FedTM: Memory and Communication Efficient Federated Learning with Tsetlin Machine

Shannon How Shi Qi*, Jagmohan Chauhan*, Geoff V Merrett†, Jonathan Hare*
School of Electronics and Computer Science, University of Southampton, Southampton, UK
{s.how,j.chauhan,jsh}@soton.ac.uk*, gvm@ecs.soton.ac.uk†

***Abstract*—Federated Learning has been an exciting development in machine learning, promising collaborative learning without compromising privacy. However, the resource-intensive nature of Deep Neural Networks (DNN) has made it difficult to implement FL on edge devices. In a bold step towards addressing this challenge, we present FedTM, the first FL framework to utilize Tsetlin Machine, a low-complexity machine learning alternative. We proposed a two-step aggregation scheme for combining local parameters at the server which addressed challenges such as data heterogeneity, varying participating client ratio and bit-based aggregation. Compared to conventional Federated Averaging (FedAvg) with Convolutional Neural Networks (CNN), on average, FedTM provides a substantial reduction in communication costs by $30.5\times$ and $36.6\times$ reduction in storage memory footprint. Our results demonstrate that FedTM outperforms BiFL-BiML (SOTA) in every FL setting while providing $1.37-7.6\times$ reduction in communication costs and $2.93-7.2\times$ reduction in run-time memory on our evaluated datasets, making it a promising solution for edge devices.**

***Index Terms*—Federated Learning, Tsetlin Machine, Communication-Efficient**

I. Introduction

Federated Learning (FL) has become increasingly popular in recent years with a broad range of applications spanning various domains such as Automatic Speech Recognition (ASR) and Image Recognition [1]. As user data such as images and personal information are highly sensitive in nature, FL provides a solution to ensure data confidentiality and privacy by allowing the data to remain on the client devices, rather than being shared with a central server for training. With the rise of an ever-increasing number of network-connected devices, such as smartphones and Internet-of-Things (IoT) devices, there is a growing demand towards developing communication-efficient FL models that can cater to the requirements of edge devices while preserving data privacy and achieving high-quality learning outcomes.

The typical FL process uses neural networks that involve rigorous arithmetic computation of gradient descent, multiple rounds of learning on the client's device, uploading and downloading neural net parameters over the network and aggregation on the server [2]. Given the heavy computational and memory-intensive nature of neural networks, it is no surprise that current FL methods might not be able to realize the vision of FL on the edge. Combining this with other issues such as high communication overheads, data heterogeneity, and scalability, the operating scenario for FL at the edge becomes extremely challenging [1].

In our work, we push the boundaries of FL by introducing FedTM, a novel FL framework that leverages the Tsetlin Machine (TM) [3] as an alternative to traditional neural networks. TM utilizes propositional logic and learning automata, providing a low-complexity design with exceptional learning capabilities, making it an excellent candidate for memory and communication-efficient FL. Our two-step aggregation approach with $\mathbf{TopK}$ and $\mathbf{AverageCW}$ offers a solution to alleviate the challenge of aggregating bit-based representations, data heterogeneity and varied participating client ratio. Through extensive experiments on three image datasets, we demonstrate that FedTM achieves a compelling balance of efficiency and performance compared to the four baselines, and outperform the BiFL-BiML (SOTA) approach while reducing average communication costs across our evaluated datasets by $3.57\times$. Our work presents an exciting and promising avenue to explore the potential of TM in the context of FL, opening up new possibilities for efficient FL on edge devices.

In summary, our contributions are as follows:

- We proposed a FL framework based on the TM, which offers a substantial reduction in parameter requirements through logic-based formulation and bit-based representation, making TM a highly appealing alternative to neural networks for FL.
- We introduced a two-step aggregation approach with $\mathbf{TopK}$ to handle bit-based aggregation and $\mathbf{AverageCW}$ to mitigate the challenge that comes with aggregating TM parameters in the presence of data heterogeneity and low participating client ratio.
- Our experiments on three image datasets demonstrated that FedTM outperforms BiFL-BiML in all FL settings while using $2.93-7.2\times$ less run-time memory and with $1.37-7.6\times$ lower communication costs. Furthermore, our framework performs significantly better than the 4 baselines in the presence of high data heterogeneity and with varying ratios of participating clients.

To the best of our knowledge, FedTM is the first FL framework that employs Tsetlin Machine to simultaneously enhance communication efficiency and memory. By leveraging the distinct properties of TM, we have developed a novel approach that outperforms FL baselines in terms of memory and communication efficiency and we believe that our work represents a significant step towards optimizing FL for edge devices.

II. BACKGROUND

We will first introduce Federated Learning, followed by the concept of the Tsetlin Machine and how it works. We intend this section to serve as a primer for our readers to get the necessary background knowledge on TM to make our design philosophy easy to understand in Section III.

A. Federated Learning

FL is deployed to allow distributed learning across an extensive network of devices without sharing local data. As a result, there is a central server to iteratively consolidate the local parameters. Federated Averaging (FedAvg) [2] is a widely adopted strategy for aggregation and it works as described in Algorithm 1. Given N number of clients, the server may choose to select a fraction of clients ($J << N$), or use all clients to perform local model training ($J = N$). It assumes the Stochastic Gradient Descent (SGD) method where each client updates their weights by computing the SGD of the global model for a predefined number of local epochs, e and learning rate λ, on their dataset during each communication round. The clients then transmit their weights to the central server where they are aggregated based on their dataset sizes $|D_j|$ as seen in (1), to obtain the global model, $\mathbf{W}_t$ at communication round t.

$$\mathbf{W}_t = \frac{1}{|D|} \sum_{j=1}^{J} |D_j| \mathbf{W}_t^j \tag{1}$$

This iterative process continues until communication round T where the global model is finalized with $\mathbf{W}_T$.

