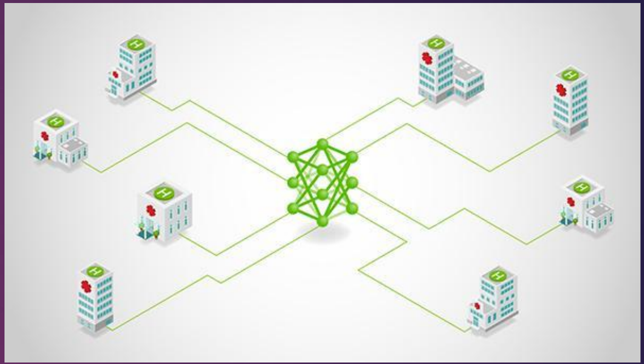
Multi-Task Federated Learning for Personalized Deep Neural Networks



TEAM -6

DEEPTI SINHA

MADHURA KADABA

PRAJNA K

SARTHAK SHARMA

SUBIN PILLAI

IISc FACULTY

PROF. CHANDRAMANI SINGH

Multi-Task Federated Learning (MTFL)

Multi-task Federated Learning (MTFL) is an extension of Federated Learning (FL) that focuses on training multiple related models across distributed devices rather than a single global model, without sharing any raw data.

KEY DIFFERENCES:-

Feature	Federated Learning (FL)	Multi-task Federated Learning** (MTFL)
Model type	Single Global Model	Multiple Personalized Models
Data distribution	Assumed similar data across all clients	Handles heterogenous data across clients (non–IID different but related datasets)
Optimization strategy	Federated Averaging (FedAvg)	Multi-task optimization (FedAvg-Adam**)
Accuracy metric	Global model accuracy	User model accuracy (UA **)
Use case	Common tasks across all clients	Personalized learning for all clients
Computation	Takes place centrally on the server	Off-loaded to client devices

https://arxiv.org/pdf/2007.09236 **proposed in this paper by Jed Mills, Jia Hu, Geyong Min

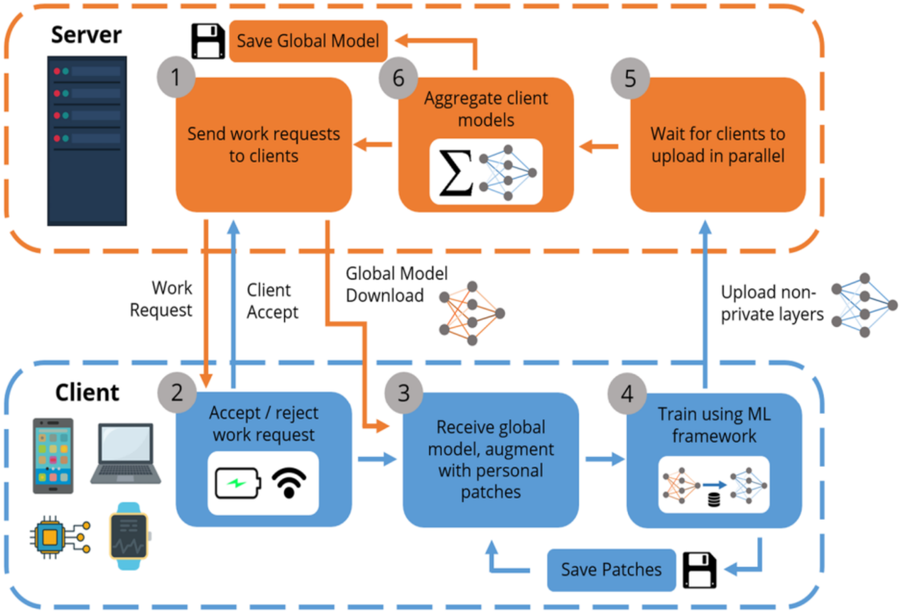
Multi-Task Federated Learning (MTFL)

Feature	Federated Learning (FL)	Multi-task Federated Learning** (MTFL)
Loss function	$$F_{\mathrm{FL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\Omega)$$ where K is the total number of clients, n_k is the number of samples on client k, n is the total number of samples across all clients, ℓ_k is the loss function on client k, and Ω is the set of global model parameters.	$$F_{\mathrm{MTFL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\mathcal{M}_k)$$ $$\mathcal{M}_k = (\Omega_1 \cdots \Omega_{i_1}, P_{k_1}, \Omega_{i_1+1} \cdots \Omega_{i_m}, P_{k_m}, \Omega_{i_m+1} \cdots \Omega_j)$$ where $\mathcal{M}_k$ is the patched model on client k, composed of Federated model parameters $\Omega_1 \cdots \Omega_j$ (j being the total number of Federated layers) and patch parameters $P_{k_1} \cdots P_{k_m}$ (m being the total number of local patches, $\{i\}$ being the set of indexes of the patch parameters) unique to client k.

https://arxiv.org/pdf/2007.09236 **proposed in this paper by Jed Mills, Jia Hu, Geyong Min

MTFL ALGORITHM

MULTI-TASK FEDERATED LEARNING



REFERENCE: https://arxiv.org/pdf/2007.09236

Step 1: The server selects a subset of clients from its database to participate in the round, and sends a work request to them.

Step 2: Clients reply with an accept message depending on physical state and local preferences.

Step 3: Clients download the global model (and any optimization parameters) from the server, and update their copy of the global model with private Batch Norm layers as patches.

Step 4: Clients perform local training using their own data, creating a different model. Clients save the private patch layers (BN) locally, and upload their non private model parameters to the server.

Step 5: The server waits for C fraction of clients to upload their non-private model and optimizer values, or until a time limit.

Step 6: The server averages all models, saves the aggregate, and starts a new round.

