Improving Collaborative Inference: Adversarial and Ensemble Learning Approaches in Distributed Machine Learning

Selim Mouaffak
mohamed.mouaffak@epfl.ch

Supervisors : Dr. Rafael Pires & Akash Dhasade
Professor : Prof. Dr. Anne-Marie Kermarrec

Scalable Computing Systems, SaCS
École Polytechnique Fédérale de Lausanne, Switzerland

Abstract

Traditionally, Federated Learning (FL) is considered the main approach when it comes to training machine learning models in distributed settings. Recently, Collaborative Inference (CI) has emerged as an attractive alternative to solve this problem, but it fails to match the performances of FL. This can be attributed to the inherent design of FL where a global model is trained that can *indirectly* leverage all the data of the clients. In opposition, in CI, we only rely on the individual models of the clients, that have been trained on significantly less data. In our work, we present two methods that aim at mitigating this problem by ameliorating the quality of the predictions at the client side. The first is based on membership inference attacks in a adversarial setting, while the second involves ensemble learning. We will perform extensive evaluations of both methods on the CIFAR-10 dataset. Our findings reveal that our schemes are competitive with traditional techniques while offering substantial practical advantages. Most notably, we manage to outperform state-of-the art CI techniques in high diversity data regimes, while allowing for seamless joining or leaving of the network.

1 Introduction

The rise in data generation and the increasing popularity of edge computing have led to a growing adoption of distributed settings. These systems distribute data storage and processing across multiple locations, offering numerous advantages. However, effectively training machine learning algorithms in these environments becomes crucial. The primary challenge is to leverage distributed systems while maintaining the efficiency and accuracy of machine learning models.

Federated learning (FL) emerges as a powerful solution to the challenges posed by distributed settings [5]. Gaining notable attention and application, this technique fundamentally alters the traditional machine learning landscape by decentralizing the learning process. In essence, federated learning is a machine learning approach that trains an algorithm across multiple devices or servers holding local data samples, without exchanging their data. This method operates by sending a model to each participating client, which is then trained on local data. After training, only the model updates are sent back to a central server. These updates are then aggregated to create an updated global model, which is sent back to all participating devices. This cyclical process repeats, enhancing the global model with each iteration. Federated learning thus enables machine learning algorithms to benefit

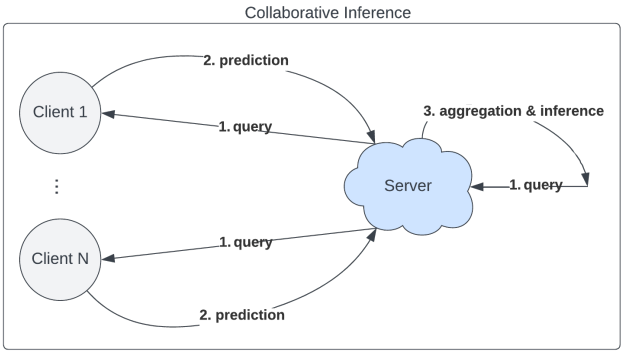


Figure 1: Collaborative Inference pipeline in action.

from diverse datasets spread across distributed settings, without needing to directly access or move the data. This ensures data privacy, making it a particularly suitable technique for distributed systems.

More recently, new paradigms for collectively training machine learning models in distributed settings have emerged. Most notably, collaborative inference (CI) is becoming an attractive alternative to federated learning. While the latter focuses on collaborative training and local inference, the former has traded this paradigm for local training and collaborative inference. More precisely, collaborative inference advocates for clients to train their model locally. Then for a given query, the central server asks each client for its prediction, and then combines them using a well defined aggregation scheme. Refer to figure 1 for further details about the CI pipeline and its functioning. Collaborative inference bears many benefits over more traditional federated learning techniques. Notably, it reduces the communication load by a factor of $10\times$ by saving up on the collaborative training communication cost. Moreover, it supports model heterogeneity as the training is done locally and independently of the other clients. Lastly, it allows for clients to join or leave the network freely. While doing so in a federated learning setup would result in the retraining of the global model, it appears seamless in collaborative inference, as no global model is learned. Regrettably, such schemes suffer a loss in performance compared to FL counterparts and our work aims at improving collaborative inference.

Recent work on collaborative inference has primarily focused on devising sophisticated aggregation schemes for implementation at the server side. That is, the main focus for improvement was at the server, while leaving potential improvements on the client side unexplored. In this context, our works presents new schemes that aim at improving the performance of collaborative inference by focusing on the client side. The main rationale behind this idea is that by producing better predictions at the clients' side, we are able to provide better inputs to the server, and thus improve our final inference power.

Summary of contributions. *(i)* We design and investigate two schemes that aim at improving the performances at the client side in collaborative inference. *(ii)* We extensively test these methods on the CIFAR-10 dataset while specifying our experimental setup, in different data regimes. Our findings reveal that better training procedures at the client side effectively help at increasing the performance of collaborative inference. *(iii)* We compare the performances of our methods against state-of-the art techniques in federated learning and collaborative inference. Our analysis shows that our methods present *significant practical advantages* while being able to compete with the performances reached by those techniques.

