SCALING UNSUPERVISED DOMAIN ADAPTATION THROUGH OPTIMAL COLLABORATOR SELECTION AND LAZY DISCRIMINATOR SYNCHRONIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Breakthroughs in unsupervised domain adaptation (uDA) have opened up the possibility of adapting models from a label-rich source domain to unlabeled target domains. Prior uDA works have primarily focused on improving adaptation accuracy between the given source and target domains, and considerably less attention has been paid to the challenges that arise when uDA is deployed in practical settings. This paper puts forth a novel and complementary perspective, and investigates the algorithmic challenges that arise when uDA is deployed in a distributed ML system with multiple target domains. We propose two algorithms: i) a Collaborator Selection algorithm which selects an optimal collaborator for each target domain, and makes uDA systems more accurate and flexible; ii) a distributed training strategy that allows adversarial uDA algorithms to train in a privacy-preserving manner. We provide theoretical justifications and empirical results to show that our solution significantly boosts the performance of uDA in practical settings.

1 INTRODUCTION

Unsupervised Domain Adaptation **(uDA)** is a sub-field of machine learning aimed at adapting a model trained on a labeled source domain to a different, but related, *unlabeled* target domain. Over the last few years, many exciting uDA algorithms have been proposed to enhance the accuracy of deep learning models in a target domain using unlabeled data (Long et al. (2015; 2017; 2018); Ganin et al. (2016); Tzeng et al. (2017); Hoffman et al. (2018); Shen et al. (2018); Zou et al. (2019)).

In this paper, we study an interesting and under-explored aspect of uDA algorithms, pertaining to their scalability in practical ML systems. To motivate our problem setting, let us consider the recent interest in predicting COVID-19 in patients by analyzing the computerized tomography (CT) scans of their chest (e.g. Zheng et al. (2020)). Assume that scientists in China (*source domain*) have collected a labeled dataset of CT scans (e.g., Zhao et al. (2020)) and trained a COVID-19 prediction model on it. This model now needs to be deployed in five countries (*target domains*) from where *labeled data is unavailable*. Due to virus mutations across the world, the CT scan samples from different countries may follow different data distributions, and the source model may not work accurately for all target countries. Hence, uDA could become a promising approach to learn a model tailored for each *target* country's distribution.

In this setting, we have a single labeled source domain and multiple unlabeled target domains (appearing sequentially) for which we want to learn a model using uDA. Below are two challenges in scaling existing uDA approaches in this setting:

(i) Finding the optimal adaptation collaborator. We define a *collaborator* as the domain with which a target domain undergoes adaptation, e.g., in A→B adaptation, A is the collaborator for the target domain B. Existing uDA methods are static by design, in that they assume that target domains would *always* adapt from the labeled source domain. However, this static approach of *always* choosing the labeled source as the collaborator may not be the most optimal. For instance, some target countries may have a virus strain very different from China (the labeled source), and hence adapting from the Chinese COVID-19 model may not be optimal for them.

To better explain this problem, we show an experiment on Rotated MNIST, a variant of MNIST in which digits are rotated clockwise by different degrees. Assume that 0°, i.e., no rotation, is the labeled source domain while 30°, 60°, 45°, 90° and 15° are five unlabeled target domains appearing sequentially, for which we would like to learn a model using uDA. Figure 1a shows the accuracy obtained in each target domain if we always choose the labeled source 0° as their collaborator. While

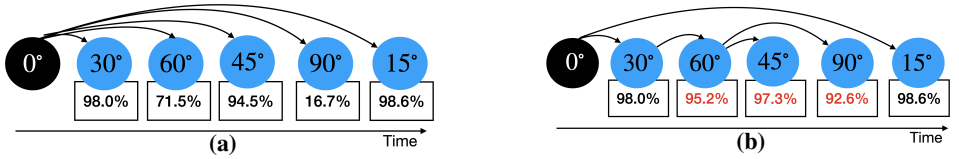


Figure 1: 0° is the labeled source domain while the domains in blue are unlabeled target domains. The numbers in rectangle denote the post-adaptation accuracy for a domain. (a) Static Design: Labeled Source acts the collaborator for each target domain. (b) Flexible Design: Each target domain chooses its collaborator dynamically. Previously adapted target domains can also act as collaborators. Note that choosing the right collaborator leads to major accuracy gains over the Static Design for many domains (shown in red).

this approach results in high accuracies for 15° and 30°, it performs poorly for other domains. Can we do better? *What if a target domain can adapt not just from the labeled source, but also from other target domains which themselves have undergone adaptation in the past*. Figure 1b shows that if target domains could flexibly choose their collaborators, they can achieve significantly higher accuracies. E.g., if 90° adapts from 60° (which itself underwent adaptation in the past), it could be achieve an accuracy of 92.6%, almost 75% higher than what could be achieved by adapting from 0°.

