FedCMR: A Library for Federated Continual Model Refinement

Anonymous Authors[1]

Abstract

Machine learning models suffer from performance degradation when out-of-distribution (OOD) data samples, which do not come from their training data distributions, emerge after model deployment. A common practice called continual model refinement (CMR) in machine learning operations (MLOps) can alleviate such performance degradation by continuously refining deployed models over OOD data samples. However, few existing works on CMR tasks have considered federated learning (FL) settings where the OOD data samples are ubiquitous. To support CMR tasks in federated learning scenarios, we present a library called FedCMR, which includes a holistic pipeline that enables end-to-end CMR task evaluation ranging from data selection and labeling to model refinement and evaluation. We further show a case of integrating FedCMR with a federated learning ecosystem backed by the FedML production system (He et al., 2020). We hope that FedCMR could provide an efficient means for developing and evaluating federated CMR algorithms. We will open-source our library upon publication.

1 Introduction

Deployed machine learning models often encounter out-of-distribution (OOD) data samples that do not come from their training data distributions. For example, in a federated surveillance system, object recognition models may see images with lighting conditions or weather that are not included in their training set (Figure 1). Various existing studies (Koh et al., 2020; Gulrajani & Lopez-Paz, 2021; Lin et al., 2022) suggest that deployed models often fail to generalize to OOD data samples and, therefore, suffer from performance degradation. Such performance degradation imposes risks in mission-critical applications due to missing detections of critical events.

A common practice to alleviate such performance degradation of deployed models on OOD data is collecting data samples from data sources (e.g., cameras) with evolving data distributions and continuously refining deployed models using the collected data samples (Figure 1). Concretely, a continual model refinement (CMR) task may repeatedly perform the following steps:

1. Collecting data samples from data sources.
2. Selecting data samples for labeling and storing.
3. Trigger refinement tasks.
4. Produce refined models.
5. Evaluate and select refined models for serving.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the Machine Learning and Systems (MLSys) Conference. Do not distribute.

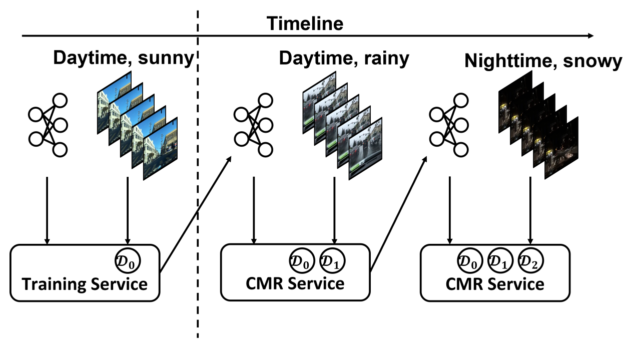


Figure 1. An example of continual model refinement (CMR) task. A model from the training service encounters out-of-distribution (OOD) data samples from different distributions than its training set $\mathcal{D}_0$. The CMR service summarizes OOD data samples into refinement sets $\mathcal{D}_1, \mathcal{D}_2$ and produces refined models that perform better on OOD data samples.

Along this direction, existing libraries suffer from a few critical limitations.

Lack of holistic support. None of the existing libraries (Lin et al., 2022; rostamiz & Yang, 2017; Huang et al., 2022; Koh et al., 2020; Gulrajani & Lopez-Paz, 2021) on CMR tasks includes all five steps in the CMR task pipeline. For example, the CMR library (Lin et al., 2022) focuses on simulating OOD data streams and benchmarking model refinement algorithms but does not include the data selection step (Step 2). The data selection step is important because (1) data sources can constantly produce a high volume of samples that are infeasible to store entirely, and (2) selection strategies can affect the following training step and model performance (Ren et al., 2020).

Table 1. Comparison between our FedCMR library and CMR (Lin et al., 2022), Avalanche (AVL) (Lomonaco et al., 2021), active learning as a service (ALaas) (Huang et al., 2022), WILDS (Koh et al., 2020), and DomainBed (DB) (Gulrajani & Lopez-Paz, 2021) libraries.

