A ALGORITHMIC DESCRIPTION OF CONCEPT MATCHING FRAMEWORK

Algorithm 2 describes the pseudo-code of the CM framework. CM executes as a multi-round, iterative FL cycle (lines 3-9). At each round (line 3), each client operates in parallel (line 5), including clients concept matching (line 13), local model updates with batches of data (line 14-16), and returning the client model weights to the server (line 18). Then, server performs clustering (line 7), aggregation (line 8), and server concept matching to update the global concept models with the aggregated cluster models (line 9).

Algorithm 2 Concept Matching Framework Pseudo-code

```
1: procedure SERVEREXECUTE:
2:     initialize $W^0 = (w_1, w_2, ..., w_K)^0$ randomly, total number of rounds $T$
3:     for $t = 1$ to $T$ do
4:         // Update Done at Clients and Returned to Server
5:         for each client $n$ do // In Parallel
6:             $\theta_n$ = n.CLIENTUPDATE($W^{t-1}$)
7:         $\Omega^t = Cluster(\{\theta_n^t\}_{n=1}^N)$
8:         $\Theta^t = Aggregate(\Omega^t)$
9:         $W^t = ServerConceptMatch(\Theta^t, W^{t-1})$

10: procedure CLIENTUPDATE($W$)
11:     // Executed at Clients
12:     require step size hyperparameter $\eta$, local dataset of current round $\mathcal{D}$
13:     $k^* = ClientConceptMatch(W, \mathcal{D})$
14:     $x_n \leftarrow \mathcal{D}$ divided into minibatches
15:     for each batch $b \in x_n$ do
16:         $\theta_n = w_{k^*} - \eta \nabla L(w_{k^*}, b)$
17:     // Results Returned to Server
18:     return $\theta_n$
```

B THEOREM PROOF

Assumption 1. *Differentiability: The loss function $L(w)$, used to optimize a neural network, is differentiable with respect to the model parameters w.*

Assumption 2. *Lipschitz continuity: The gradient of the loss function $\nabla L(w)$ is Lipschitz continuous with a positive constant L. By Lipschitz continuity definition, for any two points w^1 and w^2, the following inequality holds $\|\nabla L(w^1) - \nabla L(w^2)\| \leq \text{L}\|w^1 - w^2\|$, where $\|.\|$ denotes the norm.*

Lemma 1. *Given a loss function $L(w)$ under assumption 1 and 2, w is updated with gradient descent $w^{t+1} = w^t - \eta \nabla L(w^t)$, where t is the iteration number, η is the learning rate, and $\nabla L(w^t)$ is the gradient of the loss function with respect to w^t, the following inequality holds $\|\nabla L(w^{t+1})\| < \|\nabla L(w^t)\|$.*

Proof. To prove $\|\nabla L(w^{t+1})\| < \|\nabla L(w^t)\|$, we can use the assumption 2. Let L be the Lipschitz constant. Then, we have $\|\nabla L(w^{t+1}) - \nabla L(w^t)\| \leq \text{L}\|w^{t+1} - w^t\|$.

Now, using the reverse triangle inequality, we can write $\|\nabla L(w^{t+1})\| - \|\nabla L(w^t)\| \leq \|\nabla L(w^{t+1}) - \nabla L(w^t)\|$. Substituting the previous inequality, we get $\|\nabla L(w^{t+1})\| - \|\nabla L(w^t)\| \leq \text{L}\|w^{t+1} - w^t\|$.

Using the gradient descent update $w^{t+1} = w^t - \eta \nabla L(w^t)$, we can write $\|\nabla L(w^{t+1})\| - \|\nabla L(w^t)\| \leq \text{L}\eta\|\nabla L(w^t)\|$. Rearranging the terms, we get $\|\nabla L(w^{t+1})\| \leq (1 - \text{L}\eta)\|\nabla L(w^t)\|$.

