [8] and Xiong et al. [22].

1) Feature 1: predictive change in function value

We define the predictive change in function value as shown in Equation (2).

$$\tilde{f}_i(d_i, \Delta w + w) = f(d_i, w) + \nabla f_i^T \Delta w \tag{2}$$

This is the first-order Taylor series expansion to approximate f. Then,

$$\Delta\tilde{f}_i = \tilde{f}_i - f_i$$

Substituting the value of $\tilde{f}_i$, we get

$$\Delta\tilde{f}_i = f(d_i, w)\ +\ \nabla f_i^T \Delta w - f_i$$

$$\Delta\tilde{f}_i\ =\ \nabla f_i^T \Delta w$$

We define predictive change in function value as

$$\varphi_1 = log\ \left(Var\left(\Delta\tilde{f}i\right)\right)$$

$$\varphi_1\ =\ log\ \left(Var\left(\nabla f_i^T \Delta w\right)\right)$$

But we know that Δw is given by $-\eta\nabla F$ in the case of SGD.

$$\varphi_1\ =\ log\ \left(Var\left(-\eta\nabla f_i^T \nabla F\right)\right)$$

The feature aims to gauge how well the individual gradients agree with each other.

2) Feature 2: disagreement in function values

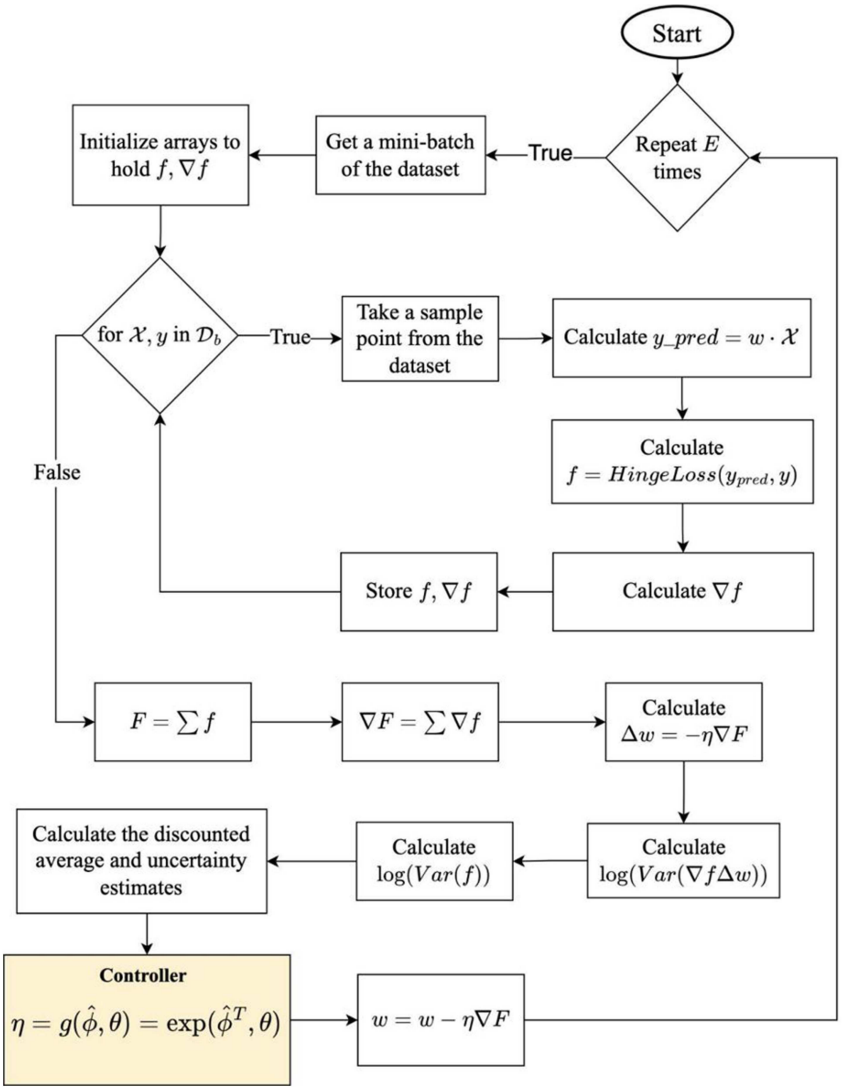
We take the variance in the current function values as our disagreement in function values as shown in Equation (3).

$$\varphi_2\ =\ log\ \left(Var\left(\ f(d_i, w)\right)\right) \tag{3}$$

The features presented here are defined for the whole dataset. But, in practice, we do not use full-batch training, we only use mini-batch training. This causes the features to have considerable noise. Few changes have to be made to the features to mitigate this noise.

3) Feature 3: discounted averaging

Figure 3
SGD control flow



The first modification is called discounted averaging. For each feature, we maintain a running average of its observed values, as shown in Equation (4).

$$\widehat{\phi_i} \leftarrow \gamma \widehat{\phi_i} + (1 - \gamma)\phi_i \tag{4}$$

1) $\widehat{\phi_i}$ denotes the running average of the i^{th} feature.
2) ϕ_i represents the observed value of the i^{th} feature in the current iteration.
3) γ is a discount factor, where $0 < \gamma < 1$. It controls the contribution of the current observation to the running average.

This approach helps mitigate the impact of outliers and acts as a form of memory in the learning process.

4) Feature 4: uncertainty estimate

Although a discounted averaging effect helps to reduce the noise, it would be nice to have an explicit estimate of the noise at each step. This estimate can be computed as the squared difference between the observed feature and the running average of the feature, $(\varphi_i - \varphi_i)^2$. But we still need to take average of this estimate discounted. Hence, the final equation is displayed in Equation (5).

$$\widehat{\varphi_{K+i}} \leftarrow \gamma \widehat{\varphi_{k+1}} + (1 - \gamma)(\varphi_i - \widehat{\varphi_i})^2 \tag{5}$$

where k is the number of "base" features. A short summary of these features, along with their intuitive meaning and purpose in the learning rate controller is given in Table 2. The dimensionality of the feature vector depends on the type of stochastic gradient descent used and is simply additive operations.