		CMR	AVL	ALAAS	ALP	WILDS	DB	FEDCMR
HOLISTIC SUPPORT	FLEXIABLE OOD DATA GENERATION	✓	✗	✗	✗	✗	✗	✓
	DATA SAMPLER	✗	✗	✓	✓	✗	✗	✓
	TASK TRIGGER	✗	✗	✗	✗	✗	✗	✓
	REFINEMENT PARADIGM	✓	✓	✗	✗	✗	✗	✓
	EVALUATION CRITERIA	✓	✓	✓	✓	✓	✓	✓
MODULARITY AND COMPATABILITY	MODULARIZED ALGORITHM	✓	✓	✓	✓	✓	✓	✓
	MODULARIZED DEPENDENCIES	✗	✗	✗	✗	✗	✗	✓
	FL ECOSYSTEM INTEGRATION	✗	✗	✗	✗	✗	✗	✓

Lack of support for federated learning systems. Most of the existing libraries (Lin et al., 2022; Huang et al., 2022) are designed for local simulation or centralized deployment on the cloud and do not support federated CMR tasks.

In this work, we present FedCMR that addresses the aforementioned limitations. The highlights of our library are:

Out-of-the-box functionalities. Our libraries support all five steps in CMR tasks, ranging from data collection to model selection. Scientists and practitioners may directly plugin their algorithms and datasets without additional coding efforts. We also include benchmark datasets and baseline algorithm implementations (Appendix A).

Modularized library design and algorithm dependencies. Our library is modularized and can be easily extended to new refinement strategies. In addition, we design flexible and generic API interfaces for each module, including algorithm dependencies (e.g., data storage) that are common in production machine learning systems. The modularized dependencies can further ease the integration of our library into a federated learning ecosystem.

Federated learning (FL) support. We integrate our FedCMR library into the FedML ecosystem that supports production-level deployments in cross-silo and cross-device settings (He et al., 2020), going beyond local simulations and centralized cloud platforms.

We list examples of integrating our library into federated learning infrastructures and provide empirical results to show the capability of our holistic library on CMR tasks.

2 Related Works

Besides the CMR library (Lin et al., 2022) that is discussed in Section 1, active learning libraries (rostamiz & Yang, 2017; Huang et al., 2022) provide comprehensive support for data selection strategies but mainly focus on improving the model performance over their training sets instead of the OOD data samples. There are also libraries (Koh et al., 2020; Gulrajani & Lopez-Paz, 2021) on OOD generalization tasks that benchmark model performance over OOD data samples (Step 4). OOD generalization tasks assume that OOD data samples are not accessible outside evaluation, differing from CMR tasks that aim to refine deployed models over OOD data samples. In addition, none of the existing libraries support federated learning systems. Table 1 shows a detailed comparison between our library and others.

3 Architecture Design

We first introduce the high-level architecture design of our library. Later, we will show a case of integrating our library into federated learning systems. In what follows, we shall present a basic workflow of our library (Figure 2) and, then, dive into the detailed modular design. The workflow includes modules that are related to the CMR task as well as their dependencies (e.g., data sources and storage). Our library provides a mock module for each dependency to ease simulations outside production systems, including data sources, data storage, training infrastructure, model card, and inference endpoint (Figure 2).

3.1 Workflow Overview

The workflow of a CMR task (Figure 2) starts with data sources that collect data samples (step 1). Later, the collected data samples will arrive at the data sampler. The data sampler operates in a batch manner: for a data batch, the informative and representative data samples are selected based on a selection strategy (step 2). Users may let the selected data samples go to the data labeler for labeling or directly add the unlabeled data to data storage (step 3). Such a data collecting-labeling-storing process is long-running. Algorithm 1 shows a data selection workflow example with a labeling option.

Algorithm 2 summarizes the remaining workflow of our library. The refinement trigger fires once a trigger condition (e.g., 10,000 new samples are labeled) is met (step 4).