Algorithm 1: MTFL

Initialise global model Ω and global optimiser values V
while termination criteria not met **do**
    Select round clients, $S_r \subset S$, $|S_r| = C \cdot |S|$

    for each client $s_k \in S_r$ in parallel **do**
        Download global parameters $\mathcal{M}_k \leftarrow \Omega$
        Download optimiser values $V_k \leftarrow V$
        for $i \in$ patchIdxs **do** $\triangleright$ Apply local patches
            $\mathcal{M}_{k,i} \leftarrow P_{k,i}, V_{k,i} \leftarrow W_{k,i}$
        end for
        for batch b drawn from local data D_k **do**
            $\mathcal{M}_k, V_k \leftarrow \text{LocalUpdate}(\mathcal{M}_k, V_k, b)$
        end for
        for $i \in$ patchIdxs **do** $\triangleright$ Save local patches
            $P_{k,i} \leftarrow \mathcal{M}_{k,i}, W_{k,i} \leftarrow V_{k,i}$
        end for
        for each $i \notin$ patchIdxs **do**
            Upload $\mathcal{M}_{k,i}, V_{k,i}$ to server
        end for
    end for

    for $i \notin$ nonPatchIndexes **do**
        $\Omega_i \leftarrow \text{GlobalModelUpdate}(\Omega_i, \{\mathcal{M}_{k,i}\}_{k \in S_r})$
        $V_i \leftarrow \text{GlobalOptimUpdate}(V_i, \{V_{k,i}\}_{k \in S_r})$
    end for
end while

patchIdxs contain the indexes of the patch layer placement in the DNN

MTFL ALGORITHM

We simulated the federated learning process with Global Server + Multi-Client setup using the Flower framework.

Each sampled client trained locally on their own data.

METRICS : User model accuracy (UA), Number of communication rounds required to achieve benchmark user model accuracy.

STOPPING CRITERIA : Reach benchmark user model accuracy (UA = 97% for MNIST, UA = 65% for CIFAR10) or run for a fixed number of communication rounds (T).

We compared the performance with custom optimization strategy (FedAvg-Adam) and various existing strategies (FedAvg, FedAdam) with/without private BN params.

REFERENCE: https://arxiv.org/pdf/2007.09236

OPTIMIZATION STRATEGIES

FedAvg

LocalUpdate is minibatch-SGD.

GlobalModelUpdate produces the new global model as a weighted (by number of local samples) **average of uploaded client** models.

Clients **do not use distributed adaptive optimization**. (Optimizers V is empty).

GlobalOptimUpdate performs no function.

FedAdam

LocalUpdate is minibatch-SGD.

GlobalModelUpdate, the server takes the **difference** (Δr) between the previous **global model** and the **average uploaded client** model to update the global model using an **Adam**-like update step.

Clients **do not use distributed adaptive optimization**. (Optimizers V is empty).

GlobalOptimUpdate performs no function.

FedAvg-Adam

All clients share a global set of **Adam optimizer** values (Optimizers V) received from server.

During **LocalUpdate**, clients perform **Adam SGD**, and the federated model layers and new **Adam** values are uploaded by clients at the end of the round.

To produce a new global model, the server **averages** the client models in **GlobalModelUpdate** and **averages** the **Adam** moments in **GlobalOptimUpdate.**

Optimisation Strategy	LocalUpdate	GlobalModel Update	GlobalOptim Update
FedAvg	SGD	Average	-
FedAdam	SGD	Adam	-
FedAvg-Adam	Adam	Average	Average

Benefits: Improved convergence speed.
Downside: Increased communication cost per round.

DATASET & FRAMEWORKS USED

Datasets:

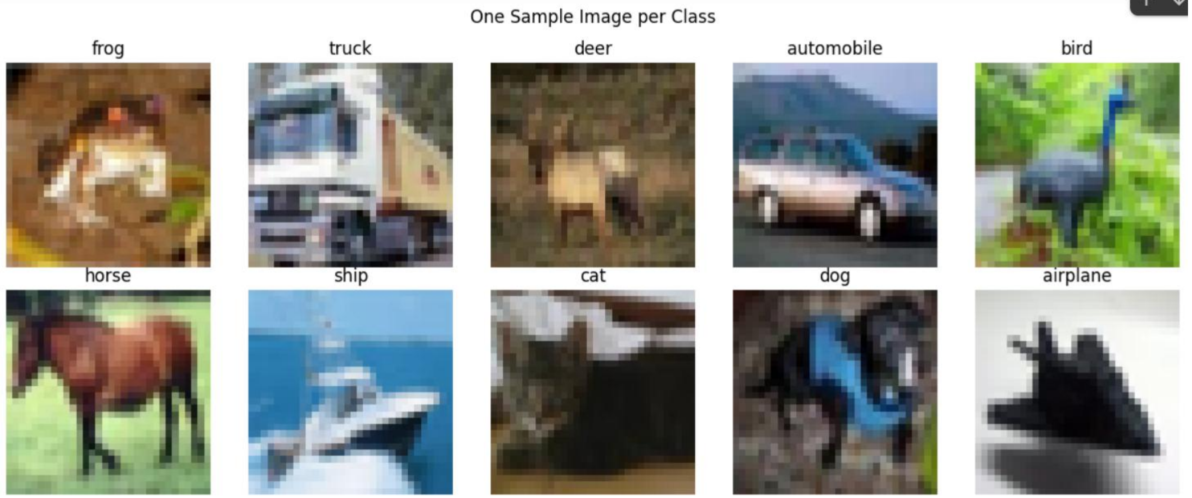
	MNIST	CIFAR10
Data Size	70,000 labelled images (60k train + 10k test)	60,000 labelled images (50k train + 10k test)
Image size	28x28x1 (Gray-scale)	32x32x3 (RGB image)
Classes	10 (handwritten digits)	10 (birds, animals, automobiles)
Transforms	Normalize, Convert to tensor	Normalize, Convert to tensor