4.3. Controller training

The mathematical formulation of the Controller is shown in Equation (6).

$$w^* = arg\ min_w\ F(D, w) \tag{6}$$

where w is the weight vector and $D = d_1, d_2, \ldots, d_n$ is the set of training samples. The function F is defined as shown in Equation (7).

$$F(D, w) = \frac{1}{|D|} \sum_{k=1}^{|D|} f(d_i, w) \tag{7}$$

Here, $f(d_i, w)$ represents the loss associated with the i^{th} sample and weight vector w.F can be interpreted as the average loss.

Table 2
Feature summary for the learning rate controller

Feature	Description	Significance
Predictive change in function value	Variability in the predicted change of the objective value based on individual gradients	Reflects the level of agreement among individual gradients; lower variability indicates more consistent descent directions
Disagreement in function values	Variance in the current objective values across samples	Captures data heterogeneity and noise present during local training
Discounted averaging	Exponential running average of observed feature values	Reduces the impact of noise and outliers while providing temporal memory
Uncertainty estimate	Smoothed estimate of the deviation between observed features and their running averages	Provides an explicit measure of noise associated with each feature

In other words, the function F can be expressed as shown in Equation (8).

$$F(D, w) = E_{d\sim D}[f(d, w)] \tag{8}$$

Here, $d \sim D$ indicates that d is drawn randomly from the set of training samples D, and (d, w) represents the loss associated with sample d and weight vector w. Therefore, F can be interpreted as the expectation of the loss over the training samples.
For training SVM, Hinge loss is being used. It is defined as shown in Equation (9).

$$f(d, w) = (0, 1 - yi(w \cdot xi + b)) \tag{9}$$

where d represents the sample, w denotes the weight vector, x_i is the feature vector of sample d, b is the bias term, and y_i is the true label of sample d.

The hinge loss function penalizes misclassifications by measuring the difference between the predicted score $w \cdot x_i + b$ and the true label y_i. If the predicted score does not exceed the margin of 1, the loss is 0; otherwise, it increases linearly with the magnitude of the margin violation. Like any optimization problem, we must have some function that gives us the value of the update at each iteration. This update value may be the product of many optimization methods, such as SGD, Adagrad, RMSProp, Adam, Momentum, and more. We use SGD and its variants in our work. For the vanilla SGD, its mathematical model is mathematically represented as in Equation (10).

$$\Delta w = -\eta \nabla F \tag{10}$$

where Δw represents the update value for the weight vector w, η is the learning rate, ∇F represents the gradient of the loss function F with respect to the weight vector w. Here, F is the average loss over all samples. ∇F is the average gradient over all samples, and it is defined as shown in Equation (11).

$$\nabla F = \frac{1}{|D|} \sum_{k=1}^{|D|} \nabla fi \tag{11}$$

$$\text{where, } \nabla fi = \frac{\partial f(di, w)}{\partial w}$$

Daniel et al. [7] defined the goal as to learn a policy π that adapts η based on features φ as shown in Equation (12).

$$\pi^*(\varphi) = \pi \int \int p(\varphi)\, \pi(\varphi)\, r(\eta, \varphi)\, d\varphi d\eta \tag{12}$$

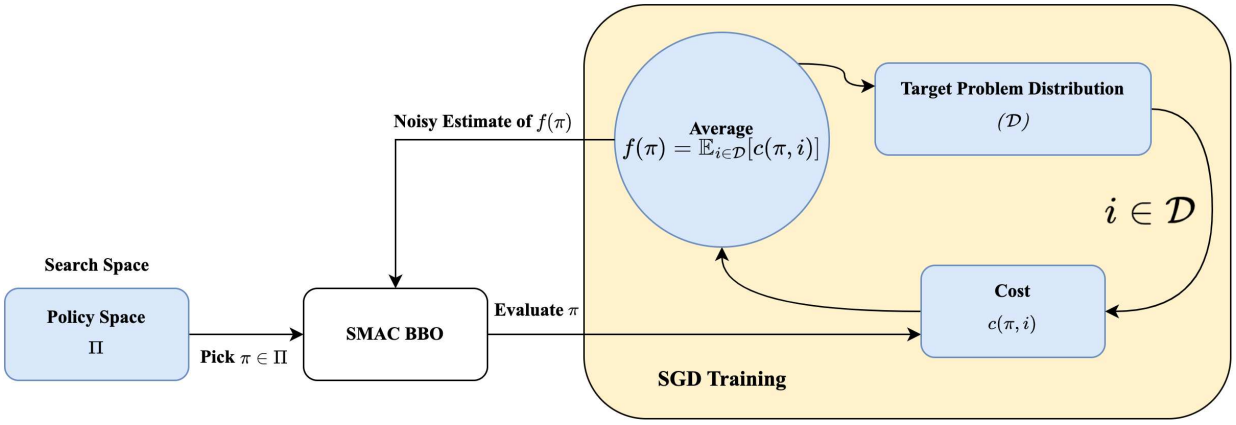
1) $\pi^*(\eta|\varphi)$ denotes the optimal policy that adapts the parameter η based on the features φ.
2) The goal is to find π that maximizes the expected reward, which is represented by the integral.
3) $p(\varphi)$ represents the distribution of features.
4) $\pi(\varphi)$ represents the policy function.
5) $r(\eta, \varphi)$ is the reward function that evaluates the expected reward for a given pair of η and φ

Meta-learned controller is trained offline using SMAC3, a Bayesian optimization framework, over multiple mini training episodes from the training dataset. In each episode, a candidate controller is tested by computing the decrease in training loss over a predetermined number of optimization steps. The controller parameters are optimized to best achieve this improvement for a variety of learning instances, which in turn helps to promote generalization rather than memorization of a particular learning-rate schedule. The training procedure along with the explicitly stated feature set and objective function allows for the reproducibility of the proposed approach.

Several small training episodes are sampled from the training set, and the meta-learned controller is trained offline through the SMAC3, a Bayesian optimization framework. A candidate controller is tested in each episode through a specified number of optimization steps, based on the loss reduction in the training, using those steps. The controller parameters are chosen to maximize this improvement on a variety of training cases, favoring generalization over memorization of a preset learning rate schedule. The training method together with the explicitly mentioned feature set and objective function enables the reproduction of the proposed method.