Algorithm 1: FedAvg [2]

```
Server:
  Initialize $\mathbf{W}_0$
  for communication round $t = 1, 2, ...T$ do
    for client $j = 1, 2, ...N$ do
      $\mathbf{W}_t^j \leftarrow$ ClientUpdate($j, \mathbf{W}_{t-1}$)
    $\mathbf{W}_t = \frac{1}{|D|} \sum_{j=1}^{J} |D_j| \mathbf{W}_t^j$

  ClientUpdate($j, \mathbf{W}$):
      for epoch $i = 1, 2, ...e$ do
        for batch $b = 1, 2, ...B$ do
            $\mathbf{W} \leftarrow \mathbf{W} - \lambda \nabla \ell(b, \mathbf{W})$
      return $\mathbf{W}$ to server
```

B. Tsetlin Machine Primer

The Tsetlin Machine (TM) is an innovative machine-learning algorithm that relies on propositional logic. It employs logic expressions as building blocks to identify distinctive patterns in data. In contrast to DNNs, the TM first converts raw data into booleans, enabling the construction of conjunctive clauses from literals, which are the input variables (X) and their negations ($\neg X$). The TM is composed of three fundamental modules, as shown in Fig. 1: the clause computing module, the summation and threshold module, and the feedback module. The clause computing module consists of teams of Tsetlin Automata (TA), which are finite state machines (FSMs) that control the output of each clause. The TA has internal states that decide on the inclusion or exclusion of the input features and their negations, using the AND bitwise operator. Half of the clauses have positive polarity (C^+) and the other half has negative polarity (C^-). The positive clauses are used to identify class $Y = 1$, while the negative clauses are used to identify class $Y = 0$. During a training cycle, the output of the clause computing module is summed using a weighted or non-weighted voting mechanism that indicates class confidence, regulated by a predetermined threshold. The feedback module then compares the expected output ($\hat{Y}$) with the actual output (Y) and assigns reinforcements, either penalty or reward, to the TA states. The probability of a penalty or reward in the automaton is influenced by a preset learning sensitivity value that controls the flexibility with which the TAs can transition between states [4]. The cycle ends when the TA updates its states and actions based on the reinforcements. This process repeats until a prescribed objective is met. Finally, inference is performed with the final states and actions of the TA, without the feedback module [3].

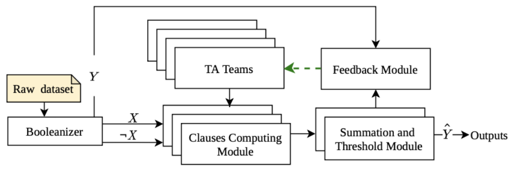

Fig. 1: The TM employs multiple teams of TA to construct discriminative conjunctive clauses with classification determined through majority voting based on the clause outputs [4].

1) Tsetlin Machine Architecture: Each clause composes of a collective of TAs and each TA is represented with bits such that state indexes of the TAs of a clause are represented with several bit sequences. The action of each TA is then derived from the most significant bit which is computed from a series of bitwise operators as seen in Fig. 2. This is required for inference and the other bit sequences can be dropped as they are used only for training, which minimizes the memory footprint of TMs [3]. Additionally, only incremental

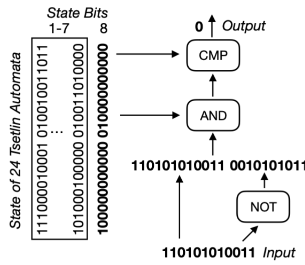

Fig. 2: Example of the bit-based representation with sequence 8 as the most significant bit which is derived from a series of bitwise operators (NOT, AND, CMP - comparison) [3].

and decremental operations are used during learning which further reduces the memory usage of TMs, making TMs ideal for deployment on devices with limited memory [4].

2) Convolutional Tsetlin Machine (CTM): CTM aims to learn better sub-patterns by using clauses as convolution filters that will be computed multiple times based on the patch size. During training, a single patch is randomly chosen to be used for updating a clause. Since each clause outputs values instead of a single output for TM, the outcome of this computation on each patch will be combined with the OR bitwise operator to form the output of a convolution clause [5]. For a given positive clause C_i^+, the output is the combination of the positive clauses, $c_i^{b,+}$, on each patch, b:

$$C_i^+[m] = \bigvee_{b=1}^{B} c_i^{b,+} \tag{2}$$

Additionally, in the multi-class scenario, the TA teams corresponding to class $Y = m$ are trained as per $Y = 1$ and a random class $Y \neq m$ is selected to train as per $Y = 0$. Feedback is modulated based on the weighted sum of clause outputs and a threshold. The final classification is then computed using the argmax operator to output the class $\hat{Y}$ with the largest sum [3]:

$$\hat{Y} = \underset{m=1...M}{\arg\max}\left(\sum_{i=1}^{c/2} W_i^+[m]C_i^+[m] - \sum_{i=1}^{c/2} W_i^-[m]C_i^-[m]\right) \tag{3}$$

The CTM further reduces the overall complexity by reiterating with the same TM across the whole data, focusing on a specific patch at a time, enabling it to learn sub-patterns as well as simple Convolutional Neural Networks (CNN), on various MNIST datasets while using less memory [5].