PyTorch

Flower

Weights & Biases

Data distribution and example for CIFAR-10



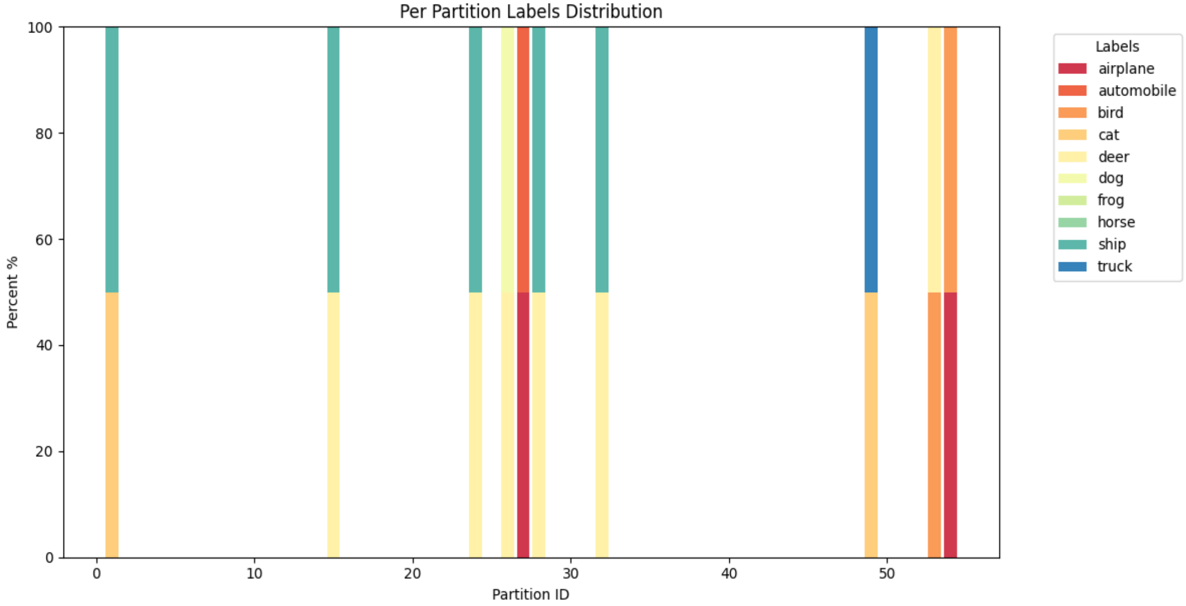
DATA PRE-PROCESSING

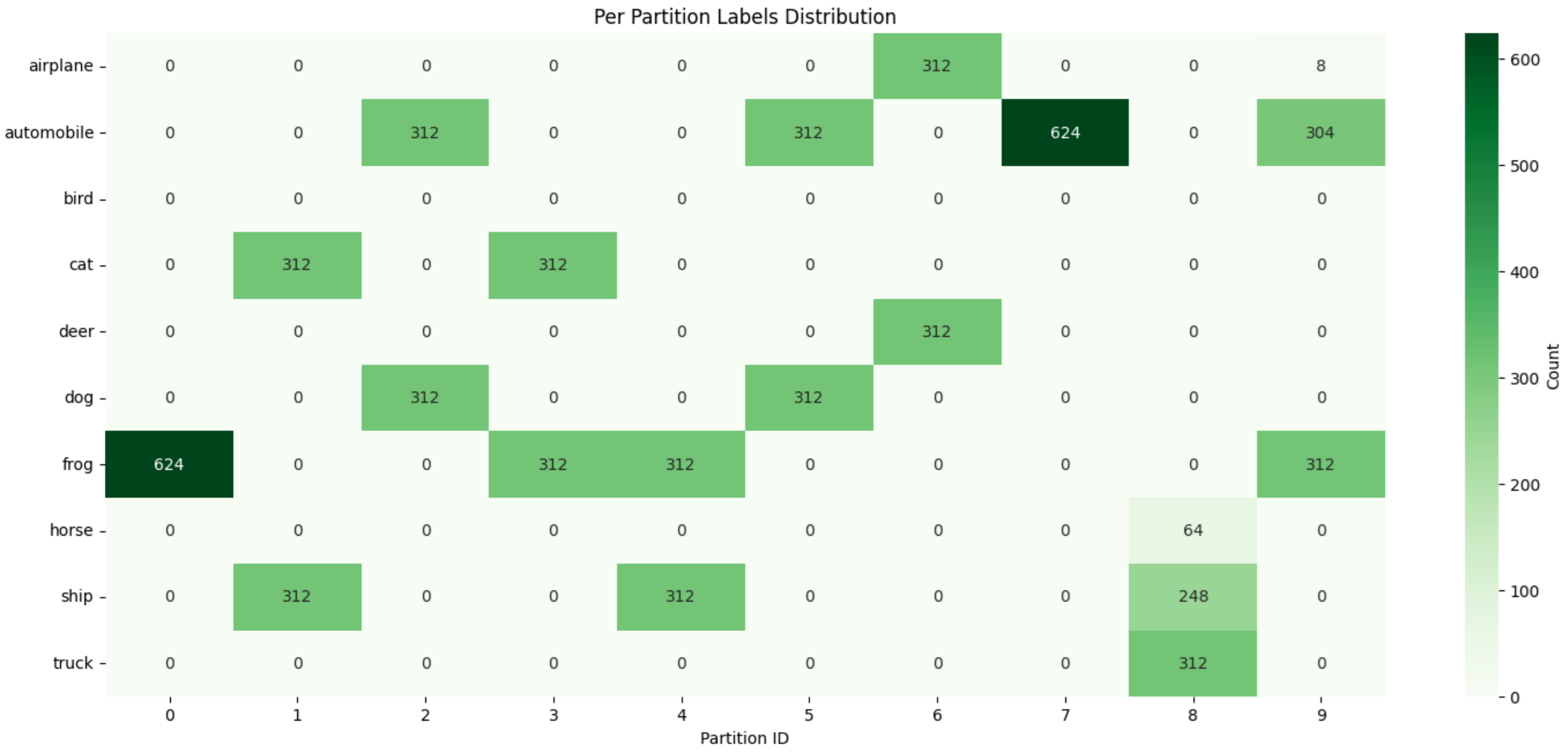
<u>Creating non-IID data-set partitions across clients</u>:

- We created a custom function to generate the non-IID partitions, aligned with the strategy described in the paper.
- First, we order the training and testing dataset by label and then perform the non-IID split.
- Each client has the same classes within their respective training and testing datasets

Number of clients = W
Number of shards = 2xW
Shards per client = 2

Data distribution for 10 random clients post non-IID data-set partition for CIFAR-10