Figure 4 describes the general flow for training a controller, where Π represents the policy space, encompassing all valid parameters for the policy function $g\left(\hat{\phi}, \theta\right)$. Here, $\hat{\phi}$ denotes the features extracted in the current step, and θ represents the parameters. Specifically, θ lies in R^3 for momentum and R^5 for RMSProp, essentially constituting a vector of real numbers. π is the candidate policy chosen to be evaluated, more specifically the parameters of the policy. $c(\pi, i)$ is the cost function. This cost function is not the one used to quantify classification loss. This cost function quantifies the cost of this training episode. More specifically, it measures how well our learning rate schedule aids convergence. It is defined as $\left(\frac{1}{k-1} log\left(\frac{E_k}{E_1}\right), 0\right)$ where k is the number of optimization steps and E_x is the batch training loss after x optimization steps. This ratio illustrates the goodness of the training, not the model. On the other hand, when the value of E_k is significantly lesser than E_1 indicates that the model has

Figure 4
SMAC flow diagram



improved significantly compared to the initial state. However, this does not mean that the model's prediction accuracy is improved but indicates a substantial drop in the loss after k steps. D is the dataset used for training, where each member of DDD is a tuple of the form $\langle D,k \rangle$. This is typically a subset of a larger dataset. In our work, each D is a random subset of the MNIST dataset, and k denotes the number of optimization steps. $f(\pi)$ is the overall cost and is defined as the expected value of the cost function $c\ f(\pi) = E_{i \sim D}\,[c\,(\pi, i)]$.

S SMAC executes multiple runs, in each run, involving the selection of a candidate policy and evaluation of its cost. This is repeated for a certain number of runs and the best solution is returned. For every run, SMAC chooses 3 or 5 real numbers following a certain policy, and then it draws a problem instance from the data set (which is a subset of the data plus a fixed number of execution steps). The policy is used to learn the model on this instance, the cost is saved and SMAC moves to the next iteration. Most importantly, in each iteration of SMAC, the entire model training procedure is followed and not just computing a single training sample. This design enables learning to learn (meta-learning). Each time a problem occurs, it has its own unique set of data and amount of training, so the policy must generalize appropriately across various contexts rather than just rote learn a single schedule.

1) Implementation details and hyperparameters

The architecture of the meta-learned controller along with the associated meta-training hyperparameters is based on the DAC setup reported in Adriaensen et al. [8] that uses the feature design from Daniel et al. [7]. In particular, the controller is a linear policy over the extracted feature vector, having 3 parameters for SGD with momentum and 5 for RMSProp, depending on the underlying SGD variant.

5. Coordinator Node

From the system-level workflow described in Section 3, this section will concentrate on the responsibilities at the coordinator node. The Server Agent of the coordinator node sets up a global model with random weights following a uniform distribution. According to empirical results from McMahan et al. [23], if common weights are used, then better performance than if each client is initialized with its own weights. On the basis of this finding, the Server Agent module takes the same random weight for all clients. A small number of k clients are then sampled to learn in each training round. Oversampling is conducted to compensate for the possibility of client drop-outs and/or slow responsiveness to enhance the convergence, as illustrated in Figure 5. The server assigns weights to a subset of the k clients and waits for a subset of the k clients to reply. The server receives new weights from involved clients and merges them to update the global model. When new

Figure 5
Federated learning (server) flow diagram

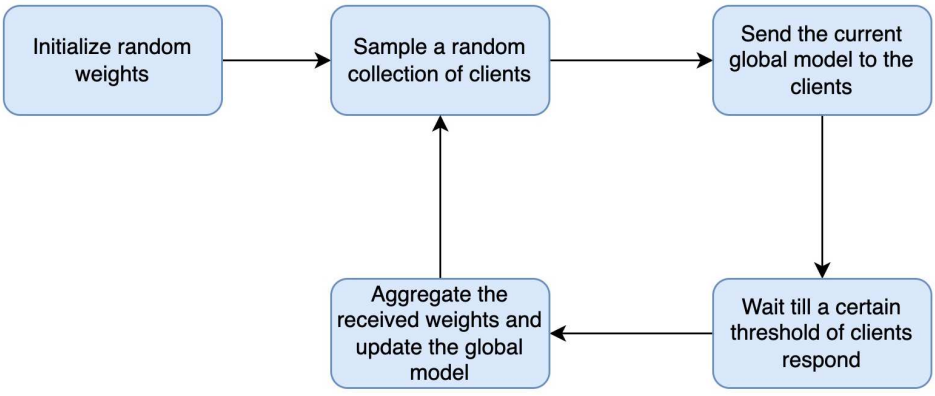
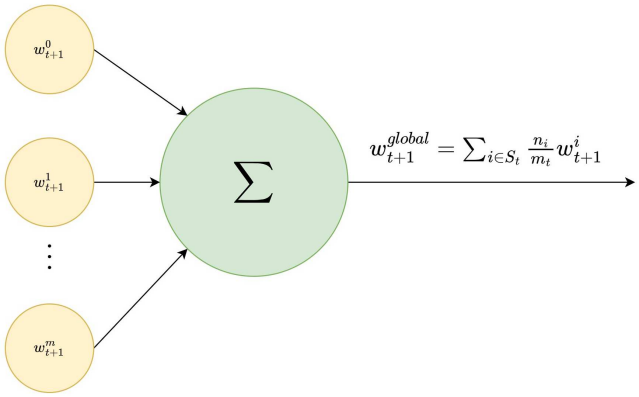


Figure 6
Aggregation process



weights are received by the server from the participating clients, the server combines the new weights to update the global model.

The aggregation process is one of the key steps in FL, represented in Figure 6. In this case, the simplest and most direct way in the aggregation process has been used. In this, the weight in the next round is computed based on a weighted average of all the weight received from the previous round. The average is computed on a per-client basis, based on the number of training samples available to each client. In Figure 6, S_t denotes the set of clients chosen for the t^{th} round of iteration, and n_k signifies the quantity of training samples within client $k \cdot m_t = \sum_{k \in S_t} n_k$ represents the aggregate count of training samples during round t and w^i_t denotes the weight conveyed by client i during the t^{th} round.