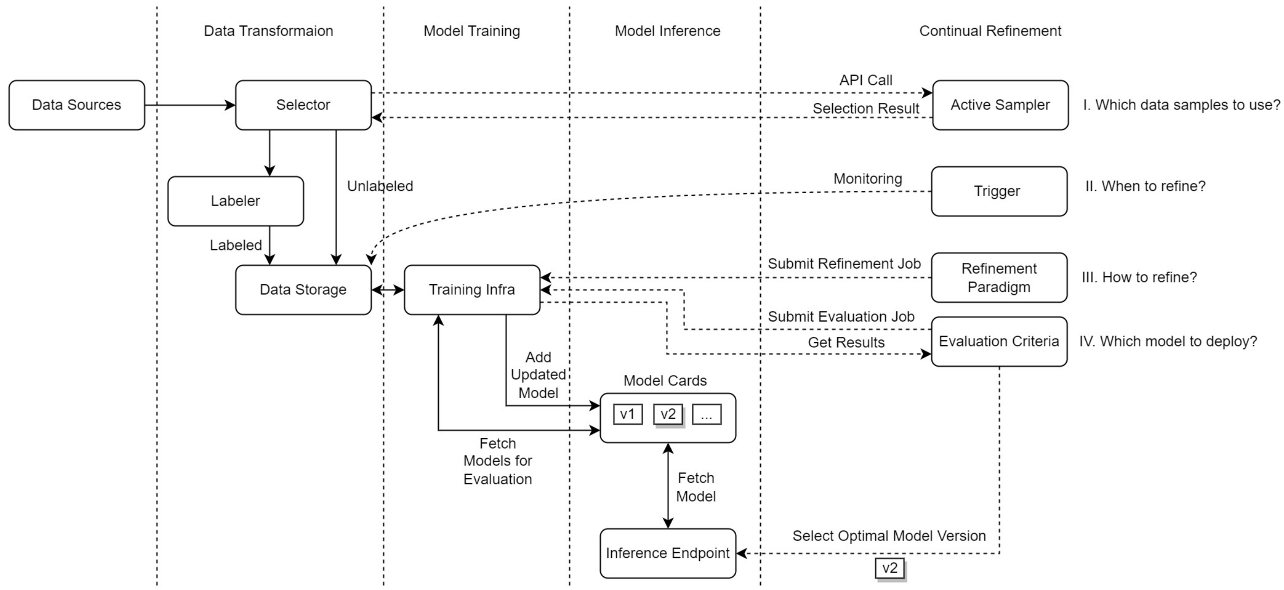


Figure 2. A workflow for continual model refinement (CMR) tasks, which include our library (rightmost column) and its dependencies.

Algorithm 1 Batch data selection workflow

Input: batch size $m > 0$, sampling budget k, option `need_label`
for $i = 1$ **to** ∞ **do**
  $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$ = DataSource.get(m)
  $\{\pi_1, ..., \pi_k\}$ = DataSampler.sample(k, $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$)
  if `need_label` = True **then**
    $\{(\boldsymbol{x}_{\pi_1}, y_{\pi_1}), ...\}$ = Labeler.label($\{\boldsymbol{x}_{\pi_1}, ...\}$)
    DataStorage.append_labeled($\{(\boldsymbol{x}_{\pi_1}, y_{\pi_1}), ...\}$)
  else
    DataStorage.append_unlabeled($\{\boldsymbol{x}_{\pi_1}, ..., \boldsymbol{x}_{\pi_k}\}$)
  end if
end for

To monitor trigger conditions, the refinement trigger may periodically query the data storage for related information. Once the refinement trigger fires, our CMR library submits a refinement job to the training infrastructure (step 5). While completing the refinement job, the training infrastructure interacts with data storage for retrieving selected data samples and updating stored data samples such as replay memory (Lopez-Paz & Ranzato, 2017). Users may directly adopt the built-in refinement job and algorithms in our library or add their own customization. The refined models then go to model cards. Our library then issues test jobs to the training infrastructure and evaluates multiple models from model cards based on the updated data set that includes the newly labeled data samples (step 6). The inference endpoint will pick the model with the best evaluation result for serving (step 7).

Algorithm 2 Continuous Mode Refinement workflow

Input: refinement jobs $\mathcal{J}_{train}$, $\mathcal{J}_{eval}$
for $i = 1$ **to** ∞ **do**
  t = Timer.get_time()
  n = DataStorage.get_data_counter()
  if Trigger.fire(t, n) = True **then**
    TrainingPlatform.execute($\mathcal{J}_{train}$)
    `best_model` = TrainingPlatform.execute($\mathcal{J}_{eval}$)
    InferenceEndpoint.update(`best_model`)
  end if
end for

Comparison. In addition to the advantage of comprehensive support for CMR tasks (Section 2), our workflow is more generic and flexible than those of existing libraries:

- The previous CMR library (Lin et al., 2022) uses scripts to generate and store data streams instead of providing a generic DataSource module. For the same CMR task, it would be valuable to evaluate a CMR strategy over data streams that are generated with different seeds. With a DataSource module, our library can directly handle different seeds. In contrast, the CMR library (Lin et al., 2022) would need users to generate and store multiple data streams, introducing additional overhead.
- The Avalanche library (Lomonaco et al., 2021) assumes that the data storage is a module of a refinement job instead of specifying a generic interface between the refinement job and the data storage. Therefore, implementing a refinement task trigger is difficult in Avalanche because the data storage is not available

until the refinement job is launched. Such an issue makes the Avalanche library less flexible for CMR tasks compared to our library.