2 Collective learning in a distributed setting

This study delves into a distributed setting, wherein a network comprising N clients collaboratively aims to acquire a classification rule. Each client i possesses a dataset denoted as D_i, consisting of input-output pairs belonging to the set $\mathcal{X} \times \mathcal{Y}$. It is worth noting that these datasets may contain sensitive information, and clients may exhibit varying levels of willingness to disclose their data. The primary objective is to derive a parametric function h_θ that minimizes the expected loss across the combined datasets $\hat{D}$, representing the union of all clients' datasets. Mathematically, this objective can be formulated as follows:

$$\min_{\theta \in \Theta} \frac{1}{|\hat{D}|} \sum_{(x,y) \in \hat{D}} \ell(h_\theta(x), y) \tag{1}$$

Here, ℓ denotes a loss function that quantifies the dissimilarity between the predicted outputs, $h_\theta(x)$, and the true labels y. The optimization problem seeks to find the optimal set of parameters, $\theta \in \Theta$, that minimize the average loss over the entire dataset $\hat{D}$. The parameterized function $h_\theta(x)$ represents the classification rule, responsible for mapping input instances, x, to predicted outputs.

The outcomes of this research contribute to advancing our understanding of learning in distributed environments, offering insights into privacy preservation, collaborative model training, and optimization techniques that facilitate effective knowledge sharing among networked clients.

2.1 Federated learning

Traditionally, federated learning has served as the primary approach for addressing the aforementioned challenge. This approach involves a trusted central server responsible for aggregating the knowledge acquired by individual learners or clients. In such setups, clients engage in continuous communication with the server over a fixed number of rounds denoted as T. Initially, the server distributes a shared model architecture to each client, who then perform local model updates based on their respective private datasets. These updates are subsequently transmitted to the server, which aggregates them to generate a global update to be applied by the clients [5].

More formally, let θ_i^t represent the parameter set learned by client i at time t. The server computes the overall update using the following procedure:

$$\theta^t = \frac{1}{N} \sum_{i=1}^{N} \theta_i^t \tag{2}$$

Consequently, the trusted central server can acquire knowledge and learn a global classification rule that has been indirectly trained on the combined dataset $\hat{D}$, all while preserving data privacy. Importantly, this scheme differs from collaborative inference in a crucial aspect. By leveraging collaborative training instead of local training, we can train a model that has been indirectly exposed to a broader range of data samples, leading to improved performance. This intuitive advantage underscores the significance of collaborative inference.

Finally, considering the well-established popularity of this extensively tested method, it will serve as a basis for comparison in our work. By examining the performance and characteristics of our proposed approaches in relation to the traditional federated learning framework, we can gain valuable insights into its effectiveness and potential advancements. This comparative analysis will enable us to assess the strengths and limitations of our methods, providing a comprehensive evaluation of their contributions and setting the stage for further advancements in collaborative learning techniques.

2.2 Collaborative inference

Collaborative inference (CI) emerges as a compelling alternative to federated learning, offering a paradigm shift by trading collaborative training and local inference, while advocating for local training and collaborative inference. In this approach, clients undertake local training on their respective private datasets, D_i. Consequently, a central trusted server leverages an aggregation rule to combine the individual inferences generated by the clients into a final prediction.

Let h_θ^i denote the classification rule learned by the i^{th} client, and f^{server} represent the aggregation method employed by the server. Previous research in this field has explored a range of aggregation methods to be executed by the server, encompassing both trainable and non-trainable approaches. Here, trainable approaches refer to parametric schemes that require rounds of training before becoming operational and are therefore denoted by f_λ^{server}. In opposition, non-trainable methods are averaging schemes that combine the individual inferences in a deterministic fashion. Building upon this foundation, this paper extends the existing work by considering the most effective collaborative inference scheme as the baseline for further comparison. By benchmarking our proposed method

against this state-of-the-art collaborative inference approach, we can comprehensively evaluate its performance and gauge its potential for advancing the field.

CI aggregation via Averaging. The landscape of CI research has been explored through meticulous testing and comprehensive evaluation methodologies, striving to design the most optimal aggregation schemes. A straightforward strategy in this context is the elementary averaging scheme, leading to a final inference expressed as follows:

$$f^{server}(x) = \frac{1}{N}\sum_{i=1}^{N} h_\theta^i(x) \tag{3}$$

Neural network aggregator in CI. In their seminal work, Dhasade *et al.* introduced a state-of-the-art approach that allows the central server to learn an aggregation rule via a trainable neural network (NN). According to their findings, this method markedly surpasses other trainable or non-trainable schemes for this purpose. The NN, in this case, takes charge of learning an aggregation rule, combining the individual predictions offered by the clients to generate a final inference that effectively addresses the classification problem. Formally, given an appropriate loss function ℓ_{NN}, the server aspires to learn a rule f_λ^{server} that solves the following optimization problem:

$$\min_{\lambda\in\Lambda} \frac{1}{|D_{server}|} \sum_{(x,y)\in D_{server}} \ell_{NN}(f_\lambda^{server}(h_\theta^1(x),\cdots,h_\theta^N(x)), y) \tag{4}$$