6. Results and Analysis

6.1. Dataset description

The experiments are performed in the early stage on The Wisconsin Breast Cancer Dataset [24] with the help of scikit-learn [25]. This dataset is selected because of its small size and real world nature. Although it is an enormous size, it's rich in features, so we were able to collect some concrete benefits. This was used for comparing all the SVMlike classifiers provided by scikit-learn. Also, all of the learning rate strategies provided by SGDClassifier were tested. These confirmed the importance of using SGD for training SVMs and further showed the importance of adaptive learning rate in SGD.

The MNIST dataset was used as the benchmark to test the CNNs studied. Moreover, the ability of the controller to learn from the CIFAR-10 dataset was tested. CIFAR-10 is a benchmark dataset used for computer vision tasks. It is more difficult and representative to test the effectiveness of the proposed method because it provides more data in 3 colors than the MNIST does.

Table 3 shows a comparison of the data in MNIST, and Table 4 shows a comparison of the data in CIFAR-10. We test the controller to see if it can be transferred to CIFAR-10. Selected datasets cover a range of learning scenarios, from small-scale tabular data, to centralized image classification, to federated non-IID scenarios featuring varying levels of data heterogeneity. This diversity allows us to evaluate the proposed controller across different data modalities and deployment settings.

Table 3
Dataset description of CIFAR-10

Attribute	Description
Image size	32 × 32 pixels
Total images	60,000
Classes	10
Train images	50,000
Test images	10,000
Image channels	RGB (3 channels)
Pixel value range	[0, 255]

Table 4
Dataset description of MNIST

Attribute	Description
Image size	28 × 28 pixels
Color depth	Grayscale
Total users	3383
Total labels	10
Training examples	341,873
Testing examples	40,832
Data range	[0, 1]

6.2. Performance analysis

We perform an evaluation of the proposed dynamic learning rate controller against several baselines of static learning rates and widely used optimization methods based on SGD both in the centralized and federated scenarios. The same hyper parameters as reported by Adriaensen et al. [8] are used. The comparison is limited to static learning-rate baselines and commonly used SGD-based optimizers and is representative of many FL frameworks which primarily support SGD and its variants.

6.2.1. Centralized setting

First, we investigate whether the controller is effective for SVM in centralized setting. In Figure 7, we compare the training loss with the static learning rates with that of the learned DAC policy. We test several values of the learning rate to illustrate its effect. The training was done for 50 epochs and batch size of 128 for all runs. The MNIST dataset was used, and images were rescaled in the interval [0, 1] (see Table 4).

The training loss is plotted against epochs for various static learning rates and for the learning rate dynamically controlled by the controller for this experiment as shown in Figure 7. The best static learning rate is evident as 0.01, whereas the other rates are too large for the particular dataset. We notice that the validation loss observed after only a few epochs is the smallest with DAC while with the static learning rate of 0.01, about 30 epochs were needed to achieve this performance.

Table 5 shows the results of the final test accuracy of the different methods. DAC is slightly better than the static methods. This finding agrees with that in Yurdem et al. [10] and Adriaensen et al. [8], where the final testing loss of the static methods was slightly worse than that of the proposed method.

Table 5
Testing loss and accuracy of static and dynamic learning rate

Learning rate	Test loss	Test accuracy
0.01	0.30	0.91
0.001	0.48	0.88
0.0001	1.2	0.83
DAC	0.27	0.93

6.2.2. Federated setting

The performance of our meta-controller is tested in a federated setting. We first performed system tests on the Federated Extended MNIST (FEMNIST) dataset, which is a federated version of EMNIST. The shuffled and unsorted images of the normal dataset are arranged in a random order, whereas in the FEMNIST dataset this shuffling is done according to the different authors. This partition assures that photographs in each subset have the same author, resulting in a non-IID (non-independent and identically distributed) situation. This increases the diversity and realism of the training environment as each section has handwriting from a different person.

All the images in the dataset are grayscale images of 28 × 28 pixels. In particular, the dataset contains 3383 users, and data are classified in 10 different labels. The dataset consists of 341,873 training examples and 40,832 test examples for training and evaluation. Note that all of the pixel values were normalized to the range [0, 1].

FL differs from distributed learning in that the datasets are not iid. The non-IID property of FEMNIST dataset is illustrated in Figure 8. Every occurrence of the digit "2" in the picture is a representation of the average hand-writing style of all the occurrences of the "2" hand-written by a specific user. Variations exist within each client's data and are representative of the differences between the handwriting of individual people. They can be evident in various ways, including the form of the loops or the overall direction of the numbers. The differences could be used to show the variety of writing from various writers and emphasize the importance of inter-client differences in the data.

Another type of heterogeneity is in the variation of the label distribution across clients. This is more noticeable in the other datasets but also exists in this dataset. Some digits are more likely to be written by some people than others and as such there is

Figure 7
Comparison of static learning rates and DAC policy

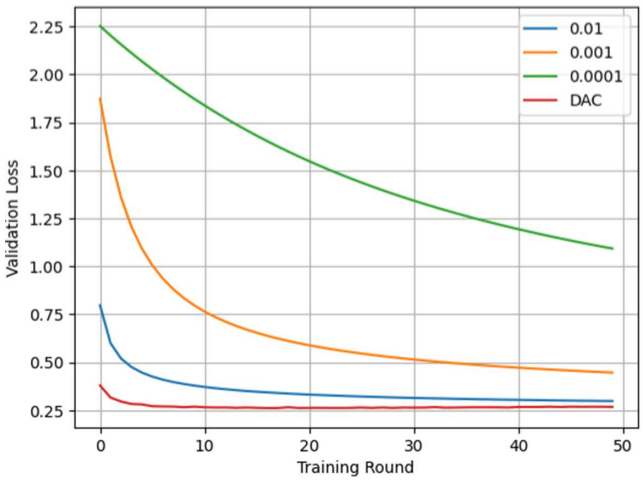
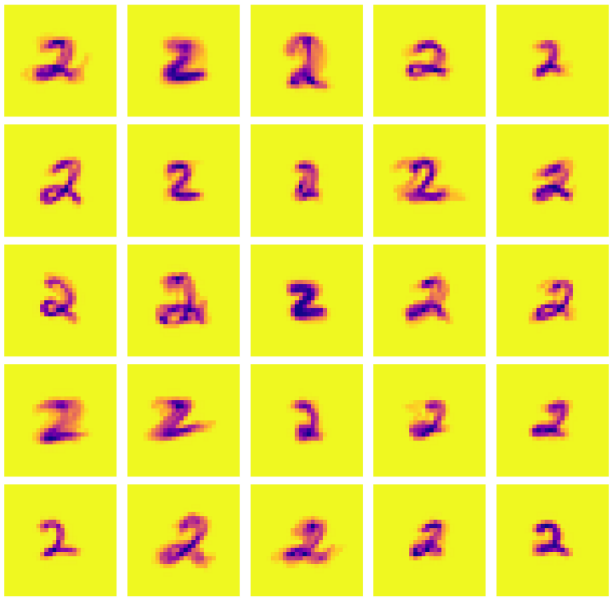


Figure 8
Mean shape of the label *2* in different client



an inherent bias to certain digits. However, it is worth noting here that this bias is not universal, it does not apply to everyone, but varies from person to person. This is represented in Figure 9.