Since L and η are both positive, we have $1 - \text{L}\eta < 1$. Therefore, we can conclude that $\|\nabla L(w^{t+1})\| < \|\nabla L(w^t)\|$. This completes the proof. □

Theorem 1. *Given a loss function $L(w)$ under assumptions 1 and 2, w is updated with gradient descent $w^{t+1} = w^t - \eta \nabla L(w^t)$, where t is the iteration number, η is the learning rate, and $\nabla L(w^t)$ is the gradient of the loss function with respect to w^t, the following inequality holds $\|w^{t+1} - w^t\| < \|w^t - w^{t-1}\|$.*

Proof. From the inequality in lemma 1, $\|\nabla L(w^t)\| < \|\nabla L(w^{t-1})\|$, using the gradient descent update $w^{t+1} = w^t - \eta \nabla L(w^t)$, we write $\|w^{t+1} - w^t\| < \|w^t - w^{t-1}\|$. This completes the proof. □

C EVALUATION DETAILS

C.1 DATASET

We evaluate CM with a "super" dataset, similar to a recent FCL work (Yoon et al., 2021), which consists of six frequently used image datasets: SVHN, FaceScrub, MNIST, Fashion-MNIST, Not-MNIST, and TrafficSigns. To simulate different concepts, the "super" dataset is splitted into five concepts. As shown in Table 6, the data in the five concepts differ across a wide spectrum of classes and number of samples. The original FaceScrub dataset has 100 classes. In order to stress test CM, it is splitted into Concept 2 and 3 with 50 different classes each, as we aim to verify whether CM can differentiate them successfully. Since MNIST datasets are easy to learn, we mix the three MNIST datasets together to make it more difficult. The training, test, and validation split of the data follows 7:2:1.

Table 6: Dataset details for each concept

Concept	1	2	3	4	5
Dataset	SVHN	FaceScrub0	FaceScrub1	MNIST, Fashion-MNIST, Not-MNIST	TrafficSigns
No. Classes	10	50	50	30	43
No. Samples	88300	9898	9899	138197	45956

C.2 MODEL

To compare with the baseline fairly, we use the same CNN-based image classification model as (Yoon et al., 2021). We believe this model is ideal in size to learn from the dataset. This model uses two convolutional layers and three dense layers to classify an image input (32*32*3) into one of the 183 classes. The two convolutional layers have 20 and 50 channels, with 5 by 5 filters, stride of 1, and ReLU activation. A 3 by 3 max pooling with stride of 2 follows them. Then, the flattened tensor is fed into three dense layers of 800, 500 and 183 neurons respectively, with ReLU and Softmax activation.

As it is common practice in class-incremental CL, the model follows the single-head evaluation setup (Shim et al., 2021; Mai et al., 2021; 2022), where it has one output head to classify all labels. This setup is ideal for clients with constrained resource capacity in FL, because the clients do not have to spend computation resources on expanding or selecting the output head. Let us note that using the total number of labels as the model output size does not mean we have to know the entire label space or even the label space size, because we can use any output size not smaller than the upper bound of the number of labels. It makes no difference in the training and testing accuracy when experimenting with a given dataset, because the weights associated with unencountered output neurons will not be updated in the backward pass.

C.3 EXPERIMENTAL SETTINGS

We implement CM with TensorFlow and scikit-learn. The experiments are conducted on a Ubuntu Linux cluster (Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz with 512GB memory, 2 NVIDIA P100-SXM2 GPUs with 16GB total memory). Non-overlapping chunks from the five concept datasets are further distributed randomly to the clients. At every round, the local data across clients are non-IID. Each client encounters one of the five local concept datasets randomly, and uses a cyclic sliding window of 320 samples in the encountered concept dataset. Unless otherwise specified, CM is tested with 20 clients (all clients participate in each training round), kmean clustering algorithm, FedAvg aggregation algorithm, and Manhattan distance for the server CM algorithm. For the local training, we use the Adam optimizer with learning rate of 0.001, weight initializer of HeUniform, and batch size of 64. Each client maintains and uses a single Adam optimizer throughout the training for all concepts. We train 15 epochs every round, and use early stopping with the patience value as 3.

We test CM with different hyper-parameters, and only present the results with the hyper-parameters that lead to the best results. The system runs 100 rounds of training for each experiment.

To quantify the overall concept matching accuracy, we evaluate under the class-incremental scenario, and assume the clients are not aware of the concepts and perform the client concept matching every round. In task-incremental scenarios, the difference is that the clients do not have to perform concept matching every round, because they know the task IDs and the concept drift due to the transition of different tasks. For the same experimental setting, task-incremental training would achieve the same model performance with lower computation overhead at the clients.