This scheme necessitates training a neural network at the central server, giving rise to the following implications:

- **Requirement of Additional Data:** The process of training a neural network aggregator introduces a demand for supplementary data, denoted by D_{server} in equation 4. While synthetic generation or sourcing from public databases is possible, this study assumes clients are willing to sharing a non-invasive fraction of their data. This data is beneficial for training the NN as it originates from the same distribution as client data. Moreover, this assumption is critical for maintaining viable comparative schemes that avoid using externally sourced data. In section 3.1, we dive into further detail, explaining how we obtain this dataset.
- **Dependency on Clients:** Opting for a trainable scheme engenders a previously non-existent dependency on clients. Unlike non-trainable methods, which permit clients to seamlessly join or leave the network without impacting server operations, the NN aggregator introduces a dependency. Specifically, the addition or removal of a client now necessitates the re-training of the aggregator.

2.3 Robust predictions in collaborative inference

2.3.1 Motivation

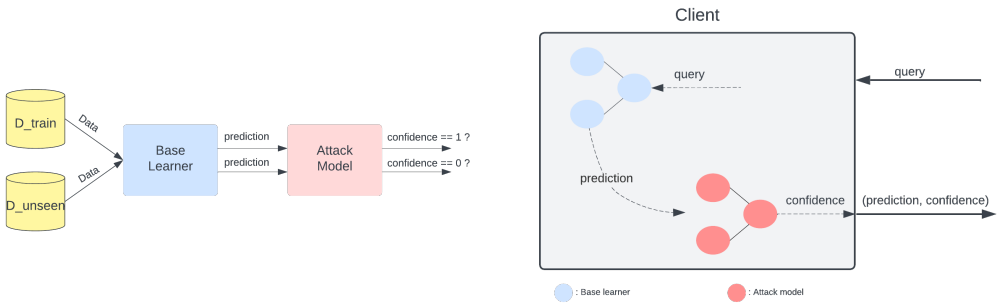
Collaborative Inference (CI) is being showcased as a robust alternative to traditional Federated Learning (FL), bearing several advantages. However, a notable challenge remains: matching the accuracy levels achieved by FL. This can be intuitively attributed to the inherent design of FL, where the global model indirectly accesses all client data, contrary to CI, where each learner exclusively trains on their individual data. This incurs a massive conceptual difference as machine learning model have historically suffered from hallucinations [4]. That is, when presented with an unseen data sample that does not belong to the training distribution, models tend to produce overconfident, wrong, predictions.

While prior research in this domain has primarily concentrated on developing better aggregation schemes to improve CI's competitiveness against FL, this paper addresses the aforementioned challenge. We propose methods to alleviate issues at the learner's end, thereby shifting our focus to the client side, while adopting simpler aggregation schemes on the server side.

In the following sections, we will introduce two distinct schemes designed for implementation at the client level. These aim to produce predictions that are not only robust but also calibrated with

confidence. Robustness and calibration here refer to predictions that are confident when they are right, and that exhibit uncertainty when in doubt. The idea is therefore to design smart schemes that allow us to determine which learners are relevant to ask on a given query. And producing robust predictions will effectively allow us to determine, given a prediction, if the learner is confident our in doubt. The underlying principle of our work is that enhancing the quality of inputs fed into the server's aggregation scheme can drive significant improvements in the CI framework.

2.3.2 Membership inference and attack models



(a) Training of the attack models.

(b) MI handling a query.

Figure 2: Showcasing some important steps in the Membership Inference pipeline.

Motivation. This technique is motivated by the work of Shokri *et al.* [7], on membership inference (MI) attacks. In their paper, they advocate the use of attack models in a adversarial-like setting, to infer whether or not a record was part of the training set of a machine learning model, given black-box access to it. While our problem differs from the exact setup presented in their work, their is still an underlying shared idea of trying to determine if the model is knowledgeable about a given record or not.