6.2.3. Federated learning results

This section provides examples of how dynamic learning rates work. This was implemented for 1000 communication rounds for each configuration. During each round, 10 clients were randomly sampled, and each client applied 20 local epochs for each global epoch, and the batch size was set as 10.

Figures 10 and 11 show the training results. We note that DAC achieves the best results, especially regarding the convergence speed. It can be seen that the static learning rate 0.01 performs similarly, but the real strength of DAC is that it is robust to the initial learning rate. It is easy to find the best static learning rate only after the experiment is complete and multiple runs are usually necessary. DAC, on the other hand, adapts and learns an effective learning rate automatically in training.

The testing accuracy results after each training round can be seen in Figure 12. We note that FL also results in a slight drop in performance from the centralized case. The best overall performance is given by DAC, and the static learning rate of 0.001 is able to catch up after about 700 communication rounds.

6.2.4. Ablation study

We studied the effect of the number of clients in each round. Seeing the impact of changing the size of the set of participants is demonstrated in Figures 13 and 14. The results are comparable in the end, but the smaller sets of participants are less stable than larger sets. This is in line with McMahan et al. [23] who also found that results of fewer clients per round declined after a certain number.

6.3. Generalizability testing

We then examine the generalization of the meta-controller to a new set of data that are quite different. In the same way that