III. Design of FedTM

Motivated by the logic-based formulation and bit-based representation of TMs [3], we propose FedTM for memory and communication efficient FL. Unlike neural networks in which the server aggregates the floating point parameters with FedAvg [2], FedTM requires a two-step aggregation approach to independently aggregate the clause weights and states which will be described next. An overview of FedTM is provided in Algorithm 2.

A. Integer Clause Weights Aggregation

TM constructs patterns with conjunctive clauses that are built from literals [5]. Clause weighting enhances accuracy, reduces computation time and memory usage as it replaces multiple clauses with one, and this improves the influence of each clause [6]. In TM, there exists c number of clauses for each class $(m = 1...M)$, as in (3). In order for the clause weights to be aggregated at every communication round, all J participating clients, would have to upload $M * c$ local parameters each to the server.

Algorithm 2: FedTM

```
Server:
  1. Initialize global parameters $\mathbf{W}_0, \mathbf{S}_0$ with the same TM architecture and clients inform the server of their local dataset sizes, $|D_j|, j = 1, 2, ...N$
  for communication round $t = 1, 2, ...T$ do
    2. For all participating clients, $J$, train a TM model with the current weights, $W_{t-1}$ on their local dataset, $D_j$, for $e$ epochs
    3. Clients upload their local parameters
    Server:
    4. Aggregation of clients' parameters
    for class $m = 1, 2, ...M$ do
      $\mathbf{W}_t[m] \leftarrow$ AverageCW$(m, \delta, t)$
      $\mathbf{S}_t[m] \leftarrow$ TopK$(m, k, t)$
    5. All clients download the new global parameters: $\mathbf{W}_t, \mathbf{S}_t$

AverageCW$(m, \delta, t)$:
    $\mathbf{W}_t[m] \leftarrow int(\frac{1}{|D|}\sum_{j=1}^{J}|D_j|\mathbf{W}_t^j[m])$
    if t $> 1$ then
      if $\forall_{j=1}^{J}\mathbf{W}_t^j[m] = 0$ then
        $\mathbf{W}_t[m] \leftarrow \mathbf{W}_{t-1}[m]$  $\triangleright$ if class $m$ is not seen in round $t$ of training then use previous weights
      else
        $\mathbf{W}_t[m] \leftarrow (1-\delta)\mathbf{W}_{t-1}[m] + \delta\mathbf{W}_t[m]$
  return $int(\mathbf{W}_t[m])$

TopK$(m, k, t)$:
    $sorted\_list \leftarrow sort(\forall_{j=1}^{J}|D_j|[m])$
    $sorted_k \leftarrow sorted\_list[0:k]$
    $\mathbf{S}_t[m] \leftarrow \bigvee_j^{sorted_k}\mathbf{S}_t^j[m]$
  return $\mathbf{S}_t[m]$
```

Inspired by FedAvg [2], we aggregate the c integer clause weights for each class m, for the participating J clients using a weighted average based on their dataset sizes as in (4).

$$\mathbf{W}[m] \leftarrow int\left(\frac{1}{|D|}\sum_{j=1}^{J}|D_j|\mathbf{W}^j[m]\right) \tag{4}$$

However, the existence of class imbalance can pose a significant challenge for FedTM. As each class has its clause weights, in the presence of data heterogeneity and low participation client ratio, a particular class can be under-represented at communication round t. Therefore, the global model may not make accurate predictions for that class, resulting in continued poor performance throughout the FL process. Therefore, to allow for sustained learning, we utilize the clause weights at communication round $t-1$ for the classes that are not present at communication round t.

Additionally, the clause selection to be used in the voting mechanism in (3) is determined by a predefined threshold parameter. This indicates that a bigger threshold value means more clauses participate in the voting and impact the feedback to TA states. Since the clients repeatedly train with the global model on their local dataset, the weights of the clauses are also being incremented or decremented accordingly [5]. The threshold is kept unchanged throughout the training process, so increasing the training epochs would inherently increase the overall magnitude of the clause weights, causing the FL training to be slow after some communication rounds when the clause weights gets bigger and closer to the threshold.

We propose a solution to address the issue of rapidly growing clause weights by introducing a scaling parameter, denoted by δ as in (5).

$$\mathbf{W}_t[m] \leftarrow (1-\delta)\mathbf{W}_{t-1}[m] + \delta\left(\frac{1}{|D|}\sum_{j=1}^{J}|D_j|\mathbf{W}_t^j[m]\right) \quad (5)$$

This parameter prevents the magnitude of clause weights from increasing excessively after each communication round, and also ensures fair treatment of classes in the presence of high data heterogeneity with a low number of participating clients. Overall, it helps to maintain consistency in the magnitude of clause weights between consecutive communication rounds, even when certain classes are absent. Our experiments in the non-IID scenario, as illustrated in Section V, Fig. 5, demonstrate the effectiveness of our approach in achieving consistent performance across varying participation ratios.