C.4 IoT Device Setup

CM is evaluated on a Qualcomm QCS605 IoT device. This device is equipped with Snapdragon™ QCS605 64-bit ARM v8-compliant octa-core CPU up to 2.5 GHz, Adreno 615 GPU, 8G RAM, and 16 GB eMMC 5.1 onboard storage. We choose it because its specifications are ideal for AIoT cameras and image-based applications. The device is connected to Internet through WiFi with a bandwidth of 300 Mbps. We re-use CM simulation Python code on the device. Since the device does not support native Linux and its the operating system is rooted Android 8.1, we need to run a Linux distribution on the Linux kernel of Android for easy package management and better support. We achieve this goal with two open source projects: termux-app and ubuntu-in-termux. Termux-app is an Android application for terminal application and Linux environment. It provides some basic Linux commands and packages, but is not on par with a mature Linux distribution, such as Ubuntu. Ubuntu-in-termux bridges the gap. Through it, we are able to install well-maintained Python environments and libraries to execute training on-device. The Python training process can be observed directly under adb shell, which does not include the overhead of Termux-app or Ubuntu-in-termux.

C.5 Parameters for Clustering Algorithms

Kmean, agglomerative, and BIRCH require the number of clusters as a parameter. Unless otherwise specified, we set this parameter to be 5. DBSCAN and OPTICS require some threshold values tuned for the data as parameters instead of the number of clusters. We adhere to the convention when selecting their parameters. For DBSCAN, there are two main parameters: $min\_samples$ and ϵ. $min\_samples$ is the fewest number of points required to form a cluster. We adjust it to be 3, which is a lower value than the average number of clients per concept (20/5). ϵ is the maximum distance between two points while the two points can still belong to the same cluster. To choose ϵ , we firstly calculate the average distance between each point (the model weights of a client) and its 3 ($min\_samples$) nearest neighbors, and then we sort distance values in the ascending order and plot them. We choose ϵ to be 20 as the point of maximum curvature in the plot. Similarly, we set $min\_samples$ parameter to be 3 for OPTICS. The other parameters for these clustering algorithms are the default values in scikit-learn.

C.6 Additional Results

Learning curves for scalability. Figure 3 shows the learning curves as the number of clients increases. The results show that CM can always learn smoothly. Figure 4 demonstrates CM can also learn smoothly with 20 clients as the model size increases or decreases 20%. To stress test CM, we further investigate the model performance when training 80 clients with the network size increased and decreased 20%. The learning curves in Figure 5 demonstrate CM's smooth learning progress, as it achieves 94.3% and 94.9% average accuracy, respectively.

Resilience to number of concepts different from the ground truth. CM requires an estimated number of concepts configured in the bootstrapping phrase. In case the system administrator fails to estimate the number of concept correctly, we experiment with the number of concepts from 3 to 7 under the same experimental settings (i.e., 5 concepts is the ground truth). As shown in Figure 6, when the configured number of concepts is higher (6 and 7) than the ground truth, 5 concept models (out of 6 or 7) learn the corresponding concepts smoothly. The extra concept models in the system do not effect the smooth learning progress, and the average model accuracy achieves 90.5% and 90.0% respectively. When the number of concepts is smaller (4 and 3) than the ground truth, the

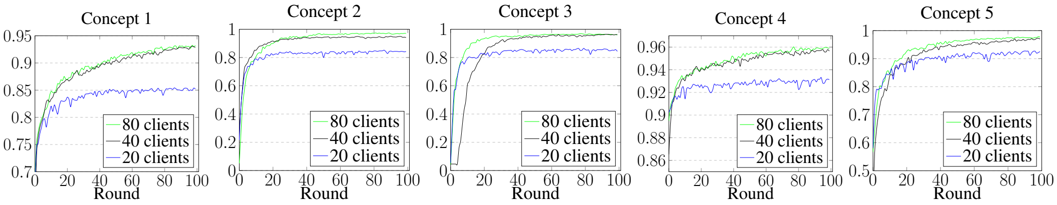

Figure 3: Test set accuracy vs. communication rounds as number of clients increases