3.2 Modularized Design

We introduce each module that is mentioned in the workflow with more details.

3.2.1 Data sources

Data sources iteratively generate data batches or streams that include OOD data samples. Since real-world data streams may not always be available, our library includes a simulated batch OOD data generation approach (Lin et al., 2022) utilizing multiple predefined datasets in addition to the training set. These predefined datasets are not available during training and are therefore considered OOD. The simulated approach (Algorithm 3) picks a major OOD dataset at each iteration using a Markov chain. Then, we mix data samples from the major OOD dataset and those from other datasets. Such a strategy offers control over the OOD level of data batches by specifying how often distribution shifts and how many OOD samples are included.

Algorithm 3 Simulated data source (Lin et al., 2022)

Input: datasets $\mathcal{D}_0, ..., \mathcal{D}_N$, batch size m, previous major dataset index c_0, transition matrix of a Markov chain $\boldsymbol{\beta}$, mixing ratios α and γ
Initialize data batch $\mathcal{D} = \varnothing$.
1. Sample a major dataset index c from a categorical distribution $c \sim \mathrm{Cat}(\boldsymbol{\beta}_c)$
2. $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_0'$ where $\mathcal{D}_0'$ is a random subset with αb elements from the training set $\mathcal{D}_0$.
3. $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_c'$ where $\mathcal{D}_0'$ is a random subset with $(1-\alpha-\gamma)b$ elements from the training set $\mathcal{D}_0$.
4. $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{0\cup c}'$ where $\mathcal{D}_{0\cup c}'$ is a random subset with αb elements from the union of all dataset except for the training set $\mathcal{D}_0$ and the major dataset $\mathcal{D} \cup \mathcal{D}_c'$.
Return $\mathcal{D}$

3.2.2 Data sampler

Data sampler is a common module in modern machine learning pipelines (Haussmann et al., 2020) because storing every data sample from data sources can be infeasible. Our sampler module takes a data batch as input and outputs a set of selected indices. Here, we detail the core-set approach (Algorithm 4) and margin-based strategy (Algorithm 5) as representative examples. The core-set approach aims to select data samples to maximize the coverage of the selected samples over the remaining samples in, for example, Euclidean distance. The margin-based strategy uses models' prediction confidence as criteria, which is measured by the difference between the largest value and the second largest value in prediction logits.

Algorithm 4 Coreset sampler

Input: m data samples $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$, budget k
Initialize $\mathcal{S}$ to be a random subset of $[m] = \{1, ..., m\}$.
repeat
  $u = \arg\max_{i \in [m] \setminus \mathcal{S}} \min_{j \in \mathcal{S}} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$
  $\mathcal{S} = \mathcal{S} \cup u$
until $|\mathcal{S}| = k$

Algorithm 5 Margin sampler

Input:
Input: m data samples $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$, budget k, model f that outputs logits, d classes
Initialize $\mathcal{S} = \varnothing$.
repeat
  $u = \arg\max_{i \in [m] \setminus \mathcal{S}} f(\boldsymbol{x}_i)_p - f(\boldsymbol{x}_i)_q$ where $p = \arg\max_{[d]} f(\boldsymbol{x}_i)$ and $q = \arg\max_{[d] \setminus p} f(\boldsymbol{x}_i)$
  $\mathcal{S} = \mathcal{S} \cup u$
until $|\mathcal{S}| = k$

3.2.3 Data labeler

The data labeler needs to assign labels to selected data samples from the data sampler. In simulations and benchmarking, we may directly use ground-truth labels.

3.2.4 Data storage

The data storage module needs to store and get both labeled and unlabeled data. Our library lets data batches that are selected at different times be stored separately. Separating data batches can ease the model evaluation tasks and provide fine-grained evaluation results over a period of time in addition to their average. The `get` method can take a time argument t to retrieve data batches that are stored before time t. We use a pointer to track which batches are included in previous refinement tasks and provide a `stash` method to update the pointer.