B. States Aggregation

Since TAs are represented as bits, integer aggregation cannot be applied to the states of the TAs. In TM, the OR bitwise operator is used between conjunctive clauses to jointly represent sub-patterns with a single clause [7]. We applied this principle between the TA states of each clause across all clients for each class m, $\mathbf{S}^j[m]$, to represent the same pattern that is expressed differently with a single clause, $\mathbf{S}[m]$ as shown in (6).

$$\mathbf{S}[m] \leftarrow \bigvee_{j=1}^{J} \mathbf{S}^j[m] \quad (6)$$

However, there are limits on using bit representation (i.e. 0-4294967295 for a 32-bit unsigned integer), so combining all TA states across all N clients will result in the quick convergence to the limit, resulting in the learning of the FL system going stale as seen in Fig. 3.

As the clause weights do not influence the TA state updates [5], the aggregation of TA states can be done independently from the clause weights aggregation meaning that if J selected clients partake in the clause weights aggregation, it does not require for all J selected clients to also partake in the TA state aggregation. Therefore, we introduce **TopK**, to prevent stale learning by only using a predetermined number of clients, K, to aggregate the states. Since the parameters are computed across clients for each class, we will select K clients based on the confidence of the TA states, using the clients with the top K size for the particular class. This further minimizes the communication costs as only K clients' parameters are required to be uploaded at each communication round.

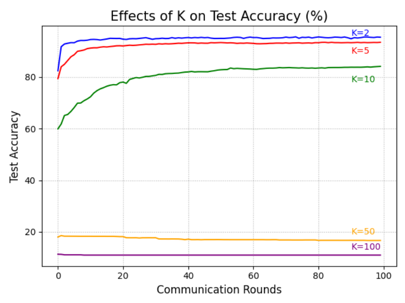

Fig. 3: An example of how the number of participating clients (K), with 32-bit representation, affects convergence of FL on the MNIST dataset with 100 clients

IV. Experimental Setup

We describe our experimental setup in detail in this section.

1) Baselines: To make a fair comparison with the existing works, we benchmark the performance of FedTM with the following four baseline models:

- FA (CNN): A Convolutional Neural Network (CNN) as the base model with FedAvg (Algorithm 1).
- BiFL-FULL: A Binary Neural Network (BNN) as the base model with FedAvg and full-precision parameters uploaded and downloaded.
- BiFL-Bi-UpDown: BNN as the base model with binarized parameters uploaded and downloaded.
- BiFL-BiML: BNN as the base model with binarized parameters uploaded but downloaded parameters are estimated with Maximum Likelihood to provide more robust parameter updates at reduced communication cost.

Note that the term **BiFL** encompasses all variations.

2) Datasets: The datasets that we used to evaluate the different frameworks are MNIST [8], Extended MNIST (FEMNIST) [9] and Fashion-MNIST (F-MNIST) [10]. All datasets are downloaded and preprocessed with PyTorch [11].

- MNIST: A dataset of handwritten digits with 10 classes. It contains 28x28 greyscale images with 60,000 training data and 10,000 testing data.
- FEMNIST: The extended version of MNIST which contains 814,255 characters with 62 unbalanced classes. Similar to BiFL [12], we only used a subset of the entire dataset for training and testing.
- F-MNIST: A dataset of Zalando's article images with 10 classes. It also contains 28x28 greyscale images with 60,000 training data and 10,000 testing data.

3) Evaluation Metrics: We ran each experiment on 5 independently initialized dataset partitions and evaluated them on the testing dataset after 100 communication rounds. We present the average of each experiment as the accuracy measure. The total upload and download communication costs are recorded for each communication round in terms of the size of the communicated parameters between the server and the 100 clients. We ran all experiments on a general compute node with 32 CPU cores and recorded the average latency of training the model at each client and estimated the run-time memory using PyPi's memory-profiler and PyTorch Profiler [11].

4) Data Distribution: Data distribution is done by sampling class priors from a Dirichlet distribution, in which a parameter α is used to determine the heterogeneity of splits [13]. As $\alpha \to 0$, the partitions tend to get more heterogeneous but when $\alpha \to \infty$, the partitions tend to get more identical. For our experiments, we used $\alpha = 10000$ as the IID (Independent and Identically Distributed) setting. For the non-IID setting, we use $\alpha = 1$ for low heterogeneity and $\alpha = 0.01$ for high.

5) Model Configurations and FL Setup: The default FL system for the experiments consists of one server with $N = 100$ clients and each experiment is carried out for 100 communication rounds. We used all N clients for aggregation at each communication round unless stated otherwise.

- We evaluated FA (CNN) using a CNN with 2 convolutional layers and 2 fully connected layers with a softmax function to output the predicted labels [12]. BiFL uses a BNN with the same structure as the 2-layer CNN, but with binarized convolutional and fully-connected layers.
- Decaying learning rate after the 30th and 60th round: {0.005, 0.002, 0.001} for MNIST and F-MNIST, {0.0003, 0.0001, 0.00005} for FEMNIST as specified in the paper [12].
- The CTM model for FedTM varies for each dataset, and we specify the values for each definable parameter and the local epochs in Table I. We used $\delta = 0.1$ in $\mathbf{AverageCW}$ for all the experiments. As observed in Fig. 3, we select $K = 2$ for $\mathbf{TopK}$ as it provides the optimal learning rate for FedTM.
- Since TM requires booleanized input, for the MNIST and FEMNIST data we encoded our data by replacing pixel values larger than 75 with 1 and below or equal to 75 as 0. As for the F-MNIST dataset, we referred to the original CTM paper and binarized it with an adaptive Gaussian thresholding procedure with window size 11 and threshold value 2 [5].