In summary, when uDA algorithms scale to settings with multiple target domains such as the COVID-19 example, selecting an optimal collaborator for each target domain becomes critical. How do we select this optimal collaborator is a key research question that this paper aims to answer.

(ii) Distributed and Private Datasets. Adversarial uDA methods generally assume that datasets from the source and target domains are available on the same machine before the adaptation process begins. While this assumption makes it easy to research uDA algorithms, it does not hold true in practice. In the example above, the CT scans of patients are sensitive health records, and source and target domains may not be allowed to share them with each other due to privacy reasons. Clearly, this bottleneck can severely limit the adoption of uDA in realistic settings. How can we extend uDA techniques to work in distributed settings while preserving the privacy of each domain?

Contributions. Our contributions are as follows: (1) We propose Optimal Collaborator Selection (OCS), an algorithm which boosts the accuracy of uDA in multi-target settings by as much as 50%. OCS is built on a novel theoretical formulation which estimates the cross-entropy error of an unlabeled target domain, based on the collaborator error and Wasserstein distance between the collaborator and target domain. (2) We propose DILS, an algorithm to extend adversarial uDA to distributed settings while preserving the privacy of source and target domains. (3) We show that OCS and DILS can be used together in an end-to-end framework to address the key challenges in scaling uDA algorithms. (4) We conduct extensive experiments on four image and speech datasets to demonstrate the superior performance of OCS and DILS over various baselines. Finally, we demonstrate that our solution is not limited to any specific uDA algorithm, and it can generalize to various uDA optimization objectives proposed in the literature.

2 Preliminaries and Problem Formulation

Primer. We first provide a brief primer on uDA when there is one source and one target domain. Let D_S be a source domain with input samples X_S and labels Y_S, and D_T be a target domain with input samples X_T, *without* labeled observations. In a domain adaptation problem, it is assumed that X_S and X_T are drawn from different distributions. We can train a feature extractor, E_S, and a classifier, C_S for the source domain using supervised learning as follows:

$$\min_{E_S, C_S} \mathcal{L}_{cls} = -\mathbb{E}_{(x_s, y_s)\sim(X_S, Y_S)} \sum_{k=1}^{K} \mathbb{1}_{[k=y_s]} [\log(C_S(E_S(x_s))]$$

The goal of uDA is to learn a feature extractor E_T for the unlabeled target domain, which minimizes the divergence between the empirical source and target feature distributions. If the divergence in feature representations between domains is minimized, we can apply the pre-trained source classifier C_S on the target features and obtain inferences, without requiring to learn a separate C_T.

To learn E_T, two losses are optimized using adversarial training: a discriminator loss $\mathcal{L}_{adv_{DI}}$ and a mapping loss $\mathcal{L}_{adv_M}$. The computation of these losses differ across uDA algorithms – e.g., ADDA (Tzeng et al. (2017)) uses label inversion to minimize the Jensen-Shannon divergence between source and target feature distributions as follows:

$$\min_{DI} \mathcal{L}_{adv_{DI}} = -\mathbb{E}_{x_s\sim\mathcal{X}_S}[\log(DI(E_S(x_s))] - \mathbb{E}_{x_t\sim\mathcal{X}_T}[\log(1 - DI(E_T(x_t))] \tag{1}$$

$$\min_{E_T} \mathcal{L}_{adv_M} = -\mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(DI(E_T(x_t))] \tag{2}$$

where DI represents a domain discriminator that aims to distinguish source and target domains.

Problem Formulation. We now formalize our problem setting of scaling uDA to settings with a single labeled source and multiple unlabeled target domains. Moreover, the domain datasets are private and located on distributed nodes. Let $\{D_T^j | j = 1, \ldots, K\}$ be the K unlabeled target domains for which we want to learn a prediction model using uDA. We assume that target domains appear sequentially, one at a time. Under this setting, we now formalize the two research problems:

(i) Optimal Collaborator Selection. We define a candidate set $\mathbb{Z}_\tau$ as the set of candidate domains that are available to collaborate with a target domain at step τ. When a uDA system initializes at step $\tau = 0$, only the labeled source domain has a learned model, hence $\mathbb{Z}_0 = \{D_S\}$. When the first target domain D_T^1 appears, it adapts from D_S and learns a model E_T^1. Having learned a model, D_T^1 is now added to the candidate set (along with its unlabeled data) and can act as a collaborator for future domains.

In general, at step $\tau = K$, $\mathbb{Z}_K = \{S\} \cup \left\{ D_T^j | j = 1, \ldots, K \right\}$. For a new target domain D_T^{K+1}, the goal of OCS is to find an optimal collaborator domain $D_{\text{opt}} \in \mathbb{Z}_K$, such that:

$$D_{\text{opt}} = \underset{i=1\ldots|\mathbb{Z}_K|}{\text{argmin}} \; \Phi(\mathbb{Z}_K^i, D_T^{K+1})$$

where $\mathbb{Z}_K^i$ is the i^{th} candidate domain in $\mathbb{Z}_K$ and Φ is a metric that quantifies the risk of collaboration between $\mathbb{Z}_K^i$ and D_T^{K+1}.