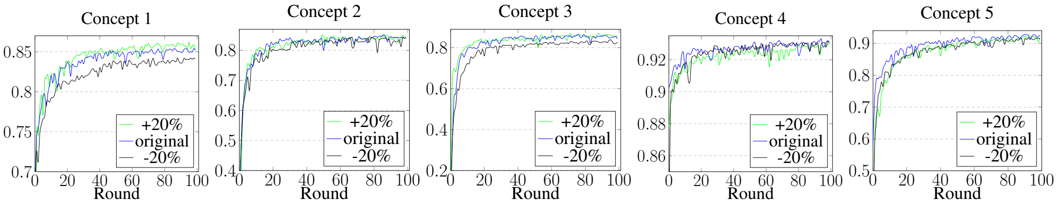

Figure 4: Test set accuracy vs. communication rounds for training 20 clients with different model size

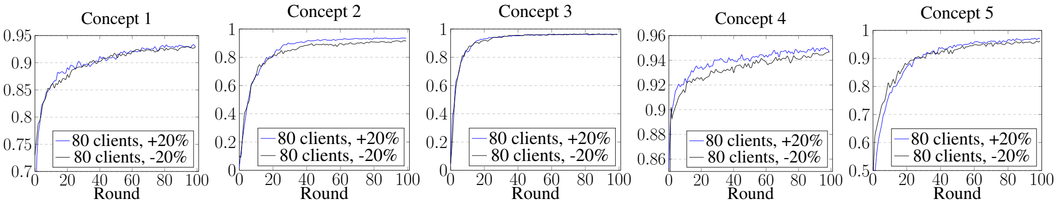

Figure 5: Test set accuracy vs. communication rounds for training 80 clients with different model size

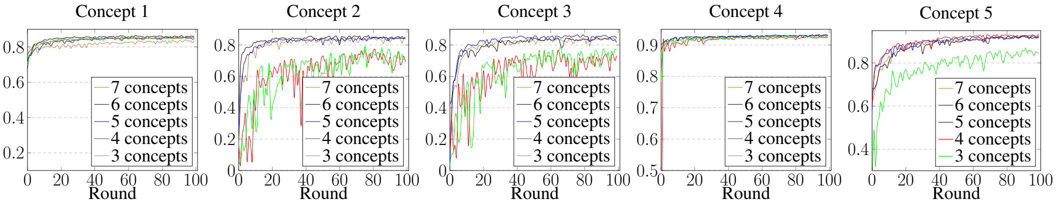

Figure 6: Test set accuracy with different number of concepts over communication rounds

Table 7: Client operation time (second) on real IoT device

	Receiving model	Concept Matching	Training	Sending model	Total
Vanilla FL	0.67	N/A	85.27	0.67	86.61
CM	3.35	8.52	85.27	0.67	97.81

system starts to treat the similar concepts (e.g., two FaceScrub concepts) as one. Although the model accuracy on the affected concepts (2, 3, and 5) exhibit minor fluctuations during the training, the smooth learning progress for the other concepts (1 and 4) is not affected. Nevertheless, the average model accuracy achieves 89.5% and 88.9% respectively, and beats vanilla FL (86.7%). These results demonstrate CM has good resilience in terms of the number of concepts configured in the system. Furthermore, they suggest it is better if system administrators over-estimate the number of concepts, because the performance is still very good in this case.

Client operation overhead. Designed for mobile or IoT devices, CM is evaluated on a real IoT device in terms of the client end-to-end operation time. Compared with vanilla FL, the client operation overhead comes from receiving multiple concept models from the server and the client concept matching. Table 7 shows the breakdown of client operation time on the Qualcomm QCS605 IoT device in one round. Compared with a multi-epoch training process over the entire experienced data, the concept matching can be achieved by testing only a portion of the data. The communication time is calculated as sending or receiving the model(s) of size 25.2MB over 300 Mbps WiFi network. The total client end-to-end operation time in one round is 97.81 seconds, which is feasible for a real-world deployment. Overall, the total CM operation time has a low overhead (11%) over vanilla FL operations on the IoT device. We believe the improvement in performance achieved by CM is worth this overhead cost.