Figure 9
Label count distribution of clients

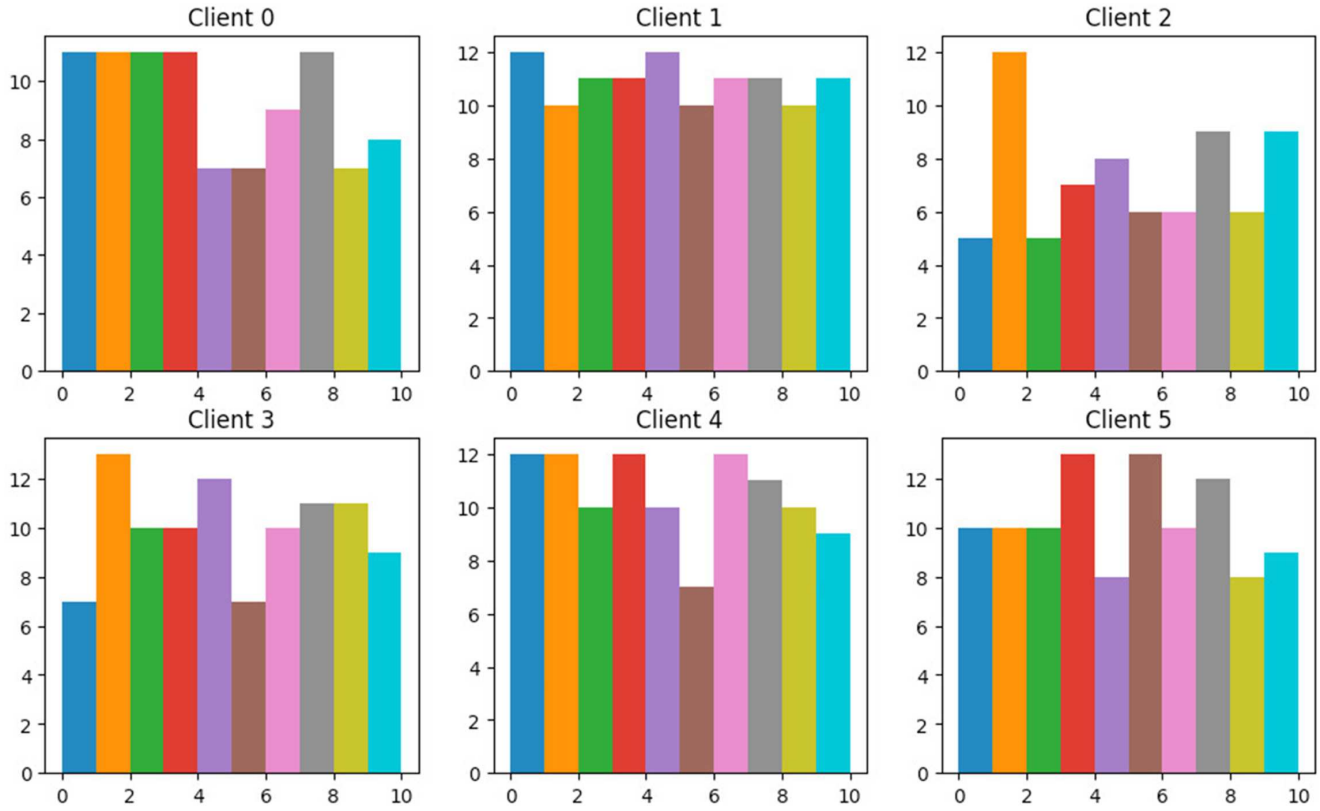


Figure 10
Effect of learning rate on training loss

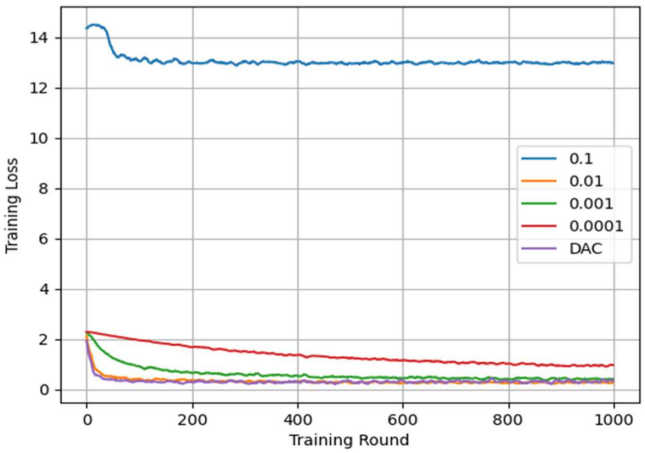
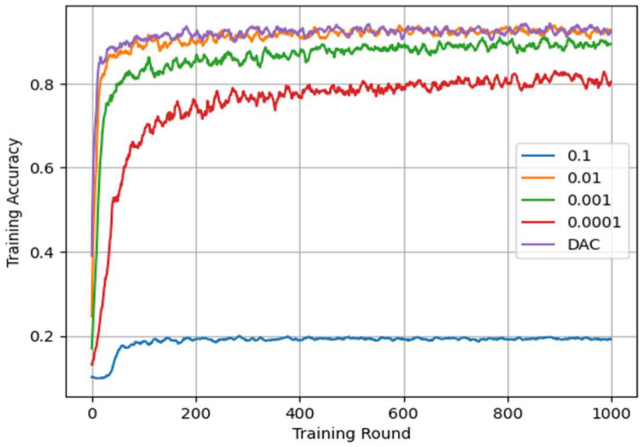


Figure 11
Effect of learning rate on training accuracy



MNIST has federated equivalent FEMNIST that naturally has the data samples split into different authors, this simple split does not exist in CIFAR-10. Thus, CIFAR-10 was partitioned using Dirichlet partitioner to simulate a non-IID, federated setup.

The selected datasets are selected to cover various learning scenarios from small-scale tables to centralized image classification to federated non-IID settings with different levels of data heterogeneity. We can train a controller on MNIST and FEMNIST and compare its performance in MNIST to that in CIFAR-10 to determine whether a controller trained on a lower dimensional space, simpler color distribution, and less complex datasets can be used to predict and control a higher dimensional space, more complex color distribution, and more complex datasets.

The number of training rounds was decreased because of computational limitations, as compared to the previous experiments. CIFAR-10 is larger (32×32 vs 28×28), is a multicolor image (three channels vs grayscale), and uses pixel values ranging from [0, 255]. These all contribute to a much larger update matrix. However, the 10 clients per round and the 10 clients per batch size were kept the same as in previous experiments to make it consistent.