(ii) Distributed Adversarial Training. Once a collaborator D_{opt} is selected, we need to perform distributed uDA between D_{opt} and the target domain D_T^{K+1}, while preventing any data leaks? This opens up two challenges: i) how do we distribute the adversarial uDA network architecture and training process across the nodes? ii) how do we ensure that the gradients exchanged between the distributed nodes during training cannot be used to reconstruct the raw data?

3 FRUDA: FRAMEWORK FOR REALISTIC UDA

We first present our two algorithmic contributions on Optimal Collaborator Selection (named OCS) and Distributed uDA (named DILS). Later in §3.3, we explain how these two algorithms work in conjunction with each other in an end-to-end framework called FRUDA, and allow for scaling uDA algorithms in multi-domain, distributed ML systems.

3.1 OPTIMAL COLLABORATOR SELECTION (OCS)

For any given target domain, the goal of OCS is to find its optimal collaborator domain D_{opt} from a set of candidate domains. Our key idea is quite intuitive: the optimal collaborator should be a domain, such that adapting from it will lead to the highest classification accuracy (or equivalently, the lowest classification error) in the target domain. We first introduce some notations and then present the key theoretical insight that underpins OCS.

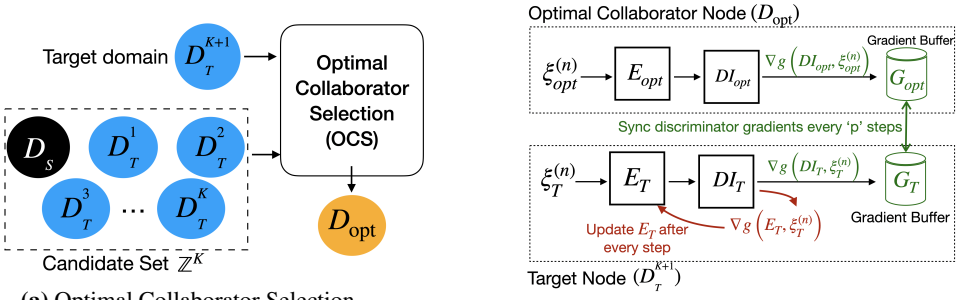
Notations. We use domain to represent a distribution D on input space $\mathcal{X}$ and a labeling function $l : \mathcal{X} \to [0, 1]$. A hypothesis is a function $h : \mathcal{X} \to [0, 1]$. Let $\varepsilon_{M,D}(h, l)$ denote the error of a hypothesis h w.r.t. l under the distribution D, where M is an error metric such as L_1 error or cross-entropy error . Further, a function f is called θ-Lipschitz if it satisfies the inequality $\|f(x) - f(y)\| \leq \theta\|x - y\|$ for some $\theta \in \mathbb{R}_+$. The smallest such θ is called the Lipschitz constant of f.

Theorem 1 *Let D_1 and D_2 be two domains sharing the same labeling function l. Let θ_{CE} denote the Lipschitz constant of the cross-entropy loss function in D_1. For any two θ-Lipschitz hypotheses h, h', we can derive the following error bound for the cross-entropy (CE) error in D_2:*

$$\varepsilon_{\text{CE},D_2}(h, h') \leq \theta_{\text{CE}} \left(\varepsilon_{\text{L}_1,D_1}(h, h') + 2\theta W_1(D_1, D_2) \right) \tag{3}$$

where $W_1(D_1, D_2)$ denote the first Wasserstein distance between the domains D_1 and D_2, and $\varepsilon_{\text{L}_1,D_1}$ denotes the L_1 error in D_1. Proof is in §E in the Appendix.

Theorem 1 has two key properties that make it apt for our problem setting. First, it can be used to directly estimate the CE error in a target domain (D_2), given a hypothesis (or a classifier) from a collaborator domain (D_1). Since target CE error is the key metric of interest in classification tasks,



(a) Optimal Collaborator Selection

(b) DILS: Distributed uDA

Figure 2: (a) A new target domain D_T^{K+1} finds its optimal adaptation collaborator D_{opt} from a set of candidate domains. (b) D_T^{K+1} performs distributed uDA with D_{opt} to learn a model for its distribution.

this bound is more useful than the one proposed by Shen et al. (2018) which estimates the L_1 error in the target domain. Secondly, the bound depends on the Wasserstein distance metric between the domains, which could be computed in a distributed way without exchanging any private data between domains. This property is very important to guarantee the domain privacy, which is one of the objectives of our work. Please refer to §D.1 for implementation details on computation of Wasserstein distance in a distributed and privacy-preserving manner.