TABLE I: FedTM model configuration for all datasets

	MNIST	FEMNIST	F-MNIST
Number of clauses	100	300	200
Feedback Threshold	1000	1000	1000
Learning Sensitivity	5	5	5
Patch-Dimension	(10,10)	(10,10)	(5,5)
Local Epochs	100	50	100

V. Results

A. Performance on IID data distribution

To begin, we assess the effectiveness of various techniques when applied to the IID data distribution. Our evaluation criteria include accuracy, communication costs and latency. In FedTM, each client will send their clause weights and TA states to the server using 32-bit integers for clause weights and 32-bit unsigned integers for the TA states representation. To compute the upload cost, we consider a fixed number of clauses (c) and classes (M). For each class, J participating clients upload their clause weights, resulting in a total of $J * c * M$ parameters. However, since we are using $\mathbf{TopK}$, only the TA states of a maximum of K clients are required for aggregation, reducing the cost to $K * M *$ number of TA states. For the download phase, all clients retrieve the global model containing the clause weights and TA states.

Table II show that FedTM outperforms BiFL-Bi-UpDown and BiFL-BiML, with lower communication costs than BiFL-BiML by 1.75×, 7.6× and 1.37× across the MNIST, FEMNIST, and F-MNIST datasets, FedTM achieves higher accuracy and exhibits less fluctuation during training as observed in Fig. 4. This is most likely because FedTM trains the dataset with more local epochs during each communication round, resulting in higher test accuracy and less fluctuation during training. While FedTM exhibits a slight performance difference from FA (CNN) and BiFL-FULL, the substantial reduction in communication costs by 16.45×, 63.47×, and 11.58× on the evaluated datasets justifies a small trade-off between accuracy and efficiency especially in cases where edge devices might be involved and communication costs are more prohibitive than the computational costs.

Despite the additional local training performed by FedTM, the latency is not significantly impacted. It takes an average of 0.89 seconds for each client in FedTM to train for MNIST, 1.93 seconds for FEMNIST and 2.07 seconds for F-MNIST. This is compared to 1.93, 8.85 and 1.91 seconds for BiFL.

B. Data Heterogeneity

We evaluate the methods on non-IID data with the same settings as in the IID experiments, without additional hyperparameter tuning, consistent with other FL experiments [12], [14]. As presented in Table III, the algorithms perform relatively well when data heterogeneity is low, comparable to the performance on the IID data distribution. However, as data heterogeneity increases, the performance of all algorithms diverges. This divergence occurs because the local data distribution progressively deviates from the global data distribution, leading to a corresponding divergence in the local parameters from the global parameters, hence a decline in the ideal performance. It is worth noting that the other baseline methods deviates significantly from their IID setting compared to FedTM and FedTM easily outperforms all baseline models when data heterogeneity is high.

C. Effect of Number of Participation Clients

As only a subset of participants can be expected to be connected to the server at any point in time, participation ratio is an important characteristic of FL. We evaluated the impact of the number of participating clients for aggregation using the same experimental setup and evaluation metrics with 100 clients for 100 communication rounds. However, instead of having all 100 clients participate in the aggregation at each round, as in the above experiments, we varied the number of clients to be {10,30,50}. Our results are shown in Fig. 5.

In general, the number of participating clients did not significantly affect the performance in the IID setting. However,

TABLE II: Test Accuracy and Communication Costs of 100 clients on IID data distribution ($\alpha = 10000$)

	MNIST		FEMNIST		F-MNIST	
	Accuracy (%)	Upload/Download (MB per round)	Accuracy (%)	Upload/Download (MB per round)	Accuracy (%)	Upload/Download (MB per round)
FA (CNN)	98.41	32.9/32.9	80.84	2640/2640	87.74	32.9/32.9
BiFL-FULL	97.12	32.9/32.9	79.27	2640/2640	87.40	32.9/32.9
BiFL-Bi-UpDown	83.70	1.0/1.0	57.36	82.6/82.6	73.23	1.0/1.0
BiFL-BiML	91.16	1.0/6.8	61.57	82.6/550	82.18	1.0/6.8
FedTM	94.85	0.47/4.0	69.50	8.78/74.4	83.36	0.88/4.8