Figure 12
Effect of learning rate on testing accuracy

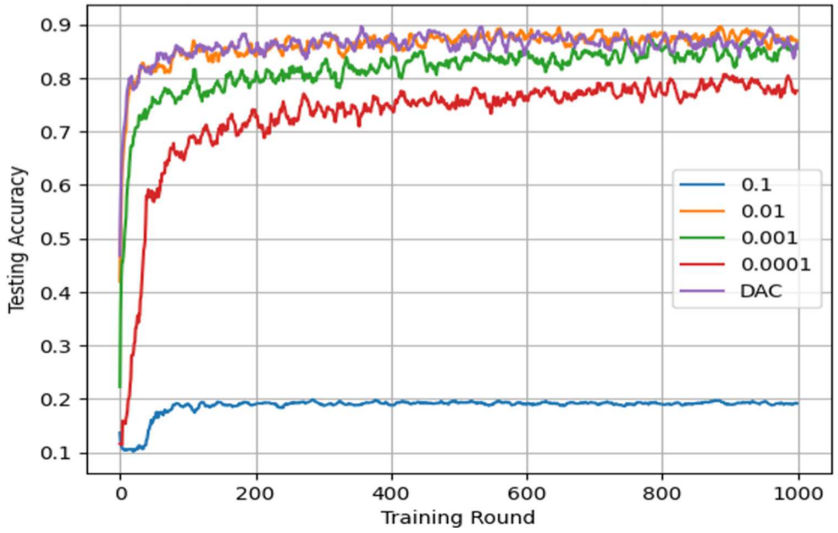


Figure 13
Effect of number of clients on training loss

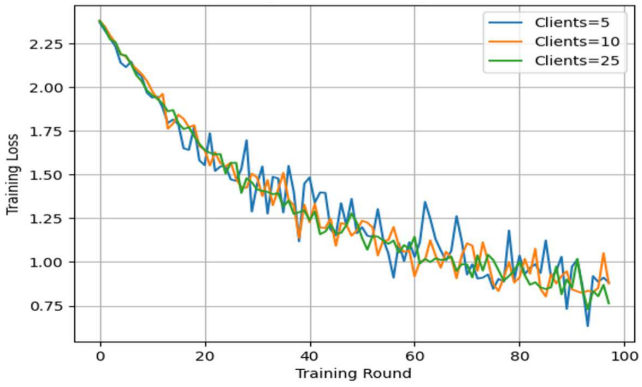


Figure 14
Effect of number of clients on accuracy

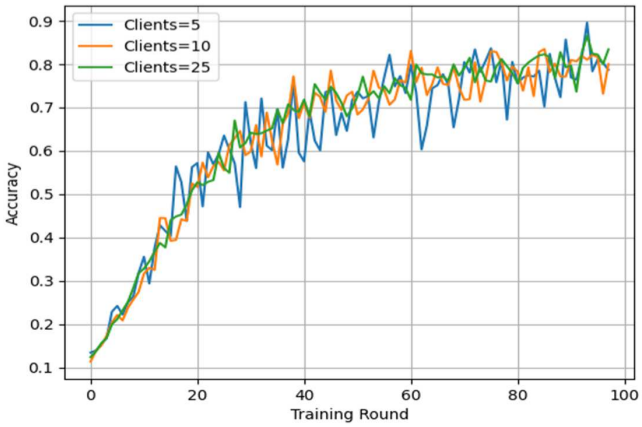


Figure 15
Learning rate and training loss

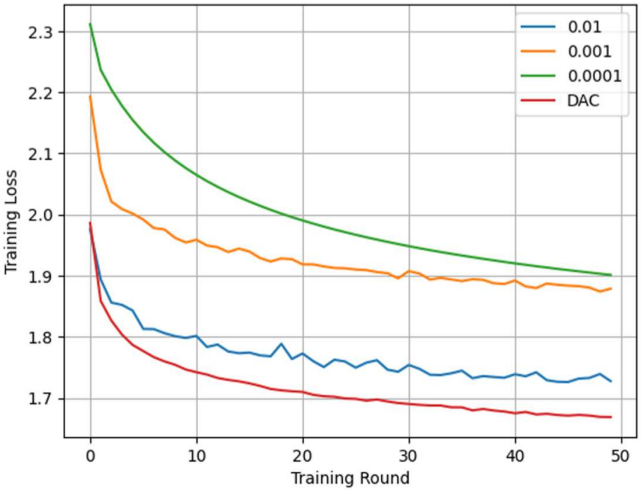
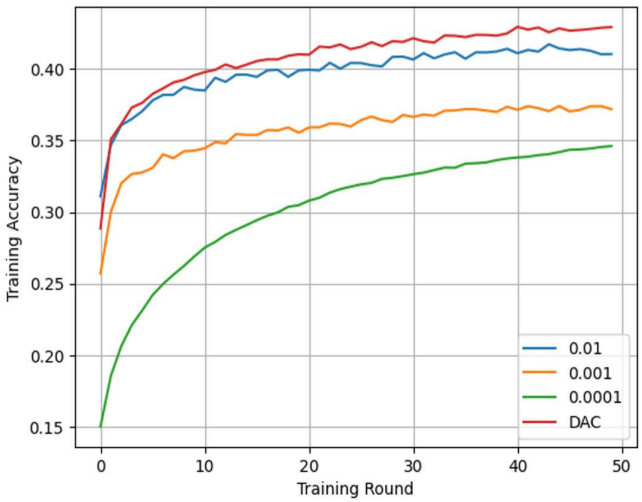


Figure 16
Learning rate and accuracy



The comparison of different learning rate is shown in Figures 15 and 16. DAC still is better than other methods. But the absolute performance of the model is relatively poor, mainly caused by the limited amount of training and the intrinsic

constraints of SVMs. For instance, the CIFAR-10 is very complicated to represent with SVMs.

From all our experiments, it is clear that learning rate plays a significant role in the performance of a machine learning model. Certain learning rates lead to poor performance and do not converge, and others only converge after a protracted training period. The sensitivity of DAC is however robust: the learning rate is dynamically calculated at every time step.

7. Discussions

The section will discuss the limitations of the proposed approach and suggest directions for further work.

A main assumption of our architecture is the capability of the learning rate controller to generalize. This allows meta-controller to be trained on a huge public dataset and then deployed on the client nodes. Although this assumption holds true in practice for the MNIST and CIFAR-10 datasets, we do not have a rigorous theoretical guarantee of the generalization of the controller to other datasets at this time. The construction of these theoretical foundations is a key way to advance in the future.

Another drawback is the computational complexity involved in training the meta-controller with SMAC3, which has a non-trivial computational overhead, at the start. A thorough analysis of existing adaptive optimization methods may aid in determining whether or not this cost is justified. Even with this overhead, controller is resilient to initial learning rate selection, a noteworthy advantage over commonly used optimizers like Adam, AdaGrad, and RMSProp. Moreover, the coordinator is currently using an aggregating approach based on the dataset size, which is commonly used in standard FedAvg. Other aggregation algorithms like performance based or adaptive weighting were not studied in this work. These strategies can also help to reduce client heterogeneity even more and can combine well with the recommended dynamic learning rate controller. We will take into account such aggregation mechanisms and their impact on the performance of the system in our future work.

We experimented with TensorFlow Federated, a simulation-based framework which is primarily focused on algorithmic research. Thus, system-level parameters (such as computational overhead, memory usage, communication efficiency, and behavior in the real world) were not measured directly. These indicators are deployment-specific, for example, hardware heterogeneity, network circumstances, and client availability, and are left for future system-level evaluation.

At first sight, the proposed controller optimizes the convergence by tuning the optimization dynamics based on the local training state. Static learning rates might not be optimal in a federated and heterogenous client data setting when there are varying loss landscapes and varying magnitudes of gradients. The controller uses attributes that reflect agreement with the gradient, variation in loss, and uncertainty to help control the updates sent by the client and make them less sensitive to the initial learning rate. The formal convergence analysis with data with heterogeneity, however, is an open problem.

Last but not least, feature set used in this work relies on previous studies and although effective, it can be restrictive for the expressiveness of controller. The focus of future work will be on rich feature representations. Additional limitations are, our strategy is applicable to SGD-trained models, but not other optimization algorithms, fairness in FL is an open problem, especially when sensitive attributes are not available. These problems are important future work items.

8. Conclusion

The proposed work is the deployment of support vector Machines (SVM) in federated environment with a controller to dynamically adjust the learning rate. NNs are typically the majority of the players in the traditionally FL arena. But they can lead to problems in resource-limited settings due to its computation complexity and network overhead. SVMs are simple and efficient and are much smaller than a typical NN. The proposed system has two phases: controller training and federated deployment. The controller is initially trained using SMAC3 and is meta-trained. Then, in FL, each client is equipped with this controller, which dynamically controls the learning rate. This adaptive mechanism enhances the convergence and gives better performance. Using these methodologies and optimizations, the accuracy of the proposed method is close to the state-of-the-art methods.

Acknowledgment

The authors would like to thank the editor and anonymous referees for their constructive comments. The authors acknowledge the use of Quill Bot for increasing the readability of the manuscript. The authors gratefully acknowledge Sri Mata Amritanandamayi Devi (Amma), Chancellor, Amrita Vishwa Vidyapeetham, for her inspiration and for providing financial support for the Article Processing Charges (APC) of this publication.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in Kaggle at https://git-disl.github.io/GTDLBench/datasets/mnist_datasets/ (MNIST), the NIST repository at https://www.nist.gov/itl/products-and-services/emnist-dataset (FEMNIST), and the University of Toronto's repository at https://www.cs.toronto.edu/~kriz/cifar.html (CIFAR-10).

Author Contribution Statement

Deebakkarthi Chinnasame Rani: Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing. **Gowtham Ramesh:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Writing – original draft, Writing – review & editing, Supervision, Project administration. **Sountharrajan Sehar:** Validation, Formal analysis, Investigation, Resources, Data curation, Writing – review & editing, Project administration. **Varun Aiyaswamy Kannan:** Formal analysis, Resources, Data curation. **Elambharathi Padmavathi Thangavel:** Formal analysis, Writing – original draft, Visualization. **Bharath Kumar Kanapareddy:** Investigation, Resources, Data curation.