Selecting the optimal collaborator. Motivated by Theorem 1, we now discuss how to select the optimal collaborator for a target domain. Given a collaborator domain D_c, a learned hypothesis h^c and a labeling function l, we can estimate the CE error for a target domain D_T using Theorem 1 as:

$$\varepsilon_{\text{CE,D}_\text{T}}(h^c, l) \leq \theta_{\text{CE}}(\varepsilon_{\text{L}_1,\text{D}_\text{c}}(h^c, l) + 2\theta W_1(D_c, D_T)) \tag{4}$$

We can tighten the bound in Eq. 4 to get a more reliable estimate of the target CE error. This is achieved by reducing the Lipschitz constant (θ) of the hypothesis h^c during training. In uDA, the hypothesis is parameterized by a neural network, and we can train neural networks with small Lipschitz constants by regularizing the spectral norm of each network layer (Gouk et al. (2018)).

Now that we have a way to estimate the target CE error, we can use it to select an optimal collaborator that yields the minimum target CE error. Let $\mathbb{Z} = \{D^k | k = 1, \ldots, K\}$ be a set of candidate domains each with a pre-trained model h^k with Lipschitz constants θ^k_{CE} and θ^k. Let D_T^{K+1} be a target domain for which the collaborator is to be chosen. We use Eq. 4 to select the optimal collaborator D_{opt} as:

$$D_{\text{opt}} = \underset{k=1,\ldots,K}{\text{argmin}}\ \theta^k_{\text{CE}}(\varepsilon_{\text{L}_1,\text{D}^\text{k}}(h^k, l) + 2\theta^k W_1(D^k, D_T^{\text{K}+1})) \tag{5}$$

3.2 Distributed uDA using Discriminator-based Lazy Synchronization (DILS)

Upon selecting an optimal collaborator D_{opt} for the target domain D_T^{K+1}, the next step is to learn a model for D_T^{K+1} by doing uDA with D_{opt}. In line with our problem setting, both domains are located on distributed nodes and cannot share their raw private data with each other.

FADA (Peng et al. (2020)) is a recently proposed technique for *adversarial* uDA in distributed settings. Our solution (DILS) differs from FADA in three important ways: (i) FADA was designed for a federated learning setup and assumes multiple labeled source domains, which is not the case in our setting. (ii) FADA exchanges feature representations of the data and corresponding gradients between nodes to achieve distributed training. Prior works (Vepakomma et al. (2020); Aono et al. (2017); Zhu et al. (2019)) have shown that these are prone to privacy attacks and can be exploited to reconstruct the raw data. Instead, we leverage a unique characteristic of adversarial training architectures and show that adversarial uDA can be performed between distributed nodes only by exchanging the *gradients of domain discriminators*. The biggest benefit of our approach is that it provides protection against state-of-the-art gradient leakage attacks. (iii) FADA exchanges the gradients between nodes after every batch of data, which can increase the overall training time of uDA due to the communication overhead associated with gradient exchange. DILS, instead, adopts a lazy gradient synchronization approach which significantly reduces the uDA training time.

Method. As shown in Figure 2b, we split the adversarial architecture across the distributed nodes. The feature encoders of the collaborator (E_{opt}) and target (E_T) reside on their respective nodes, while the discriminator DI is split into two components DI_{opt} and DI_T. At every training step n, both nodes feed their private training data ξ^n_{opt} and ξ^n_T into their encoders and discriminators, and compute the gradients of the discriminators, i.e., $\nabla g(DI_{\text{opt}}, \xi^n_{\text{opt}})$ and $\nabla g(DI_T, \xi^n_T)$ respectively.

How often should we exchange the discriminator gradients between nodes? By exchanging discriminator gradients after every training step, we can keep the discriminators synchronized and ensure that distributed training converges to the non-distributed solution. However, this approach has a major downside, as gradient exchange every step incurs significant communication costs and increases the overall uDA training time. To increase training efficiency, we propose a *Lazy Synchronization* approach, wherein instead of every step, the discriminators are synchronized after every p training steps, thereby reducing the total gradient exchange by a factor of p. We denote the training steps at which the synchronization takes place as the *sync-up steps* while other steps are called *local steps*. Following the notations in §2, we present our DILS strategy in Algorithm 1. As evident, DILS works on the principle of using synced stale gradients (∇g_{sync}) from the last sync-up step to update the discriminators, instead of local gradients. This is very important as it prevents the distributed discriminators from diverging in the local steps. The stale gradients are refreshed at every *sync-up* step. Our results show that this approach reduces the uDA training time with minimal effect on target accuracy.