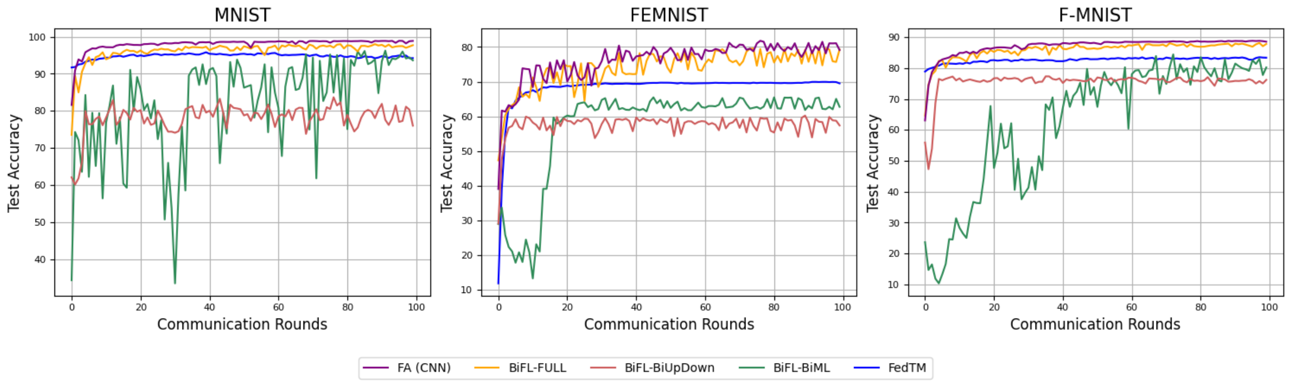

Fig. 4: Comparison of convergence speed between FedTM and the baselines

TABLE III: Test Accuracy (%) on non-IID data distribution

		MNIST	FEMNIST	F-MNIST
Low ($\alpha = 1$)	FA (CNN)	97.66	77.30	85.98
	BiFL-FULL	96.48	76.01	85.82
	BiFL-Bi-UpDown	83.15	57.14	77.54
	BiFL-BiML	88.04	65.23	80.82
	FedTM	94.72	70.24	82.64
High ($\alpha = 0.01$)	FA (CNN)	59.61	12.43	33.07
	BiFL-FULL	11.53	13.91	13.06
	BiFL-Bi-UpDown	11.03	8.64	18.29
	BiFL-BiML	10.22	12.24	13.10
	FedTM	69.61	44.25	50.36

we did observe a slight improvement in FedTM's performance with a smaller client participation ratio. This suggests that we could potentially use fewer devices and still maintain FedTM's performance while reducing communication costs further.

For non-IID data with varying numbers of participating clients, it remains a significant challenge for BiFL as FL training does not guarantee convergence. However, a varied number of participating clients does not affect the performance of FedTM as much as BiFL. This suggests that FedTM is better equipped to handle non-IID data scenarios and can be potentially deployed in constrained environments where the available number of connected devices is limited.

D. Scalability

We also evaluated the performance of FedTM in terms of scalability in both the cross-silo and cross-device setting. We conducted our experiments for 10 clients in the cross-silo setting and increased the number of clients to 200 in the cross-device setting. The results demonstrate that all FL algorithms

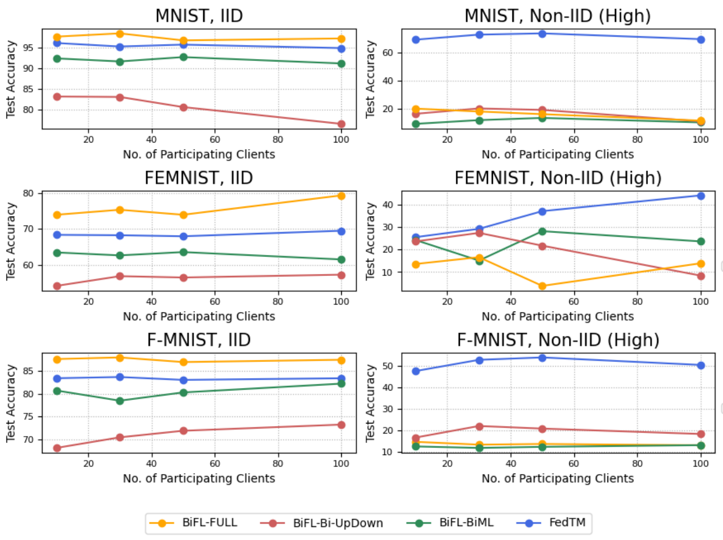

Fig. 5: Effect of participating clients on performance

varies similarly with FedTM outperforming BiFL-Bi-UpDown and BiFL-BiML with a varying number of clients, suggesting that FedTM can be an efficient solution to large-scale FL applications.

E. Memory Costs

Model storage size is an important consideration, particularly for edge devices with limited storage capacity. We compared the model storage size of FedTM with that of FA (CNN) and BiFL. As shown in Table IV, we observed that the

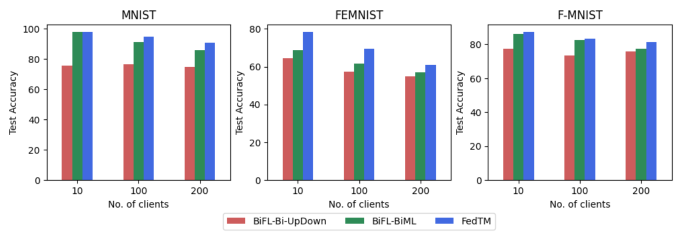

Fig. 6: Testing accuracy with different number of clients

bit-based representation used in CTM significantly reduced the number of parameters required for model storage, making it a more storage-efficient option for edge devices. Notably, CTM used in FedTM is considerably smaller than that of the CNN and BNN (BiFL-FULL) with reductions of $8.24\times$, $35.7\times$, and $6.6\times$ across the evaluated datasets. The low storage costs for BiFL-Bi-UpDown and BiFL-BiML compared to FedTM come at the expense of reduced accuracy as seen in Table II and III. Overall, the smaller size of FedTM maintains a good balance between accuracy and efficiency.