3.2.5 Model cards

A model cards module (Mitchell et al., 2018) needs to store models and their parameters. Each model is indexed by a card and can be retrieved.

3.2.6 Task trigger

The task trigger decides when to launch model refinement tasks. Our current implementation supports time-based and sample-based trigger strategies, which fire a trigger once a given amount of time passes or a certain number of samples are appended.

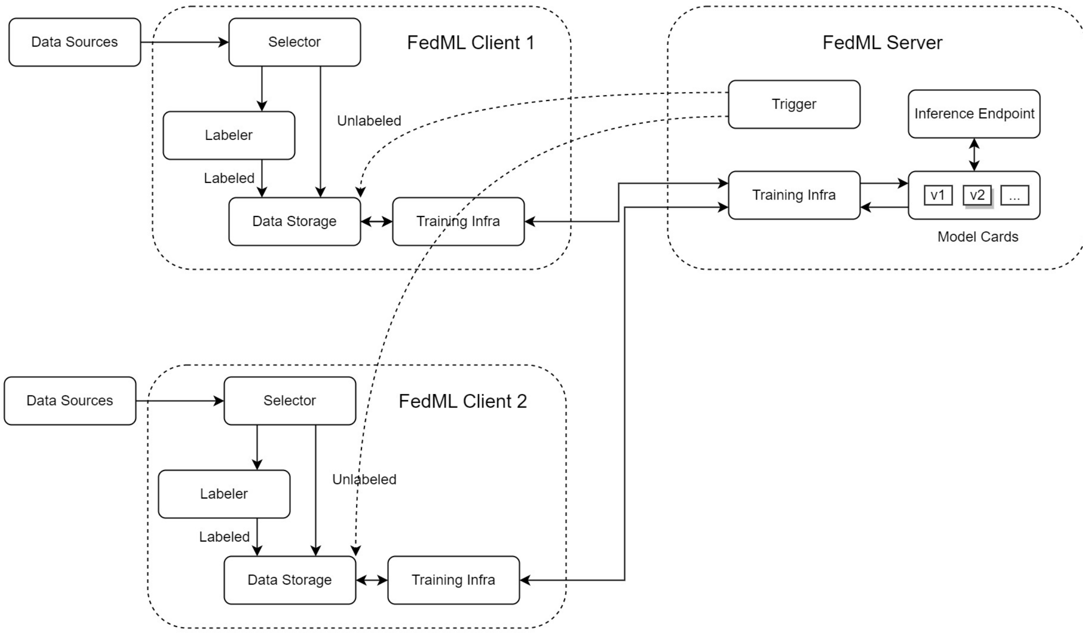


Figure 3. A workflow for federated continual model refinement (FedCMR) tasks, which include our library and the FedML ecosystem (He et al., 2020). We integrate our FedCMR modules into the server and clients from FedML.

Algorithm 6 Refinement job template

Input: current time t, model f, data storage DataStorage, model card ModelCard
1. `recent_batch` = DataStorage.get(t)
2. `replay_buffer` = DataStorage.get_replay_buffer()
3. Refine model f and produce updated model f'
4. Update `replay_buffer`
5. DataStorage.update_replay_buffer(`replay_buffer`)
6. DataStorage.stash(t)
7. `card` = ModelCard.add(f')

Return `card`

3.2.7 *Training infrastructure*

A training infrastructure needs to accept and complete refinement jobs with specified datasets, models, and job scripts. Interacting with the data storage module and the model card module is necessary.

3.2.8 *Refinement paradigm*

A refinement paradigm specifies how to refine a model. Our library provides generic refinement job templates (Algorithm 6). The generic templates specify the interactions between the job and the data storage and model card dependencies, which are necessary and sufficient for common replay-based and regularization refinement paradigms (Lin et al., 2022).

3.2.9 *Evaluation criteria*

We provide job templates for evaluating the accuracies of models over the training set and data batches selected by the data sampler. On benchmark datasets, we also compute accuracies on pre-defined OOD datasets.

3.2.10 *Inference end-point*

Inference endpoints [1] ease model deployments. Our library sends the best model card to an endpoint, which will subsequently fetch the best model from the model card module using the best model card and deploy it.