Algorithm 1: DILS

Result: E_T

Input: Pre-trained E_{opt}; Randomly Initialize DI_{opt}; Initialize $E_T = E_{\text{opt}}$; DI_T=DI_{opt}; Sync up step p; total steps N ;

for $n = 1, 2, ..., N$ **do**

- Sample a batch of data on both nodes, $\xi_{\text{opt}}^{(n)}$ and $\xi_T^{(n)}$. Then feed $\xi_{\text{opt}}^{(n)}$ and $\xi_T^{(n)}$ into the respective Encoder-Discriminator model separately on both nodes;
- Based on different loss functions, calculate the gradients locally. On collaborator node, calculate $\nabla g(DI_{\text{opt}}, \xi_{\text{opt}}^{(n)})$; On target node, calculate $\nabla g(E_T, \xi_T^{(n)})$ and $\nabla g(DI_T, \xi_T^{(n)})$;
- Add $\nabla g(DI_{\text{opt}}, \xi_{\text{opt}}^{(n)})$ to gradient buffer G_{opt}, add $\nabla g(DI_T, \xi_T^{(n)})$ to target gradients buffer G_T ;
- **if** *isTargetNode* **then**
  - Apply $\nabla g(E_T, \xi_T^{(n)})$ to E_T;
- **if** $n\%p == 0$ **then**
  - Exchange gradients buffer and update the latest synced gradients $g_{\text{sync}} = \frac{G_{\text{opt}}+G_T}{2p}$;
  - Clear G_{opt} and G_T ;
- Apply g_{sync} to DI_{opt} and DI_T separately;

Privacy Analysis. DILS does not exchange raw data, extracted features or the gradients of the feature extractor during training, all of which are prone to reconstruction attacks (Vepakomma et al. (2020); Zhu et al. (2019)). Instead, DILS exploits a unique feature of adversarial architecture and performs distributed uDA by only exchanging gradients of the discriminators. Through a theoretical analysis presented in §E.2, we show that discriminator gradients cannot be used to reconstruct the raw domain data, even with state-of-the-art privacy attacks (Zhu et al. (2019)).

Convergence Analysis. While DILS accelerates training through lazy gradient sync, it is important to study if it can maintain the same convergence properties as non-distributed uDA. In §E.1, we build upon prior GAN convergence and distributed training theory to prove that when the sync-up step p is sufficiently small, DILS has a similar convergence rate as the non-distributed setting.

Theorem 2. In *Lazy Synchronization*, given a fixed target encoder, we have the following convergence rate for the discriminators DI_{opt} and DI_T after T iterations:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}\left[\frac{f(x_0)-f(x_{t+1})}{\mu T} + (2p-1)L\mu\sigma^2 + \frac{\mu L\sigma^2}{2} + \frac{\mu^3 L^3\sigma^2(2p-1)}{2}\right]$$

where p is the sync-up step, μ is the learning rate. Set $\mu = O(1/\sqrt{T})$. When $p << L\sigma^2$, the impact of stale update will be very small, and thus it can converge with rate $O(1/\sqrt{T})$, same as the non-distributed setting.

3.3 Combining OCS with DILS

We now discuss how OCS and DILS work together to address the challenges introduced in §1. As shown in Figure 2a, a new target domain D_T^{K+1} first performs OCS with all candidate domains in $\mathbb{Z}^K$ to find its optimal collaborator D_{opt}. This step makes uDA systems more flexible and ensures that each target domain is able to achieve the best possible adaptation accuracy in the given setting. Next, as shown in Figure 2b, D_T^{K+1} and D_{opt} use DILS to engage in distributed uDA. This step ensures that no private data is exposed during adaptation and yet the target domain is able to learn a model for its distribution. Finally, the newly adapted target domain D_T^{K+1} (with its model and unlabeled data) is added to the candidate set $\mathbb{Z}$ to serve as a potential collaborator for future domains.

4 EVALUATION

Datasets. We evaluate FRUDA on four image and speech datasets: Rotated MNIST, Digits, Office-Caltech, and Mic2Mic. *Rotated MNIST* is a variant of MNIST with digits rotated clockwise by different degrees. Each rotation is considered a separate domain. *Digits* has five domains: MNIST (M), USPS (U), SVHN (S), MNIST-M (MM) and SynNumbers (SYN), each consisting of digit classes ranging from 0-9. The *Office-Caltech* dataset contains images of 10 classes from Amazon (A), DSLR (D), Webcam (W), and Caltech-256 (C). Finally, Mic2Mic (Mathur et al. (2019)) is a speech keyword detection dataset recorded with four microphones: Matrix Creator, Matrix Voice, ReSpeaker and USB. Each microphone represents a domain. More details about the datasets are provided in §F.0.2 in the Appendix.

4.1 PERFORMANCE OF DISCRIMINATOR-BASED LAZY SYNCHRONIZATION (DILS)

We first evaluate the convergence properties of DILS against two baselines: Non-Distributed uDA and FADA. As discussed in §3.2, FADA was originally designed for multiple sources in a federated learning setup. For a fair comparison with our single-source setting, we modify FADA by only implementing its Federated Adversarial Alignment component and setting the number of source domains to one. The modified FADA has the same optimization objectives as single-source DA, but it operates in a distributed setting. Hence, it is fair to compare it with DILS. As we discussed earlier, DILS provides *privacy benefits* over the baselines and is robust against gradient leakage attacks. However, two key questions still remain: (i) *Training Time*: to what extent can DILS reduce the training time for distributed uDA? (ii) *Target Accuracy*: can the use of stale gradients and lazy synchronization in DILS degrade the classification accuracy in the target domain?