TABLE IV: Memory Storage (MB) of the algorithms used in our experiments

	Dataset		
	MNIST	FEMNIST	F-MNIST
FA (CNN) BiFL-FULL	0.33	26.43	0.33
BiFL-Bi-UpDown BiFL-BiML	0.01	0.83	0.01
FedTM	0.04	0.74	0.05

We also computed the average run-time memory for FedTM and BiFL per client during training. Table V shows that FedTM has significantly lower run-time memory usage per client compared to BiFL that utilizes auxiliary real-valued weights for training [12], which contributed to their higher run-time memory usage. Specifically, CTM used $2.93\times$, $4.98\times$, and $7.2\times$ less run-time memory than BNN. Overall, our results suggest that FedTM is not only storage-efficient but is also more memory-efficient than BiFL.

TABLE V: Run-time Memory (MB) of each client

	Dataset		
	MNIST	FEMNIST	F-MNIST
BiFL	30.60	355.9	30.28
FedTM	10.45	71.4	4.20

VI. Discussion

FedTM consistently performs well compared to FL with neural networks, achieving peak accuracy within 30 communication rounds across all datasets. Moreover, FedTM exhibits robustness against varying numbers of participating clients, enabling the reduction of communication costs by ensuring convergence with fewer participants and reduced number of communication rounds. While data heterogeneity can lead to a slight trade-off between efficiency and accuracy, we suggest exploring sparser representations for the states of each TA to improve performance while minimizing communication costs. Furthermore, future work could also focus on deriving a theoretical proof for the scaling parameter, δ in our $\mathbf{AverageCW}$ approach to enhance the performance of FedTM. While our experiments showed promising results, particularly with the image datasets, we plan to extend FedTM's capabilities to constrained devices and to other domains such as audio and Natural Language Processing (NLP) to prove its generality.

VII. Related Work

Federated Learning (FL) has gained significant traction due to its ability to address concerns related to data privacy. However, challenges arise due to data heterogeneity and client selection, which impact communication overhead. To tackle data heterogeneity, several techniques have been proposed, such as encoding local seed samples for data reconstruction at the server [15], clustering users based on contextualization [16] and bi-partitioning [17], and knowledge distillation [18].

The FedAvg algorithm [2], known for its ability to randomly select clients for aggregation, may lead to slow training efficiency and performance due to the variability of communication and computing resources among the clients. Various client selection techniques have been proposed to overcome this challenge. Some techniques aim to aggregate local clients that can benefit the global model [19]–[21], while others focus on balancing the label distribution of the participants [22]. Additionally, some techniques only select clients with a lower degree of data heterogeneity [23]. These methods have shown promising results in improving the efficiency and performance of FL models, and further research can explore their effectiveness in different scenarios.

However, the aforementioned challenges of FL exacerbate communication costs. To mitigate these costs, various methods have been proposed such as enabling more device-level computations [24], [25] or introducing additional variables [24], [26], [27] to reduce the number of required communication rounds for convergence. Other common techniques include quantization [28], [29], sparsification [30], [31] and pruning [14], [19], [32], which reduce the upload size of updates. These approaches have successfully reduced communication costs while maintaining comparable performance. Notably, BiFL has demonstrated effective training of Binary Neural Networks (BNN) with lower storage memory and communication costs while maintaining sufficient performance.

FL systems rely on neural networks due to their powerful pattern recognition capabilities, which has led to a focus on resolving FL challenges based on neural network training in prior research. However, the computational complexity of training and optimizing neural networks presents significant challenges. The Tsetlin Machine (TM), a logic-based machine learning framework, offers a promising alternative to neural networks by significantly reducing parameter requirements while maintaining competitive performance in a range of applications, including low-power audio keyword spotting [33], natural language understanding [34], and computer vision [35].

Furthermore, TM's reliance on logic operations and energy-efficient propositional logic-based learning makes it appealing for hardware implementations and constrained devices [4].

The earliest implementation of FL with neural networks on microcontrollers relied on transfer learning [36]. However, this approach only focused on re-training the weights of the last fully-connected layer for inference, neglecting the training of the entire model on the local dataset. As a result, this approach can lead to significant accuracy degradation due to the lack of comprehensive training. Furthermore, this method requires higher memory storage compared to the innovative techniques used in FedTM and BiFL. In comparison, our approach is the first to leverage the potential of TM that can enable practical realization of FL on many edge devices.

VIII. Conclusions

In this paper, we introduced FedTM, a novel FL framework that utilizes the Convolutional Tsetlin Machine to significantly reduce communication and memory costs while maintaining FL's performance. Our two-step aggregation scheme with $\mathbf{TopK}$ and $\mathbf{AverageCW}$ effectively addresses challenges such as data heterogeneity, client participation ratio, and bit-based parameter aggregation. FedTM was compared with four baselines, and the results demonstrated that it provides a promising tradeoff between efficiency and performance. Notably, FedTM outperforms all baselines for the non-IID scenario with varying numbers of participating clients, and significantly outperforms BiFL-BiML in all FL settings, reducing communication costs by $1.37-7.6\times$ and run-time memory by $2.93-7.2\times$. To further explore the scalability and generality of FedTM, future work could focus on addressing challenges such as data heterogeneity and client selection. As the first FL framework utilizing TMs, FedTM shows great potential for efficient FL with TMs on edge devices.