4 A CASE OF FL ECOSYSTEM INTEGRATION

In this section, we shall present a case (Figure 3) of combining our FedCMR libraries with a horizontal FL system that is supported by the FedML production system (He et al., 2020). The FedML system supports a broad class of hardware platforms, optimization algorithms, and learning paradigms and has applications in real-world scenarios. Specifically, the FedML system adopts a worker-oriented design pattern that allows developers to specify the behavior of each worker (e.g., server or client) in a system. Such a worker-oriented design allows flexible customization of messaging flows in

[1] `https://huggingface.co/inference-endpoints`

any network topology (He et al., 2020). The modularized design of our library, including the dependencies, allows us to directly place each module on the server or client workers with a minimum amount of effort and introduce CMR tasks for real-world deployment (FedML, 2022).

4.1 Framework and Workflow

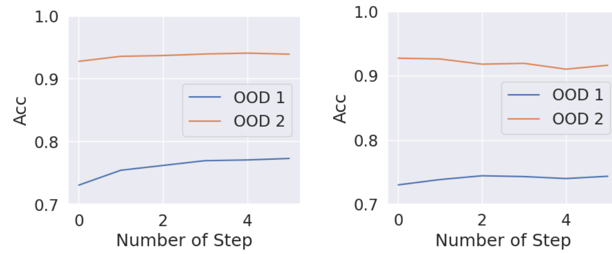
We present a FedCMR framework in Figure 3. Thanks to the modularized design of our library, on the client side, we can directly integrate data sources, data sampler, data labeler, and data storage modules together with the client-side federated learning infrastructure. While clients are performing data selection workflows (Algorithm 1), the server monitors the trigger status and periodically queries clients for necessary information (e.g., the number of labeled new samples on each client). The messaging flow between clients and servers is FedML built-in. Upon the trigger fire, the server launches refinement jobs. The server further dispatches jobs to clients and aggregates a refined global model. Then, the server launches an evaluation job in a similar way to refinement jobs and gets the best model card upon job completion. The best model card will be sent to the inference endpoint to fetch a new model for federated deployment.

5 Experiments

We show the capability of our FedCMR library via experiments in a centralized setting and in a federated learning system with 10 clients. The aggregator in federated learning settings is FedAvg (McMahan et al., 2016). We perform experiments using the Fed_CIFAR10 dataset and a Resnet-18 model that is trained on the Fed_CIFAR10 dataset. We increase and decrease the brightness of Fed_CIFAR10 images to generate OOD 1 and OOD 2 datasets, respectively.

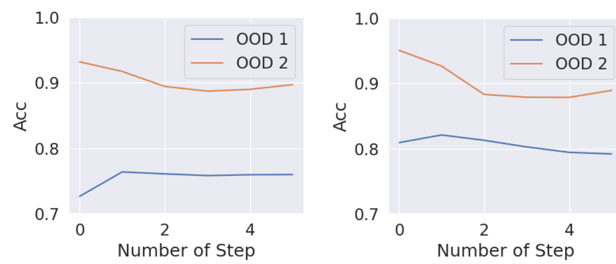
We employ three data samplers (random, coreset, and margin samplers) and two data sources. The first data source only includes a single OOD dataset and the second one mixes OOD datasets according to Algorithm 3. The refinement algorithm is the random memory-replay. Next, we highlight some results that are not covered by existing libraries and benchmarks but are useful for further research.

Refinement is beneficial but not universally. In Figure 4, we can see that experiments with the data source 1 that only uses a single OOD dataset show accuracy improvements by up to on both OOD datasets. In contrast, with data source 2, the increased accuracy over the brighter OOD images (OOD 1) can sacrifice the accuracy over darker OOD images (OOD 2). This result suggests that conflicts between datasets may diminish the refinement benefit. Such conflicts between datasets or tasks are common in multi-task learning settings, and methods from multi-task learning literature (Yu et al., 2020) may help alleviate conflicts.



(a) Data source 1 (b) Data source 2

Figure 4. Accuracy plots of CMR tasks with 5 centralized refinement steps (i.e., the trigger fires 5 times) and two data sources.



(a) Client 1 (b) Client 2

Figure 5. Accuracy plots on two clients in a federated CMR task.