Results. Table 1 shows the performance of DILS against the baselines on 4 datasets and 8 adaptation tasks. First, we look at the mean training times for uDA (denoted as 't' in Table 1). As expected, non-distributed uDA, wherein source and target datasets are required to be on the same machine, has the fastest convergence because there is no gradient communication time involved; however this comes at the expense of domain privacy. Importantly, the end-to-end convergence of DILS is 37% faster than FADA in a distributed setting, primarily due to the reduced overhead of gradient communication between nodes. Further, Table 1 shows a very promising result that DILS can achieve similar accuracy as the baselines, which confirms our theoretical analysis that lazy synchronization of discriminator gradients does not degrade the target accuracy. Averaged over all adaptation tasks, the accuracy difference between DILS and the baselines is less than 0.5%, and it could be further reduced by choosing a smaller p. We provide more discussion on the choice of p in §4.3.

	RMNIST			Office-Caltech			Digits			Mic2Mic		
Training	**30 → 60**	**150 → 180**	**t** (mins)	**W → C**	**D → A**	**t** (mins)	**M-M → U**	**Syn → U**	**t** (mins)	**C → U**	**R → C**	**t** (mins)
No Adaptation	32.68	61.51	-	87.81	85.28	-	57.54	79.32	-	73.98	67.47	-
Non-Distributed	69.61	91.86	2.4	91.74	92.11	21	82.14	90.1	5.69	79.85	77.52	5.8
FADA	69.30	91.21	65.28	91.02	92.03	73.8	82.13	89.9	86.94	79.81	77.39	37.8
DILS	68.34	90.16	35.2	90.56	91.77	55.05	81.66	89.78	54.6	79.33	77.38	22.5

Table 1: Target Domain Accuracy and mean uDA Training Time (t). DILS has a 37% faster convergence time than FADA on average, without a major drop in adaptation accuracy. The sync-up step p for DILS is set to 4.

4.2 PERFORMANCE OF THE PROPOSED FRAMEWORK, FRUDA

Recall that FRUDA comprises of two algorithms: Optimal Collaborator Selection (OCS) and distributed uDA (DILS) algorithms. In §4.1, we established that DILS is an effective algorithm for distributed uDA, in terms of training time, data privacy and adaptation accuracy. Now we evaluate how DILS can work in conjunction with OCS to scale uDA in practical settings.

Experiment Setup. In our problem setting, domains appear sequentially in an order. Let $\{\mathbf{D_S}, D_T^1, D_T^2 \cdots D_T^K\}$ denote an ordering of one labeled source domain $\mathbf{D_S}$ and K unlabeled target domains. For each target domain $D_T^i|_{i=1}^K$, we first choose a collaborator domain, which could be either the labeled source domain $\mathbf{D_S}$ or any of the previous target domains $D_T^j|_{j=1}^{i-1}$ that have already learned a model using uDA. Upon choosing a collaborator (using OCS or any of the baseline techniques), we use DILS to perform distributed uDA between the target domain and the collaborator, and compute the post-adaptation test accuracy Acc_T^i in the target domain. We report the mean adaptation accuracy obtained over all target domains, i.e., $\frac{1}{K}\sum_{i=1}^K \text{Acc}_T^i$.

	RMNIST		Digits		Office-Caltech		Mic2Mic	
	Order_1	Order_2	Order_1	Order_2	**D**,W,C,A	**W**,C,D,A	Order_1	Order_2
No Adaptation	34.65	35.54	59.59	72.09	66.40	85.25	76.45	75.83
Random	28.66±6.50	37.11±4.32	62.77±2.19	69.13±4.0	69.18±1.51	80.1±2.44	80.17±1.60	77.34±1.09
Labeled Source (LS)	47.14 ± 0.85	49.08± 0.75	64.89±0.23	79.87±0.31	67.77±0.15	**90.62±0.13**	80.86 ± 0.09	79.91±0.05
Multi-Collaborator	40.51±0.30	42.73±0.39	60.94±0.13	75.91±0.30	68.90±0.24	82.17±0.87	76.90 ± 0.13	79.0±0.24
Proxy A-Distance	93.51 ± 0.22	74.14±0.05	70.09±0.45	83.07±0.15	69.37±0.2	**90.62±0.13**	80.34 ± 0.19	80.02±0.31
FRUDA (Ours)	**97.08 ± 0.14**	**81.72±0.3**	**73.01±0.87**	**85.31±0.26**	**74.56±0.52**	90.62±0.13	**81.43 ± 0.06**	**81.81±0.10**

Table 2: Mean accuracy over all target domains in a given order, e.g., Order_1=**D**,W,C,A for Office-Caltech.