Description of the method. We adapt the described idea to our work in the following fashion. We remember that our goal is to enhance the robustness of the prediction at the client side. That is, we want to know whether or not a client is knowledgeable about a data record. Hence, the strategy is to create and train, for each client, an attack model alongside the already existing model that they have. We will refer to such schemes as membership inference (MI) based methods. The attack model's objective is to learn how to analyze the output prediction of the client and to produce a scalar value, between 0 and 1, expressing how confident and robust the original prediction is. As clients produce probability distributions as their output, the attack models have to effectively learn how to analyze them. The server side aggregation can now leverage both the original prediction of the client plus the confidence value of each client to produce its aggregation. A detailed explanation of this scheme can be found in figure 2. Based on what we have described, we expand on what it means for a model to be knowledgeable about a data record :

- **Class inference:** One possibility is to train the attack models to be able to recognize the classification labels seen during training. That is, the attack model is able to distinguish between predictions that have been generated by a classification label that was seen during the training or not. In doing so, the attack model is effectively capturing the real of knowledge of the learner. Conceptually, the attack model would output 1 if it believes that the queried record belongs to a classification label that the learner has seen in training, and is therefore knowledgeable about. Conversely, the attack model outputs 0 if it believes that the prediction of the client was generated by a data record that it is not knowledgeable about, and thus outside the set of classes the learner has seen in training.
- **Sample inference:** Another possibility is to train the attack models similarly to what has been suggested in the paper. That is, we train the models to distinguish between probability distributions that have been generated by a data record that belongs to their training set or not. Here again, the attack models are tasked to detect if the client's output is worthy to take into account or not.

Training of the attack models. Firstly, the training of the attack models requires the base learners (i.e. the clients' models) to be already trained. Then, we can craft a pipeline that would allow the attack models to learn how to analyse the outputs of the base learners. Both *class* and *sample* inference variations would require the need for extra data that is disjoint from the client's training data. Indeed, in order to correctly train our binary classifiers, we need to present them with both positive records (seen during training) and negative samples (not seen during training). For now, we assume that the clients have a separate dataset, D^i_{unseen}, that contains samples unseen during training. In Section 3.1, we explain how we derive this dataset while enabling a fair comparison with other methods. Lastly, in section 3.2.1, we will further detail the loss criterion ℓ_{attack} that we use. Algorithm 1 fully explains the training procedure of the attack models.

Algorithm 1 Train attack model i

Initialize $D_{\text{attack}} = []$

for each data in D_{train} **do**
    $pred \leftarrow BL(data)$
    $label \leftarrow 1$ ▷ Positive sample because it was seen during training
    D_{attack}.append($[pred, label]$)
end for

for each data in D_{unseen} **do**
    $pred \leftarrow BL(data)$
    $label \leftarrow 0$ ▷ Negative sample because it was *not* seen during training
    D_{attack}.append($[pred, label]$)
end for

for each $(pred, label)$ in D_{attack} **do**
    $confidence \leftarrow$ attack$(pred)$
    Minimize $\ell_{\text{attack}}(confidence, label)$
end for

end procedure

Server side aggregation. Finally, the last step of the pipeline is to specify the server side aggregation scheme. For MI based methods, we will restrain the analysis to two simple aggregation strategies that we detail below :

- **Averaging:** The first scheme that we describe is a simple averaging procedure. By that, we mean that the final inference is obtained through of a weighted average of each client's prediction by it's confidence value. More formally, we denote by $\pi_i(x)$ the i^{th} client's prediction on input x, and by $c_i(x)$ the confidence value, then the final inference is generated by :

$$f^{server}(x) = \frac{1}{\sum_i c_i(x)} \sum_i c_i(x)\pi_i(x) \tag{5}$$

  This scheme will effectively serve as a baseline for comparison with the other strategies.

- **Softmax weighted averaging:** This second scheme is very similar to the one described earlier. The main difference resides in the preprocessing of the confidence values before using them as weights for the averaging. More precisely, we apply a softmax function across the confidence values to scale them into the $[0, 1]$ interval. Formally, we obtain the following procedure :

$$w_i = \frac{e^{c_i}}{\sum_j e^{c_j}} \tag{6}$$

$$f^{server}(x) = \frac{1}{\sum_i w_i(x)} \sum_i w_i(x)\pi_i(x) \tag{7}$$

From what is mentioned above, it entails that MI based methods account for 4 variations, as we investigate both class and sample inference, coupled with simple averaging and softmax weighted

averaging. In Section 3.2, we will go over extensive testing of these schemes while providing our exact experimental setup.

2.4 Ensemble learning and predictive uncertainty estimation

Motivation. This idea was adapted from the work of Lakshminarayanan *et al.* [2]. In their paper, they advocate the use of ensembles, stating how they improve the predictive uncertainty estimations in the context of neural networks. More precisely, they state that training ensemble will produce predictions that are confident when they are right, while exhibiting uncertainties when in doubt. This can be directly measured from the probability distribution that is the output of the model. For this purpose, they use entropy as a measure of uncertainty. That is, predictions are considered confident when the probability distribution has low entropy and the model is certain about its answer. In opposition, they will exhibit a high entropy when the model is uncertain.

From a technical standpoint, the paper suggests the use of the *Ensemble PyTorch* library [1] that comes with a functional ensemble learning framework. In our work, we adopt the use of the *Voting Classifier*, which is a model encapsulating the ensemble. It effectively trains an ensemble model before aggregating their prediction using a simple voting scheme. In the following sections, we will be using this abstraction as a black-box placeholder for our clients' models.

This strategy aligns with our objective of producing more robust predictions, and we can leverage ensemble training to produce well calibrated predictions at the client side, allowing us to estimate the confidence of the client.