Per Partition Labels Distribution
airplane 0 0 0 0 0 0 312 0 0 8
automobile 0 0 312 0 0 312 0 624 0 304
bird 0 0 0 0 0 0 0 0 0 0
cat 0 312 0 312 0 0 0 0 0 0
deer 0 0 0 0 0 0 312 0 0 0
dog 0 0 312 0 0 312 0 0 0 0
frog 624 0 0 312 312 0 0 0 0 312
horse 0 0 0 0 0 0 0 0 64 0
ship 0 312 0 0 312 0 0 0 248 0
truck 0 0 0 0 0 0 0 0 312 0
0 1 2 3 4 5 6 7 8 9
Partition ID
600
500
400
300
200
100
0
Count

MODEL DETAILS (DNN with private BN)

28 x28 Grey Scale image (MNIST) serialized as 784x1

Linear Fully-connected layer
(Input dims:784, Output dims: 200)

ReLu activation

Batch Normalization layer
(dim: 200)

Linear Fully-connected layer
(Input dims:200, Output dims: 200)

ReLu activation

Linear Fully-connected layer
(Input dims:200, Output dims: 10)

10x1 output array

32 x32x3 RGB image (CIFAR10)

Conv layer: Input channels: 3, output channels: 32,Filter: 3

ReLu activation

Max Pool 2x2

Batch Normalization layer (dim: 32)

Conv layer: Input channels: 32, output channels: 64,Filter:3

ReLu activation

Max Pool 2x2

Batch Normalization layer (dim: 64)

Flatten (dim: 6x6x64)

Linear Fully-connected layer
(Input dims:2304, Output dims: 512)

ReLu activation

Linear Fully-connected layer
(Input dims:512, Output dims: 10)

10x1 output array

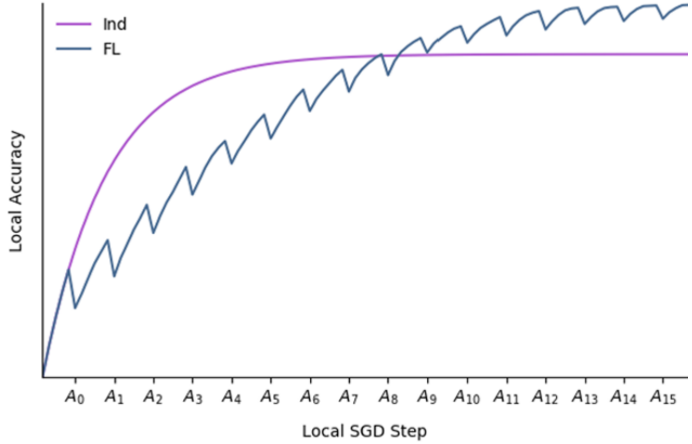
PRIVATE BATCH NORM LAYERS (PATCHES)

Batch Normalization (BN) layers **normalize** the **activations** of a neural network **within each mini-batch** to stabilize and accelerate training by reducing internal covariate shift.

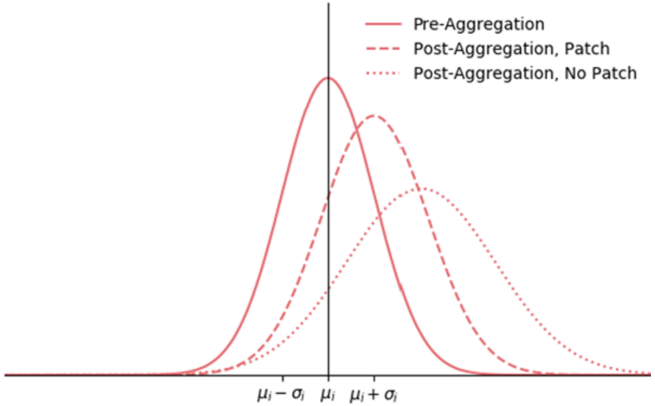
Private BN layers allow local adaptation to **non-IID data**

Adding BN **throughout the network** acts as regularization → local outputs remain closer to their own distribution

Noisy clients can’t “pollute” the global model as easily because **BN updates stay local**