For each dataset, we use two random orderings of source and target domains, e.g., for *Office-Caltech*, we choose Order_1 = **D**,W,C,A and Order_2 = **W**,C,D,A. Please refer to §F for details on orderings used for other datasets, and our computing infrastructure. The optimization objectives of ADDA presented in Eq. 1 and 2 are used for adaptation in this experiment.

Collaborator Selection Baselines. We compare OCS against four baselines: (i) *Labeled Source* wherein each target domain only adapts from the labeled source domain; (ii) *Random Collaborator*: each target domain chooses a random collaborator from the available candidates; (iii) *Proxy A-distance (PAD)* where we choose the domain which has the least PAD (Ben-David et al. (2007)) from the target; (iv) *Multi-Collaborator* is based on MDAN (Zhao et al. (2018)), where all available candidate domains contribute to the adaptation in a weighted-average way. However, MDAN was developed assuming that all candidate domains are labeled, which is not the case in our setting. Hence, we modify MDAN and only optimize its adversarial loss during adaptation (details in §D).

Results. Table 2 reports the mean accuracy obtained over the target domains for two domain orderings in each dataset. We observe that FRUDA outperforms the baseline collaborator selection techniques in most cases. Importantly, it can provide significant gains over the *Labeled Source* (LS) baseline, which verifies our hypothesis that the labeled source domain is not *always* optimal for uDA. We highlight three key results from Table 2: (a) in RMNIST, FRUDA provides 41% accuracy gains over LS on average — this could be partly attributed to the large number of target domains ($K = 11$) in this dataset. As the number of target domains increase, there are more opportunities for benefiting from collaboration selection, which led to higher accuracy gains over LS in this dataset. (b) For Office-Caltech (order = **W**,C,D,A), the labeled source domain **W** turns out to be the optimal collaborator for all target domains, and FRUDA managed to converge to the Labeled Source baseline. (c) Multi-collaborator baseline often performed worse than even LS. We surmise that this is due to negative transfer caused by some collaborator domains that are too different from the target domain. OCS, on the other hand, is able to filter out these bad collaborators, which leads to higher adaptation accuracy. In general, our results demonstrate that as uDA systems scale to multiple target domains, the need for choosing the right adaptation collaborator becomes important, hence warranting the need for accurate collaboration selection algorithms.

4.3 Discussion and Analysis

Generalization to other uDA optimization objectives. The results presented in Table 2 used the optimization objectives of ADDA from Eq. 1 and 2. However, FRUDA is intended to be a general framework not limited to one specific uDA algorithm. We now evaluate FRUDA with three other uDA loss formulations (i) *DANN*, which uses a Gradient Reversal Layer (GRL) to compute the mapping loss (Ganin et al. (2016)), (ii) *WassDA*, which uses Wasserstein Distance as a loss metric for the domain discriminator (Shen et al. (2018)) and (iii) CADA by Zou et al. (2019) which operates by enforcing consensus between source and target features. In Table 3, we observe that while different uDA techniques yield different target accuracies, FRUDA can work in conjunction with all of them to improve the overall accuracy over the *Labeled Source* baseline.

	RMNIST (Order_1)				Digits (Order_1)			
	ADDA	GRL	WassDA	CADA	ADDA	GRL	WassDA	CADA
No Adaptation	34.65	34.65	34.65	34.65	59.59	59.59	59.59	59.59
Labeled Source	47.14	47.26	44.39	41.30	64.89	65.51	70.34	65.22
FRUDA(Ours)	**97.08**	**97.35**	**91.15**	**83.37**	**73.01**	**69.80**	**75.36**	**70.19**

Table 3: Mean target accuracy for four uDA methods. Our framework can be used in conjunction with various uDA methods, and improves mean accuracy over the Labeled Source baseline.

Effect of sync-up step p. In DILS, p controls the frequency of gradient exchange between the nodes. In §C.1 and Figure 3, we vary p from 1 and 10 and calculate the adaptation accuracy in the target domain. Results show that even at high values of p, DILS provides comparable accuracy to non-distributed training, while reducing gradient exchange by a factor of p.

Properties of OCS. We now discuss two interesting properties of OCS. In sequential adaptation where a target domain adapts from previous domain(s), negative transfer and error propagation is a possibility, especially if an unrelated domain appears in the sequence. In §C.2, we show that OCS can overcome this important issue in sequential adaptation.

Although this paper's scope was limited to studying the setting with a single labeled source, OCS can also boost adaptation accuracy in multi-source settings. §C.3 shows that OCS can work with MDAN, a multi-source uDA algorithm, and improve its performance by up to 13%.

5 Related Work

Related work for OCS. There are prior works on computing similarity between domains, e.g., using distance measures such as Maximum Mean Discrepancy (Wang et al. (2020)), Gromov-Wasserstein Discrepancy (Yan et al.), A-distance (Wang et al. (2018)), and subspace mapping (Gong et al. (2012)). However, our results in §4.2 show that merely choosing the most similar domain as the collaborator is not optimal. Instead, OCS directly estimates the target cross-entropy error for collaborator selection. Another advantage of OCS over prior methods is that it can work in a fully distributed manner without compromising domain privacy.