Description of the method. In our work, we adopt and adapt the aforementioned idea in the following way. Every client now trains an ensemble of base learners instead of just one. As this will produce robust predictions, we can estimate how confident the client is about his prediction and leverage this additional information in our server side aggregation scheme. While incurring a heavier training load, this method requires no additional data than the data available at the client.

Having detailed the strategy at the client side, we will present in the following aggregation schemes that effectively leverage the robust predictions of the ensembles :

- **Averaging:** The most straightforward idea to implement at this stage is a plain averaging of the predictions. This is based on the underlying principle that more robust will naturally help improve the inference accuracy, as models will tend less to produce overconfident wrong predictions that can hurt performance. This scheme will serve as a baseline for further comparison.
- **Entropy weighted average:** Another idea here is to exploit the inherent robustness of the predictions of the ensembles. That is, we measure the entropy of the probability distribution that we get as output. The entropy will effectively act as a measure for the confidence of the prediction, and can help us scale the predictions of the clients. Clients that are confident in their prediction will be assigned a greater weight than the clients that are in doubt. More formally, if we denote by $\pi_i(x)$ the prediction vector of the i^{th} client on input x, then $\pi_i(x)[j]$ refers to the j^{th} component of the probability distribution. We can compute the entropy of the distribution, and the averaging weights $v_i(x)$ as follows :

$$Entropy(\pi_i(x)) = -\sum_j \pi_i(x)[j] \log(\pi_i(x)[j]) \tag{8}$$

$$v_i(x) = e^{-Entropy(\pi_i(x))} \tag{9}$$

  The exponential scaling along with the negative sign effectively ensure that higher entropy values (more uncertainty) result in lower weights. Finally, we obtain the following aggregation scheme :

$$f^{server}(x) = \frac{1}{\sum_i v_i(x)} \sum_i v_i(x)\pi_i(x) \tag{10}$$

In section 3.3, we will perform exhaustive testing of these methods across different data regimes.

3 Experimental section

We present different evaluations of our methods using the CIFAR-10 dataset. In what follows, the performances of our schemes will be compared against the best performing techniques in both FL and CI.

3.1 Data partitioning

For all experiments in this section, we consider a setting with 12 clients on the CIFAR-10 dataset. This dataset comprises a training set of 50 000 samples and a testing set of 10 000 samples. We report our accuracies on the original test split referred to as D_{test}.

K-Shards partitioning. We further partition the original training set, referred to as D_{train} using K-Shards partitioning `Shards(K)`, as done in several previous works [9]. The parameter `K` controls the number of shards into which D_{train} is divided. Shards are an ensemble of consecutive data points, sampled starting from a random point. Hence, higher values of `K` imply that the data is divided into a higher number of shards. In this scenario, clients are exposed to more diverse data, but they get to observe a lesser number of examples per classification label. In opposition, lower values of `K` mean that the clients are exposed to clustered data but get to see a high number of datapoints per classification label. The effect of changing `K` is visualized in figures 3 and 4 . After the partitioning, each client i is assigned a dataset that is finally divided using a 80%-20% split, into respectively a training set D^i_{train} and a validation set $D^i_{validation}$.

NN aggregator. As discussed in previous sections, the best performing CI inference techniques involves a parametric trainable aggregator. This aggregator, located in a central trusted server, requires additional data to be trained. In light of that, we decide to allocate each client's validation set for this purpose. That is, the training set for the aggregator neural network D^{NN}_{train} is $\bigcup_i D^i_{validation}$.

MI class and sample. As explained in previous sections, our MI based methods account for two variations, class and sample, that entail small differences in data partitioning, in opposition to the other methods. In order to train the attack models, we need extra data that must be allocated from the CIFAR-10 dataset, as to establish a fair comparison. Moreover, and for both variations, we need to present the attack models with positive and negative samples. Positive samples can simply be obtained from the client's training set but negative samples need to obtained otherwise. In light of that, the dataset used to train the attack models is gathered from the validation sets of the other clients. We assume that the clients accept to share a small portion of their personal data, accounting for their validation set, as it considered to be non-privacy invasive. Hence, for each client i, we can gather the remaining validation sets as $\bigcup_{j\neq i} D^j_{validation}$. As this dataset is often far greater than D^i_{train}, and to avoid bias in our binary classifier, we downsample the union of validation sets to be roughly the same size as the original training set. Moreover, in class methods, we filter the union of validation sets to exclude any label that also exists in the client's training set. This is done to effectively have two datasets, one with the positive samples and the other with the negative samples. In sample methods, the filtering step is omitted as any datapoint that is not in D^i_{train} is a negative sample. Finally, the training set for the attack model i is obtained as $D^i_{train} \cup (\bigcup_{j\neq i} D^j_{validation})$.

3.2 MI methods

3.2.1 Setup and hyperparameters

We use LeNet [3] as the backbone base learner and a fixed batch size of 16 across all our experiments. Other parameters for each setup are described below.