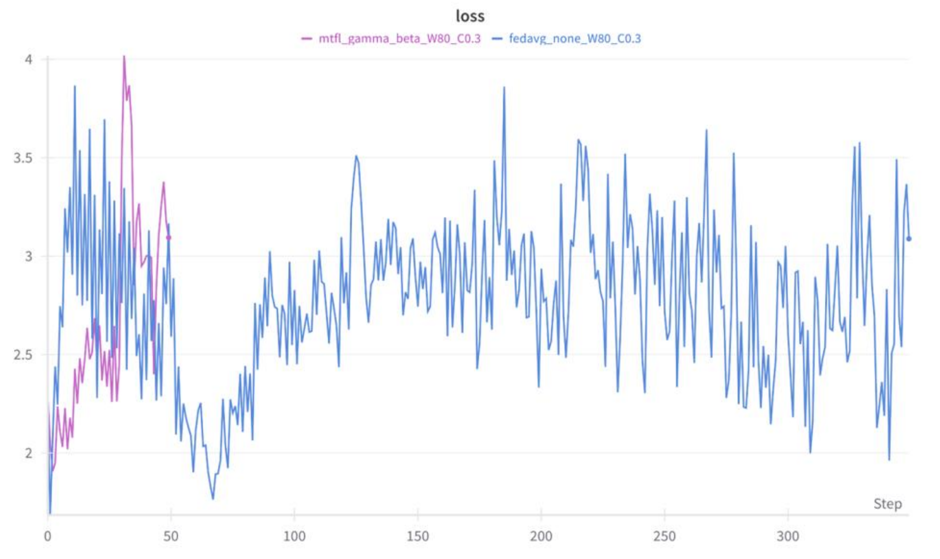
(a) Local accuracy during training

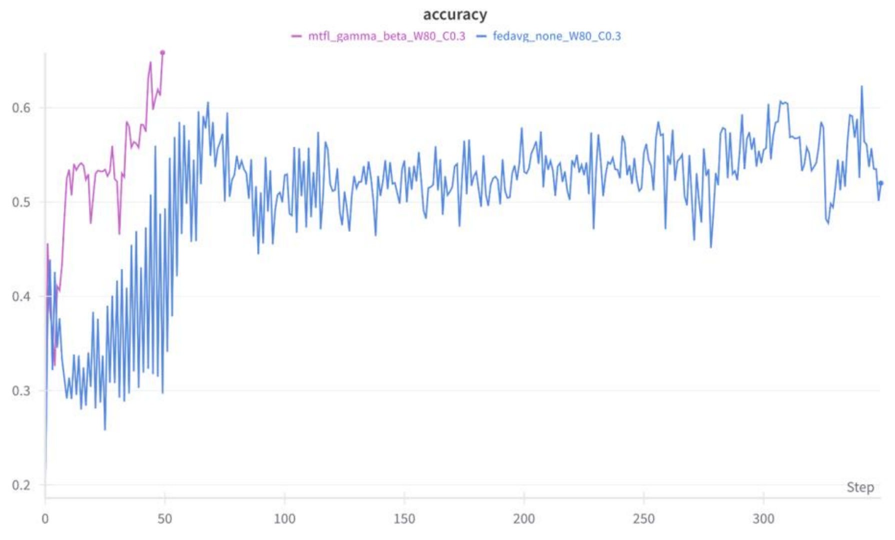


(b) Effect of patch layers on activations

EXPERIMENTS & OBSERVATIONS - (1)

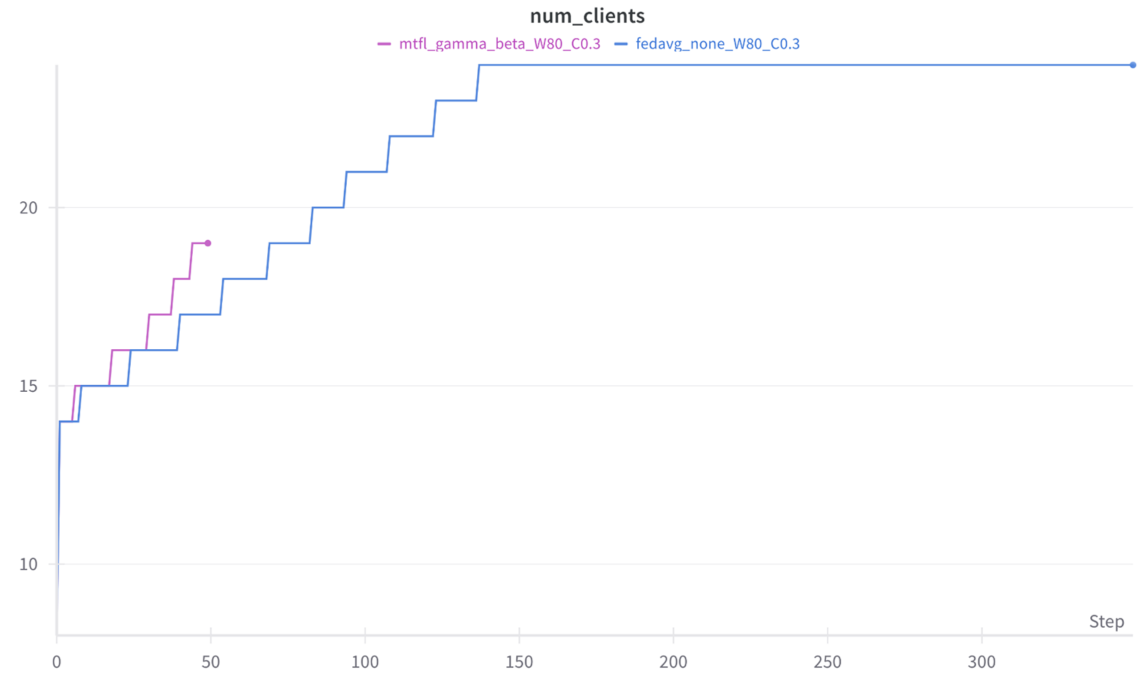
Comparing Loss and User Accuracy for FL-FedAvg vs MTFL-FedAvg-Adam (private gamma and beta) using **CIFAR10 data**





EXPERIMENTS & OBSERVATIONS - (2)

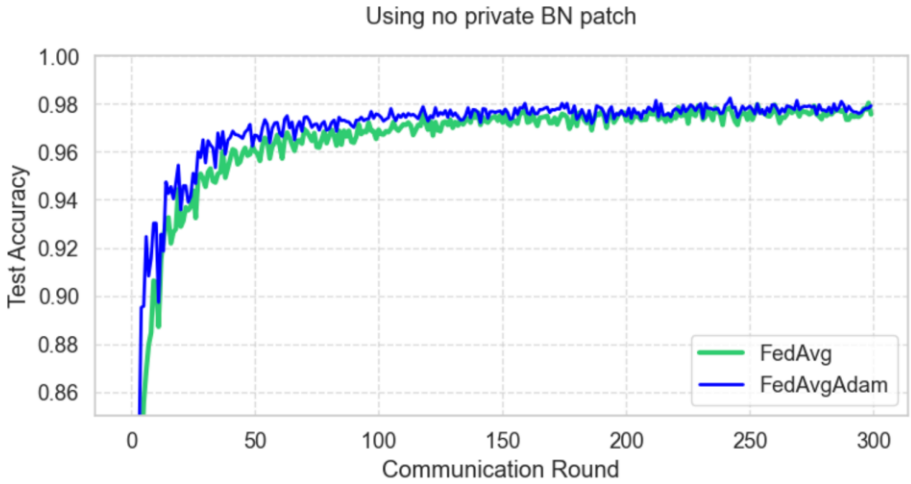
Caveat: Target accuracy for experiment was reached for MTFL-FedAvg-Adam before C (fraction of clients) was connected for training



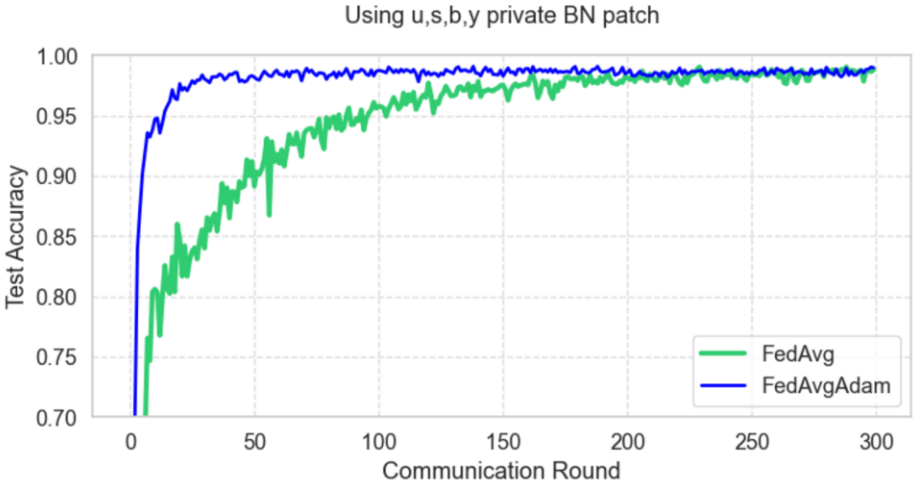
EXPERIMENTS & OBSERVATIONS - (3)