Acknowledgment

This work was supported by the UK Research and Innovation (UKRI) Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1] and the Engineering and Physical Sciences Research Council (EPSRC) International Centre for Spatial Computational Learning [EP/S030069/1].

References

[1] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," *IEEE IoT-J*, vol. 9, no. 1, pp. 1–24, 2022.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *AISTATS*, 2016.

[3] O.-C. Granmo, "The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," 2021.

[4] J. Lei, A. Wheeldon, R. Shafik, A. Yakovlev, and O.-C. Granmo, "From Arithmetic to Logic based AI: A Comparative Analysis of Neural Networks and Tsetlin Machine," in *IEEE ICECS*, pp. 1–4, 2020.

[5] O.-C. G. et al., "The Convolutional Tsetlin Machine," 2019.

[6] A. Phoulady, O.-C. Granmo, S. R. Gorji, and H. A. Phoulady, "The Weighted Tsetlin Machine: Compressed Representations with Weighted Clauses," 2020.

[7] L. Jiao, X. Zhang, and O.-C. Granmo, "On the Convergence of Tsetlin Machines for the AND and the OR Operators," 2021.

[8] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE SPM*, vol. 29, no. 6, pp. 141–142, 2012.

[9] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: Extending MNIST to handwritten letters, year=2017," in *IJCNN*, pp. 2921–2926.

[10] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," 2017.

[11] P. A. et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *NeurIPS*, 2019.

[12] Y. Yang, Z. Zhang, and Q. Yang, "Communication-Efficient Federated Learning With Binary Neural Networks," *IEEE JSAC*, vol. 39, no. 12, pp. 3836–3850, 2021.

[13] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the Effects of Non-Identical Data Distribution for Federated Visual Classification," 2019.

[14] L. A. et al., "Hermes: An Efficient Federated Learning Framework for Heterogeneous Mobile Clients," MobiCom, p. 420–437, 2021.

[15] M. S. et al., "XOR Mixup: Privacy-Preserving Data Augmentation for One-Shot Federated Learning," 2020.

[16] Y. Mansour, M. Mohri, J. Ro, and A. T. Suresh, "Three Approaches for Personalization with Applications to Federated Learning," *CoRR*, 2020.

[17] F. Sattler, K.-R. Müller, and W. Samek, "Clustered Federated Learning: Model-Agnostic Distributed Multitask Optimization Under Privacy Constraints," *IEEE TNNLS*, vol. 32, no. 8, pp. 3710–3722, 2021.

[18] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble Distillation for Robust Model Fusion in Federated Learning," in *NeurIPS*, pp. 2351–2363, 2020.

[19] L. S. et al., "Joint Model Pruning and Device Selection for Communication-Efficient Federated Edge Learning," *IEEE TCOMM*, vol. 70, no. 1, pp. 231–244, 2022.

[20] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient Federated Learning via Guided Participant Selection," in *USENIX OSDI*, pp. 19–35, 2021.

[21] D. Y. et al., "AUCTION: Automated and Quality-Aware Client Selection Framework for Efficient Federated Learning," *IEEE TPDS*, vol. 33, no. 8, pp. 1996–2009, 2022.

[22] M. J. et al., "Client Selection Based on Label Quantity Information for Federated Learning," in *IEEE PIMRC'21*, pp. 1–6.

[23] W. Zhang, X. Wang, P. Zhou, W. Wu, and X. Zhang, "Client Selection for Federated Learning With Non-IID Data in Mobile Edge Computing," *IEEE Access*, vol. 9, pp. 24462–24474, 2021.

[24] X. Yao, T. Huang, C. Wu, R.-X. Zhang, and L. Sun, "Federated Learning with Additional Mechanisms on Clients to Reduce Communication Costs," 2019.

[25] L. Y. et al., "FedBCD: A Communication-Efficient Collaborative Learning Framework for Distributed Features," *IEEE TSP*, vol. 70, pp. 4277–4290, 2022.

[26] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *MLSys*, 2020.

[27] D. A. E. Acar, Y. Zhao, R. M. Navarro, M. Mattina, P. N. Whatmough, and V. Saligrama, "Federated Learning Based on Dynamic Regularization," in *ICLR*, 2021.

[28] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," in *NeurIPS*, vol. 30, 2017.

[29] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary Compression for Communication-Efficient Federated Learning," *IEEE TNNLS*, vol. 33, no. 3, pp. 1162–1176, 2022.

[30] R. et al., "FetchSGD: Communication-Efficient Federated Learning with Sketching," ICML, 2020.

[31] A. F. Aji and K. Heafield, "Sparse Communication for Distributed Gradient Descent," in *EMNLP*, 2017.

[32] J. Y. et al., "Model Pruning Enables Efficient Federated Learning on Edge Devices," *IEEE TNNLS*, pp. 1–13, 2022.

[33] L. J. et al., "Low-Power Audio Keyword Spotting Using Tsetlin Machines," *JLPEA*, vol. 11, no. 2, 2021.

[34] R. Saha, O.-C. Granmo, V. I. Zadorozhny, and M. Goodwin, "A Relational Tsetlin Machine with Applications to Natural Language Understanding," 2021.

[35] S. Glimsdal and O.-C. Granmo, "Coalesced Multi-Output Tsetlin Machines with Clause Sharing," 2021.

[36] K. Kavya and L. Eric, "TinyFedTL: Federated Transfer Learning on Tiny Devices," 2021.