CI averaging. Each client performs local training for 50 epochs using SGD as the local optimizer. The learning rate is set to 0.0025. As described in previous sections, our baseline refers to local training for each of our 12 clients. The collaborative final inference is obtained by simple averaging of their individual predictions, as explained in section 2.2.

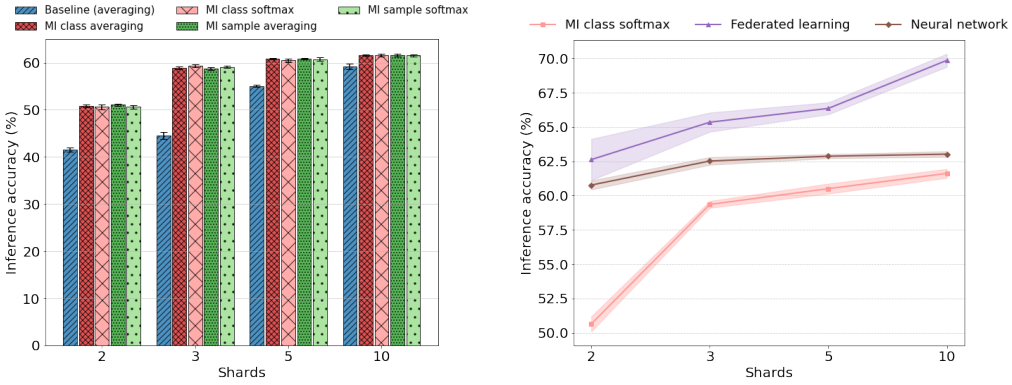
CI neural network aggregator. For the NN aggregator, we rely on a simple multilayer perceptron with 1 hidden layer having 120 neurons each using a ReLu activation with a final softmax output layer. The aggregator is trained for 20 epochs, using the ADAM optimzer with a learning rate of 5×10^{-5}. It further adopts an exponential decay with the parameter $\gamma = 0.999$. Finally, the loss is computed using a standard Cross Entropy loss criterion [10].

Baseline FL. For comparison with FL, we consider a standard FEDAVG algorithm [5]. We train it for 50 communication rounds, with a learning rate of 0.01. Each client performs 2 local training epochs per round.

MI methods. While trained on different datasets, the attack models adopt the same architecture. We use a simple binary classifier with 1 hidden layer of 32 units each using ReLu activations and a final sigmoid output layer. The attack models are trained on the Binary Cross Entropy loss criterion [6], using a SGD optimizer with a learning rate of 0.01, for 30 epochs.

3.2.2 Results

All of the following experiments are run 3 times and are presented with error bars to account for the variance.



(a) (b)

Figure 3: **(a) MI methods:** MI based methods perform significantly better than baseline, especially for low shards. **(b) MI vs FL:** NN aggregator and FL both outperform MI based methods. The gap between the NN aggregator and MI closes for high value of shards as they reach similar accuracies.

MI vs CI averaging. Figure 3a shows the inference accuracies of all variations of MI based methods against our baseline, for different values of shards. We observe that all MI methods comfortably outperform the baseline, where the difference is more staggering for low values of shards. For shards ≥ 5, the difference reduces while the MI methods still perform better. This suggests that the attack models are able to perform better when presented with more examples per classification label. In this scenario, they are able to generalize better and help produce more robust inferences. In higher shards settings, the attack models fail to generalise as they are presented with more diverse data with less samples per classification label, and the gap with the baseline reduces. It is important to note the overall increasing trend that all methods share. This suggests that the backbone learners perform better when they get a sense of all the classification labels that they can be queried on. This makes sense as models tend to hallucinate on unseen data, which is what our attack models try to mitigate. Remarkably, all MI based methods seem to perform similarly. We choose MI class inference with softmax weighted averaging approach for comparison against the best performing methods.

MI vs NN aggregator and FL. In figure 3b, we compare the MI class softmax approach against the best performing CI method (NN aggregator) and FL. Firstly, we observe that FL still outperforms both methods by a significant margin. When comparing MI class softmax with the NN aggregator, we first observe that the latter achieves better accuracies for low values of shards. For instance, when shards = 2, the gap between the two schemes reaches 10%. This suggests that the neural network is

more able at capturing the knowledge of the base learners when data diversity is low at the client side. Nevertheless, and as we increase the number of shards, we can see that MI based methods are able to catch up, as the gap between the two schemes now closes at 1.4% when shards = 10. This indicates that for more diverse data regimes, the attack models are showcase a better ability to predict the confidence of predictions.

Discussion. Overall, figure 3 clearly pictures MI based methods as a robust alternative to our best performing CI inference approach. This appears to be the case especially in high data diversity settings, where MI class inference with softmax aggregation only performs $\sim 1\%$ worse than the NN aggregator. The conclusion from this experiment are the following :

(i) *Training cost:* The MI based methods carry a heavier training load as they require the training of N attack models, whereas the NN aggregagtor only requires the training of one neural network.