Comparing Test Accuracy for
FL- FedAvg/FedAvg-Adam vs MTFL- FedAvg/FedAvg-Adam using **MNIST** data
Clients = 200, Rounds = 300, Sample = 0.5

FL (No private parameters)



MTFL (u, s, y, b private BN parameters)

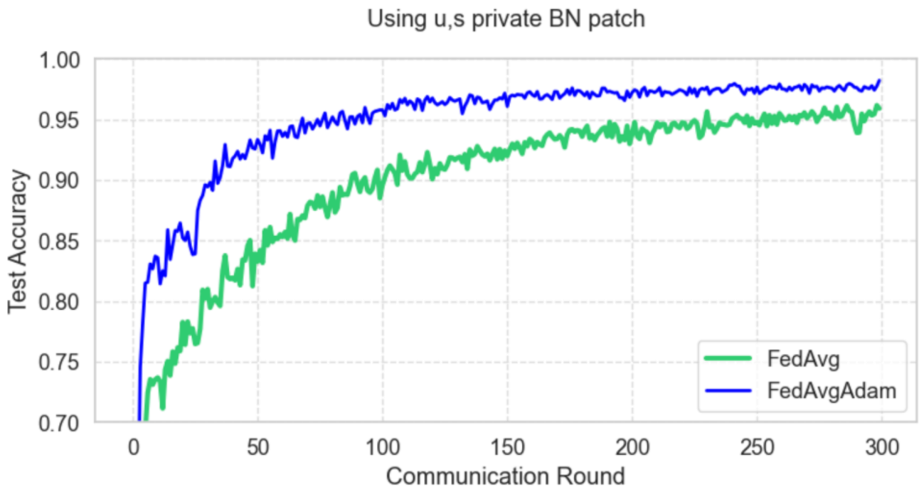


Convergence is faster with FedAvg-Adam algorithm when compared with FedAvg.
No. of rounds needed to reach 97% benchmark accuracy is much lower for case 2 than case 1

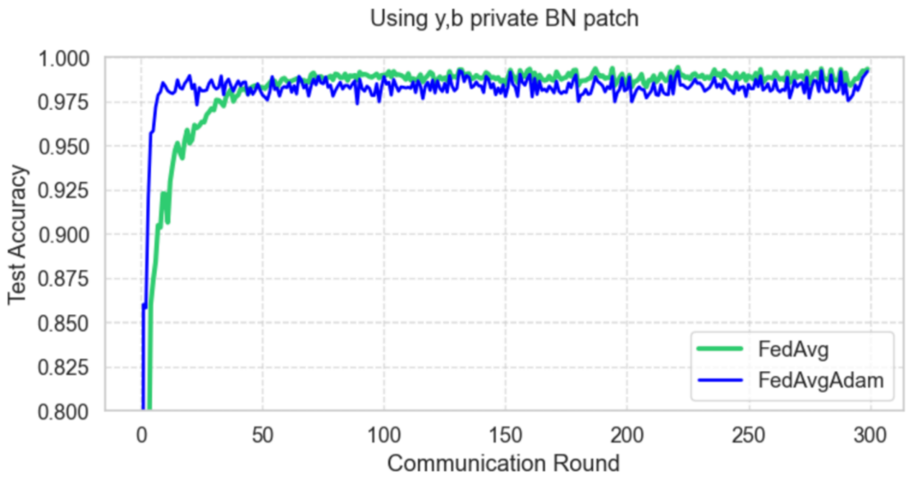
EXPERIMENTS & OBSERVATIONS -(4)

Comparing Test Accuracy for
FL- FedAvg/FedAvg-Adam vs MTFL- FedAvg/FedAvg-Adam using **MNIST** data
Clients = 200, Rounds = 300, Sample = 0.5

MTFL (u,s private BN parameters)



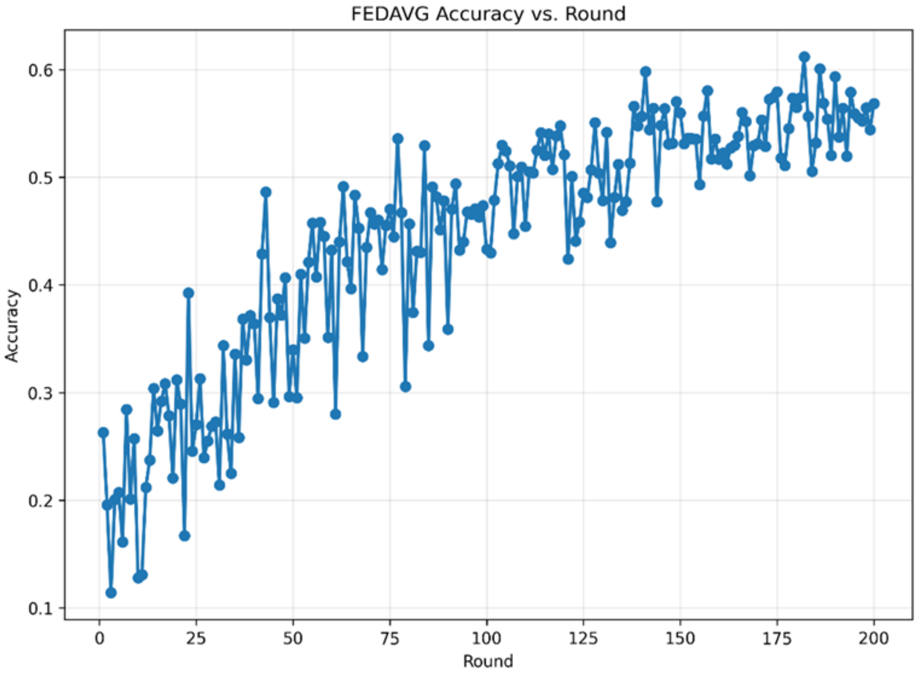
MTFL (y,b private BN parameters)