References

[1] Wong, K.-S., Nguyen, D.-M., Le, K., Ho, L., Do, C., & Le-Phuoc, D. (2026). An empirical study of federated learning on IoT–Edge devices: Resource allocation and heterogeneity. *IEEE Transactions on Neural Networks and Learning Systems*, *37*(2), 753–765. https://doi.org/10.1109/TNNLS.2025.3611415

[2] Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 27. https://doi.org/10.1145/1961189.1961199

[3] Nair, D. G., Aswartha Narayana, C. V., Jaideep Reddy, K., & Nair, J. J. (2022). Exploring SVM for federated machine learning applications. In *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2022*, 295–305. https://doi.org/10.1007/978-981-19-1018-0_25

[4] Luo, C., Cui, S., Song, J., Zhang, X., Wu, W., Liu, C., . . . , & Hu, C. (2025). SMTgazer: Learning to schedule SMT algorithms via bayesian optimization. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering*, 1273–1285. https://doi.org/10.1109/ASE63991.2025.00109

[5] Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., & Lindauer, M. (2020). Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *24th European Conference on Artificial Intelligence*, 427–434. https://doi.org/10.3233/FAIA200122

[6] Eimer, T., Biedenkapp, A., Reimer, M., Adriaensen, S., Hutter, F., & Lindauer, M. (2021). DACBench: A benchmark library for dynamic algorithm configuration. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 1668–1674. https://doi.org/10.24963/ijcai.2021/230

[7] Daniel, C., Taylor, J., & Nowozin, S. (2016). Learning step size controllers for robust neural network training. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, *30*(1), 1519–1525. https://doi.org/10.1609/aaai.v30i1.10187

[8] Adriaensen, S., Biedenkapp, A., Shala, G., Awad, N., Eimer, T., Lindauer, M., & Hutter, F. (2022). Automated dynamic algorithm configuration. *Journal of Artificial Intelligence Research*, *75*, 1633–1699. https://doi.org/10.1613/jair.1.13922

[9] Thérien, B., Joseph, C.-E., Knyazev, B., Oyallon, E., Rish, I., & Belilovsky, E. (2024). µLO: Compute-efficient meta-generalization of learned optimizers. In *OPT2024: 16th Annual Workshop on Optimization for Machine Learning*.

[10] Yurdem, B., Kuzlu, M., Gullu, M. K., Catak, F. O., & Tabassum, M. (2024). Federated learning: Overview, strategies, applications, tools and future directions. *Heliyon*, *10*(19), e38137. https://doi.org/10.1016/j.heliyon.2024.e38137

[11] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, 106775. https://doi.org/10.1016/j.knosys.2021.106775

[12] Uddin, M. P., Xiang, Y., Hasan, M., Bai, J., Zhao, Y., & Gao, L. (2025). A systematic literature review of robust federated learning: Issues, solutions, and future research directions. *ACM Computing Surveys*, *57*(10), 245. https://doi.org/10.1145/3727643

[13] Li, H., Huang, K., Yang, W., Wang, S., & Zhang, Z. (2020). On the convergence of FedAvg on non-IID data. *arXiv Preprint:1907.02189*.

[14] Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2020). Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems*, *2*, 429–450.

[15] Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S., Stich, S., & Suresh, A. T. (2020). SCAFFOLD: Stochastic controlled averaging for federated learning. In *Proceedings of the 37th International Conference on Machine Learning*, *119*, 5132–5143.

[16] Wang, J., Liu, Q., Liang, H., Joshi, G., & Poor, H. V. (2020). Tackling the objective inconsistency problem in heterogeneous federated optimization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 7611–7623.

[17] Reddi, S. J., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., & McMahan, H. B. (2021). Adaptive federated optimization. *arXiv Preprint: 2003.00295*

[18] Rateria, D., Siddesh, G. M., Laddha, L., Prakash, D., Singh, P., & Sekhar, S. R. (2026). FedDive: Escaping local minima through divergence-rewarded exploration in federated learning: A robust approach with gated divergence rewards. *Knowledge and Information Systems*, *68*(1), 28. https://doi.org/10.1007/s10115-025-02623-y

[19] Khan, Z. A., Xia, Y., Jiang, W., & Anwar, M. S. (2025). FedEmo: A federated learning framework for privacy-preserving emotion detection from handwriting on consumer IoMT devices. *IEEE Transactions on Consumer Electronics*, *71*(4), 11315–11326. https://doi.org/10.1109/TCE.2025.3599558

[20] Jiang, W., Zhang, Y., Han, H., Liu, X., Gwak, J., Gu, W., . . . , & Maple, C. (2025). Fuzzy ensemble-based federated learning for EEG-based emotion recognition in Internet of Medical Things. *Journal of Industrial Information Integration*, *44*, 100789. https://doi.org/10.1016/j.jii.2025.100789

[21] Bai, Y., Jiang, W., Mu, J., Liu, S., Gu, W., & Wang, S. (2025). Enhancing IoT security via federated learning: A comprehensive approach to intrusion detection. *IET Information Security*, *2025*(1), 8432654. https://doi.org/10.1049/ise2/8432654

[22] Xiong, Y., Lan, L.-C., Chen, X., Wang, R., & Hsieh, C.-J. (2022). Learning to schedule learning rate with graph neural networks. In *International Conference on Learning Representations*.

[23] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, *54*, 1273–1282.

[24] Wolberg, S., Mangasarian, O., Street, N., & Street, W. (1993). *Breast Cancer Wisconsin (Diagnostic)* [Data set]. In *UCI Machine Learning Repository*. https://doi.org/10.24432/C5DW2B

[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *arXiv Preprint: 1201.0490*

How to Cite: Chinnasame Rani, D., Ramesh, G., Sehar, S., Aiyaswamy Kannan, V., Thangavel, E. P., & Kanapareddy, B. K. (2026). Federated Learning with SVMs: Dynamic SGD for Efficient Hyperparameter Optimization. *Journal of Computational and Cognitive Engineering*. https://doi.org/10.47852/bonviewJCCE62027792