(ii) *Performance:* While comparative performance varies depending on the data regime at hand, MI based methods appear to be very attractive alternatives to the best performing CI method when data is diverse but sparse.

(iii) *Practicality:* Both methods exhibit the need for additional data to train either the attack models or the NN aggregator. Therefore, both methods are level in this regard. Nevertheless, We advocate the use of MI based methods over the NN, as it comes with local asynchronous changes. That is, the NN can only be trained after each of the clients' models, when attack models are trained locally and individually by each client. Moreover, the NN is dependent on the network and if a client leaves or joins, this would lead to the re training of the aggregator. MI based methods in opposition are agnostic to their neighbours and any client can join or leave the network seamlessly.

3.3 Ensemble learning methods

3.3.1 Setup and hyperparameters

Here again, LeNets are used as the backbone base learners with the same fixed batch size of 16. Additionally, we will proceed again to the comparison of EL methods to the best performing schemes in FL and CI.

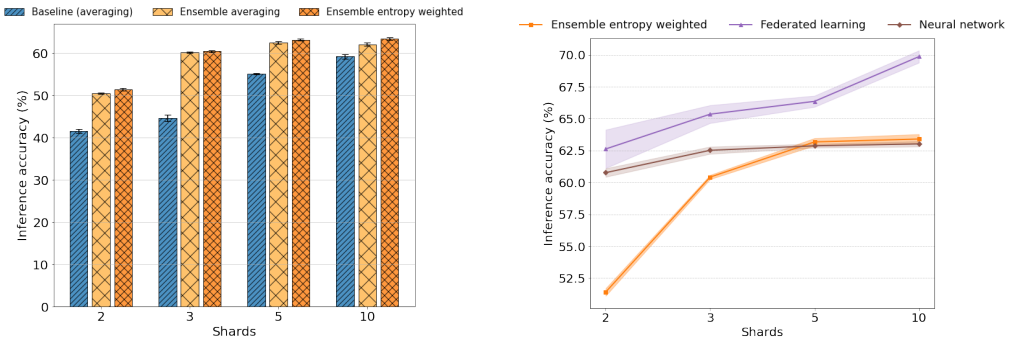
Baseline CI and FL. The setup here for both baselines, collaborative inference and federated learning, is exactly the same as in Section 3.2.1.

Ensemble learning. In this scenario, we use the Ensemble-PyTorch library [1] and instantiate a *Voting Classifier* per client. Each *Voting Classifier* is an ensemble of 10 LeNets, where the predictions of each base classifier are aggregated using a simple voting procedure. This is the setup that was advocated by the work of Lakshminarayanan *et al.* [2]. Ensembles are trained for 50 rounds, as we setup an SGD optimizer with a learning rate of 0.0025.

3.3.2 Results

All experiments in this section were run 3 times to provide error bars and enable a fair comparison.

EL vs baseline. It appears clear in figure 4a that ensemble based methods are able to achieve better performances than our baseline. The gap is at its largest for lower values of shards (i.e. shards ≤ 5), suggesting an ability for ensemble to generalise when presented with a subset of all the classification labels. This effect mitigates in high data diversity settings, indicating a improvement in performance of the base learners. The overall increasing trend evinces the ability of the base learners to generalise better thanks to being exposed to more labels during training. In such scenarios, EL based methods are still performing better than the baseline, but the gap tends to grow smaller as the effect of the ensemble mitigates. We observe that the entropy weighted averaging is able to consistently produce better performances than regular averaging. For this reason, we choose this approach as our best performing ensemble method, and we compare it against the other best performing methods.



(a) (b)

Figure 4: **(a) EL Methods:** EL methods significantly outperform the baseline, especially for low shards. **(b) EL vs FL:** The NN aggregator only outperforms EL for shards ≤ 3. In higher data diversity regimes, EL seems to be a more performing alternative.

EL vs NN aggregator and FL. Figure 4b highlights ensemble based methods as a very attractive alternative to NN aggregator. In fact, it is only outperformed by the NN aggregator for shards ≤ 3, by a margin of $\sim 2\%$. In other data diversity settings, ensemble methods can produce accuracies that are better than the NN aggregator by up to 0.5%. While FL is still unmatched, the ensemble method coupled with entropy weighted average is able to mitigate the difference to only 3%, when shards = 5.

Discussion. Finally, figure 4 showcases ensemble learning techniques as robust methods that are able to effectively increase the performance of the collaborative inference scheme. Entropy weighted averaging ensemble learning in particular is outperforming the best CI inference technique, while offering a competitive alternative to traditional Federated Learning techniques. The conclusions of the experiment are below :

(i) *Training cost:* EL based methods bear a significant training load compared to both FL and the NN aggregator as it requires the training of 10 base learners per client.

(ii) *Performance:* In mid and high data diversity settings (i.e. shards ≥ 5), EL provides good accuracies as it is able to beat the NN aggregator but only by a small margin. FL remains unmatched.