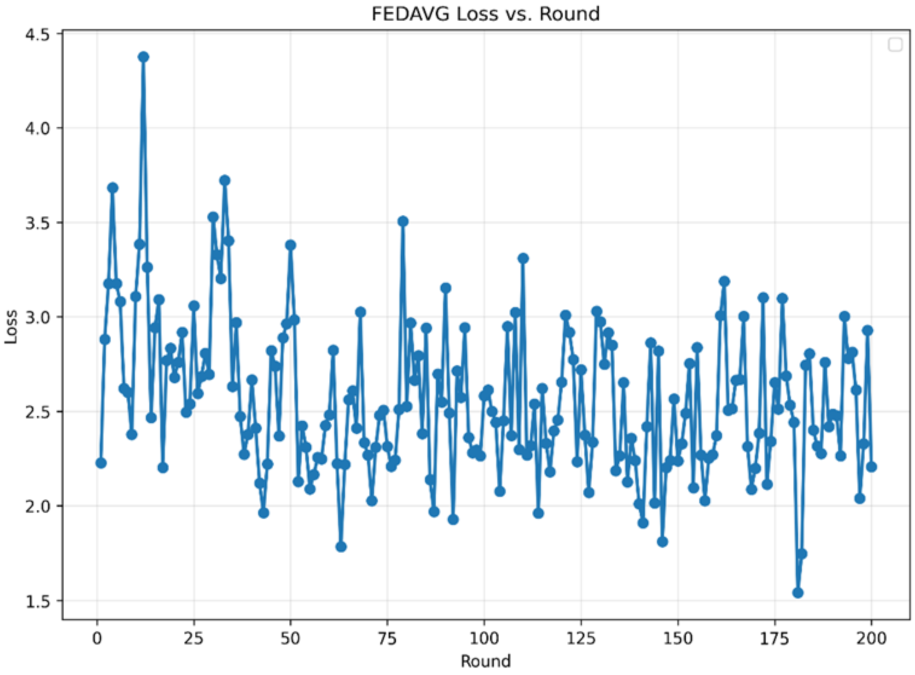


Convergence is faster with FedAvg-Adam algorithm when compared with FedAvg.
No. of rounds needed to reach 97% benchmark accuracy is much higher in case 1 than case 2

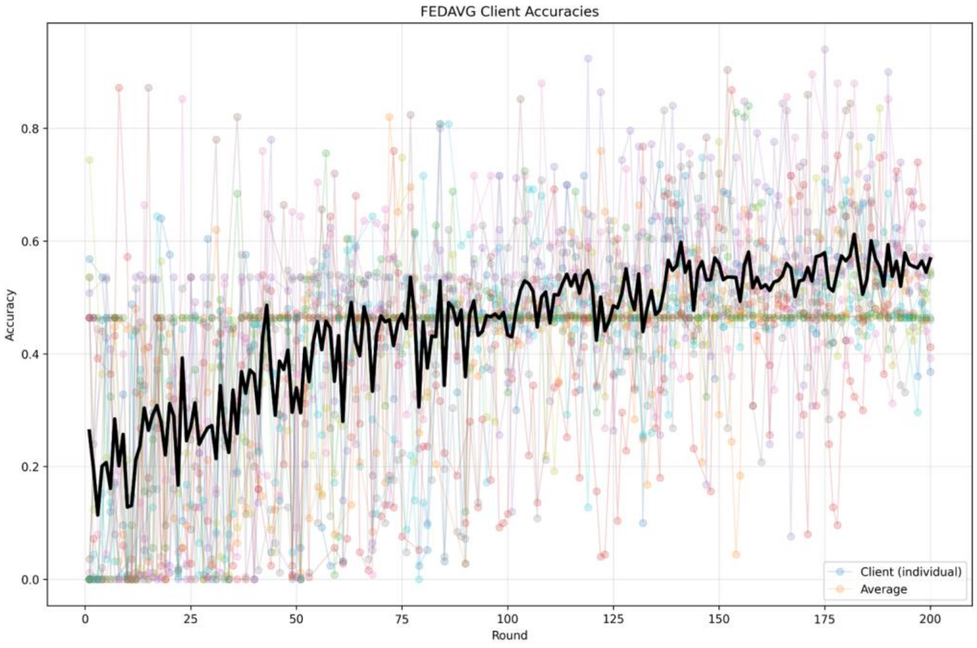
RESULTS AND DISCUSSIONS

Variation in the per-round testing User accuracy and loss in the FL-FedAvg approach





Variation in the per-round testing User Accuracy in the FL-FedAvg approach



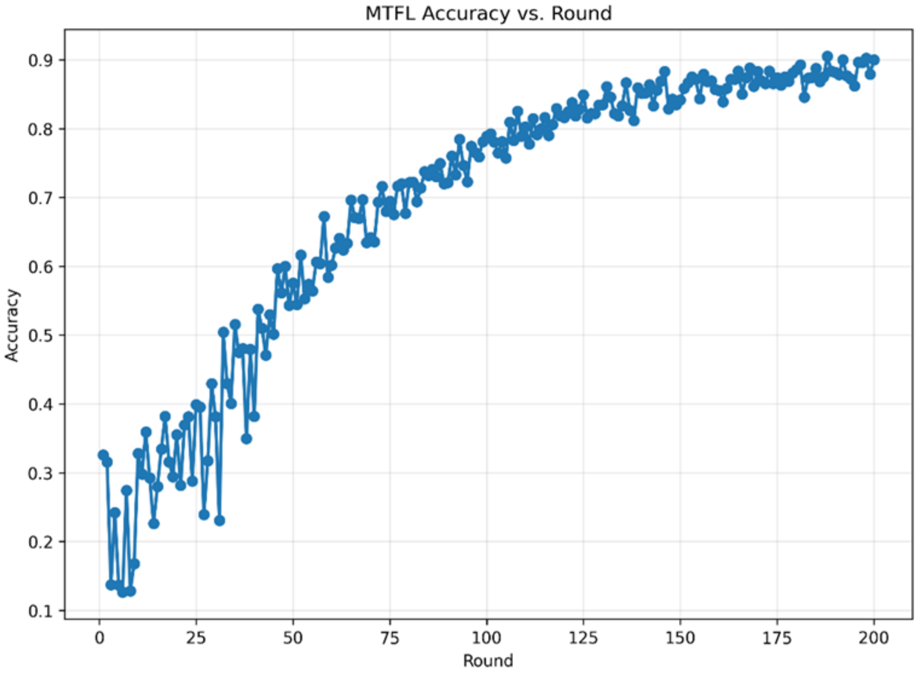
Evidence of **catastrophic forgetting** in FedAvg