Random sampler is competitive in CMR tasks. Neither of the coreset and margin samplers that are designed for training from scratch settings significantly outperforms the random sampler over the OOD 1 and 2 datasets (Table 2) in CMR tasks. In addition, the coreset sampler yields higher accuracy variation across datasets.

Table 2. Sampler comparison.

Sampler	OOD 1	OOD 2
Random	.743	.916
Coreset	.749	.864
Margin	.752	.920

Federated CMR task needs fairness. In a federated CMR task, some clients may suffer from performance degradation over both OOD datasets (Figure 5b) while others can enjoy refinement benefits (Figure 5a).

6 Conclusion and Future Work

In this paper, we present the FedCMR library that provides comprehensive support for continual model refinement (CMR) tasks and extends CMR tasks to federated learning settings. We will add more datasets, refinement algorithms from various paradigms, and federated learning settings in the future.

References

FedML. Fedml ai platform releases the world's federated learning open platform on the public cloud with an in-depth introduction of products and technologies. https://medium.com/@FedML, 2022.

Gulrajani, I. and Lopez-Paz, D. In search of lost domain generalization. In *International Conference on Learning Representations*, 2021.

Haussmann, E., Fenzi, M., Chitta, K., Ivanecky, J., Xu, H., Roy, D., Mittel, A., Koumchatzky, N., Farabet, C., and Alvarez, J. M. Scalable active learning for object detection. In *2020 IEEE intelligent vehicles symposium (iv)*, pp. 1430–1435. IEEE, 2020.

He, C., Li, S., So, J., Zhang, M., Wang, H., Wang, X., Vepakomma, P., Singh, A., Qiu, H., Shen, L., Zhao, P., Kang, Y., Liu, Y., Raskar, R., Yang, Q., Annavaram, M., and Avestimehr, S. Fedml: A research library and benchmark for federated machine learning. *ArXiv*, abs/2007.13518, 2020.

Huang, Y., Zhang, H., Li, Y., Lau, C. T., and You, Y. Active-learning-as-a-service: An efficient mlops system for data-centric ai. *ArXiv*, abs/2207.09109, 2022.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Beery, S., Leskovec, J., Kundaje, A., Pierson, E., Levine, S., Finn, C., and Liang, P. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, 2020.

Lin, B. Y., Wang, S., Lin, X. V., Jia, R., Xiao, L., Ren, X., and Yih, W.-t. On continual model refinement in out-of-distribution data streams. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*, 2022.

Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T. L., Lange, M. D., Masana, M., Pomponi, J., van de Ven, G., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara, S., Parisi, G. I., Cuzzolin, F., Tolias, A., Scardapane, S., Antiga, L., Amhad, S., Popescu, A., Kanan, C., van de Weijer, J., Tuytelaars, T., Bacciu, D., and Maltoni, D. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop, 2021.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *NIPS*, 2017.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., and Gebru, T. Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2018.

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. A survey of deep active learning. *ACM Computing Surveys (CSUR)*, 54:1 – 40, 2020.

rostamiz and Yang, Y. D. Active learning playground. https://github.com/google/active-learning, 2017.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. Gradient surgery for multi-task learning. *ArXiv*, abs/2001.06782, 2020.

A Baselines and Benchmarks

Tables 3, 4, and 5 list data sampler baselines, refinement paradigms, and benchmark datasets, respectively. We are actively developing our library and adding more baselines and datasets.

Table 3. Data sampler.

SAMPLER	CONFIDENCE-BASED	CLUSTERING-BASED
RANDOM	✗	✗
CORESET	✗	✓
MARGIN	✓	✗
MC-DROPOUT	✓	✗
INFORMATIVE AND DIVERSE	✓	✓
GRAPH DENSITY	✗	✓

Table 4. Refinement paradigm.

REFINER	SUPERVISED	REPLAY-BASED
FINE-TUNING	✗	✗
MEMORY-REPLAY	✓	✓
PSEUDO-LABELING	✗	✓

Table 5. Benchmark datasets.

DATA SET	DATA TYPE	TASK	OOD TYPE
CIFAR10	IMAGE	IMAGE CLASSIFICATION	BRIGHTNESS
FED_CIFAR10	IMAGE	IMAGE CLASSIFICATION	BRIGHTNESS
COCO	IMAGE	OBJECT DETECTION	BRIGHTNESS
IWILDCAM	IMAGE	IMAGE DETECTION	CAMERA ANGLE, WEATHER, ETC