(iii) *Practicality:* While the NN aggregator requires additional data for its training, EL based methods are self sufficient and the clients' data is enough in those settings. Moreover, EL methods also allow for clients to join and leave the network seamlessly, in opposition to both CI with NN aggregator and FL.

3.4 Cross comparison

Figure 5 highlights ensemble learning as a better method than MI based techniques in all data regimes. While this difference hits its maximum for mid diversity data schemes reaching 3%, it is reduced to a minimum for shards = 2, where it reaches 0.8%. In such scenarios, it is highly recommended to opt for MI based techniques, where the drop in accuracy is compensated by a lighter training cost. In fact, while EL requires the training of 10 LeNets per client, MI necessitates only 1 LeNet and 1 attack model per client. In setups where clients are presented with a medium level of data diversity, it may be preferable to rely on EL techniques as the cost of a heavier training can be justified by a significant increase in accuracy. Moreover, the training of MI attack models assumes that the other clients are willing to share public or non-sensitive data, whereas ensemble learning techniques require no more data than what is available to the client. This contributes to compensate further the heavier training load.

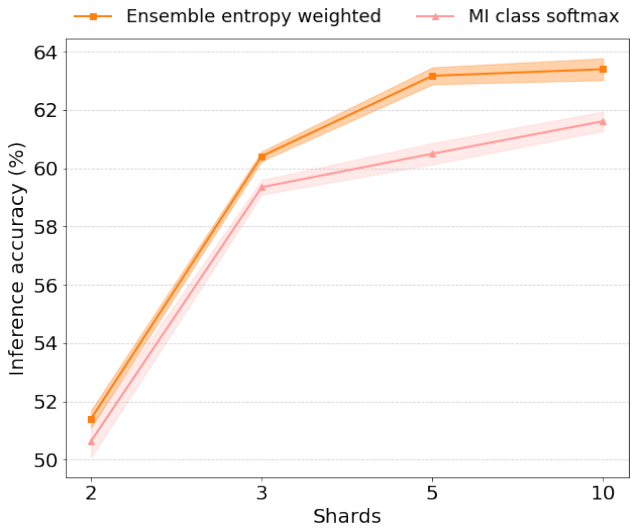


Figure 5: **MI vs EL:** Entropy weighted averaging for EL consistently outperform MI.

4 Discussion

Throughout this work, we have conceptualised, analysed and tested two alternatives in the collaborative inference pipeline that aim at producing more robust and confident predictions to improve the final inference accuracy. Both MI and EL based methods come with benefits and limitations that will be further analysed in this section.

Benefits. The two methods propose great benefits as they have been able to match or improve on the best aggregation schemes in collaborative inference. Additionally, both methods suggest improvements at the client level, which allows for local asynchronous training. Clients can freely join and leave the network, without having the need to retrain. That is not the case neither in FL, where the final model depends on the individual updates each client provides, nor in CI with NN aggregator, where the aggregator would have to be trained again from scratch. Moreover, both methods also come with a significant decrease in communication costs as every suggested improvement is done locally. FL incurs a great communication cost due to model updates exchanges between clients and the server and so does the NN aggregator whose training requires constant communication with every client for their predictions.

Limitations. While offering substantial benefits, our methods suffer from certain limitations. Mainly, they incur a heavier local training load compared to both FL and NN aggregator. As for MI, it involves training an additional binary classifier per client on top of the already existing LeNet. Ensemble learning approaches on the other hand requires even more extensive training as it suggests to train 10 LeNets per client, making it the most expensive method to train. Finally, another notable limitation introduced by MI based methods, is the need to obtain additional training data from the other clients. While it is also the case that this additional data is needed for the NN aggregator, EL approaches are self sufficient in terms of data as they only require the client's training set to be trained.

5 Conclusion

To conclude, this work presents and highlights two methods that aim at improving the inference accuracy of the collaborative inference scheme. Both membership inference and ensemble learning offer attractive alternatives to federated learning and to the best performing collaborative inference schemes. The performed experiments showcased their effectiveness especially in settings where the data diversity is medium to high. Moreover, they offer additional benefits such as the possibility to perform local asynchronous training and the ability to join and leave the network seamlessly.

Future work. Future research could delve deeper into improving collaborative inference by developing more advanced membership inference techniques and exploring various ensemble learning methods. More precisely, the idea of adversarial ensemble learning [2] seems as a natural development over the presented methods. Additionally, meta-learning techniques, such as few-shot learning [8], also appear to be suitable techniques, given their inherent ability to recognise unseen data. These directions could help enhance the effectiveness and practicality of collaborative inference in distributed machine learning settings.

References

[1] Ensemble pytorch, 2021.

[2] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[3] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015.

[4] Zhizhong Li and Derek Hoiem. Reducing overconfident errors outside the known distribution. 2018.

[5] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[6] Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, 9(10), 2020.

[7] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[8] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[9] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J Andrew Zhang, and Ren Ping Liu. Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181, 2020.

[10] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.