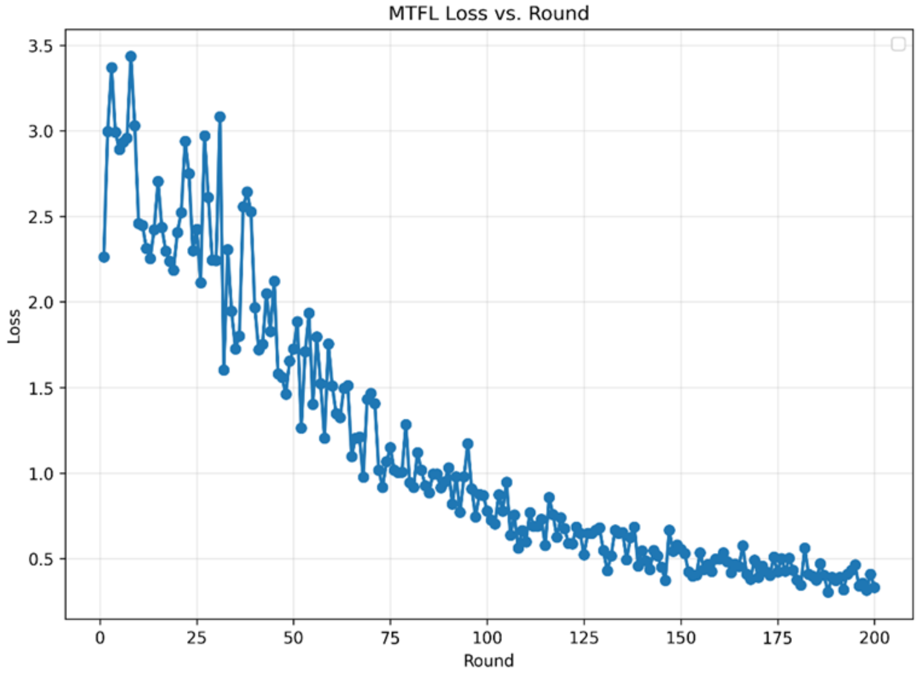
FedAvg has **oscillations** and slower convergence, affected by client drift

FedAvg loss hovers inconsistently between 2.0–3.5, showing **instability** in optimization

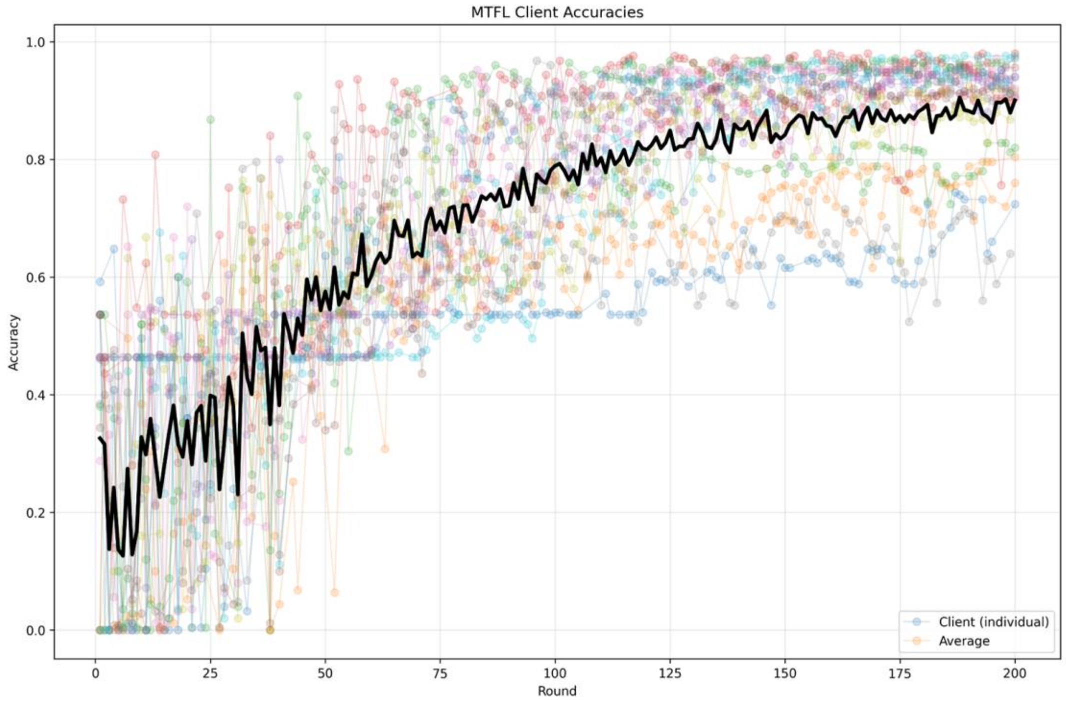
RESULTS AND DISCUSSIONS

Variation in the per-round testing User accuracy and loss in the MTFL-FedAvg approach





Variation in the per-round testing User Accuracy in the MTFL-FedAvg approach



MTFL achieves significantly **better** final accuracy

MTFL **converges faster** and more smoothly

MTFL reduces inter-client variance, **better consistency and stability** in MTFL

MTFL loss graph shows **sharp, smooth decrease** down to ~0.4

Personalized optimization and private BN clearly handle **client heterogeneity**

“Thank You”

<u>**Team -6 :**</u>

Deepti Sinha, Madhura Kadaba, Prajna k, Sarthak Sharma, Subin Pillai