There are also works on selecting or generating intermediate domains for uDA. Tan et al. (2015) studied a setting when source and target domains are too distant (e.g., image and text) which makes direct knowledge transfer infeasible. As a solution, they propose selecting intermediate domains using A-distance and domain complexity. However, as we discussed, merely using distance metrics does not guarantee the most optimal collaborator. Moreover, this work was done on a KNN classifier and did not involve adversarial uDA algorithms. Gong et al. (2019) and Choi et al. (2020) use style transfer to generate images in intermediate domains between the source and target. Although interesting, these works are orthogonal to OCS in which the goal is to select the best domain from a given set of candidates. Moreover, these works are primarily focused on visual adaptation, while OCS is a general method that can work for any modality. Finally, Wulfmeier et al. (2018); Bobu et al. (2018) are techniques for incremental uDA in continuously shifting domains. However, in our problem, different target domains may not have any inherent continuity in them, and hence it becomes important to perform OCS.

Related work for DILS. There is prior work on *distributed model training*, wherein training data is partitioned across multiple nodes to accelerate training. These methods include centralized aggregation (Li et al. (2014); Sergeev & Del Balso (2018)), decentralized training (Lian et al. (2017a); Tang et al. (2018)), and asynchronous training (Lian et al. (2017b)). Similarly, with the goal of preserving data privacy, *Federated Learning* proposes sharing model parameters between distributed nodes instead of the raw data Konečnỳ et al. (2016); Yang et al. (2019). However, these distributed and federated training techniques are primarily designed for supervised learning and do not *extend directly to uDA architectures*. A notable exception is FADA by Peng et al. (2020) which extends uDA to federated learning. As we extensively discussed in § 3.2, FADA exchanges the features and gradients of the *feature extractor* between nodes to achieve domain privacy, which are prone to privacy attacks. Instead, DILS operates by exchanging *discriminator gradients* between nodes, which bring both privacy and training-time benefits over FADA.

Related work on practical uDA. Kundu et al. (2020) and Liang et al. (2020) are two very promising recent works on source-dataset-free uDA. Although the scope of these works are different from us, we share the same goal of making uDA techniques more practical. Specifically, we focus on developing general algorithms and frameworks to scale existing uDA approaches in realistic settings.

For a tabular summary of the related work, please refer to Table 4.

6 Conclusion

This paper identified two critical bottlenecks in scaling uDA algorithms in real-world ML systems, namely the lack of flexibility in choosing adaptation collaborators, and the need to exchange private data during adaptation. Our proposed solutions — (i) an optimal collaborator selection algorithm and (ii) a distributed uDA algorithm — address these bottlenecks and bring a novel and cross-disciplinary perspective to uDA literature. We provided theoretical justifications and proofs on the privacy and convergence of our algorithms and extensively evaluated them on four datasets.

REFERENCES

Transfer Learning Repository. https://github.com/jindongwang/transferlearning/blob/master/data/dataset.md, 2019.

Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2017.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in neural information processing systems*, 2007.

Andreea Bobu, Eric Tzeng, Judy Hoffman, and Trevor Darrell. Adapting to continuously shifting domains. 2018.

Jongwon Choi, Youngjoon Choi, Jihoon Kim, Jin-Yeop Chang, Ilhwan Kwon, Youngjune Gwon, and Seungjai Min. Visual domain adaptation by consensus-based transfer to intermediate domain. In *AAAI*, pp. 10655–10662, 2020.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2066–2073. IEEE, 2012.

Rui Gong, Wen Li, Yuhua Chen, and Luc Van Gool. Dlow: Domain flow for adaptation and generalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2477–2486, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *The Journal of Machine Learning Research*, 15(1):2489–2512, 2014.

Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International Conference on Machine Learning*, pp. 1994–2003, 2018.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

Jogendra Nath Kundu, Naveen Venkat, Ambareesh Revanur, R Venkatesh Babu, et al. Towards inheritable models for open-set domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12376–12385, 2020.

Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pp. 583–598, 2014.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 5330–5340, 2017a.

Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1710.06952*, 2017b.

Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. *arXiv preprint arXiv:2002.08546*, 2020.

Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2208–2217. JMLR. org, 2017.

Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, pp. 1640–1650, 2018.

Akhil Mathur, Anton Isopoussu, Fahim Kawsar, Nadia Berthouze, and Nicholas D Lane. Mic2mic: Using cycle-consistent generative adversarial networks to overcome microphone variability in speech systems. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pp. 169–180, 2019.

Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 691–706. IEEE, 2019.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Xingchao Peng, Zijun Huang, Yizhe Zhu, and Kate Saenko. Federated adversarial domain adaptation. *ICLR*, 2020.

Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ben Tan, Yangqiu Song, Erheng Zhong, and Qiang Yang. Transitive transfer learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1155–1164, 2015.

Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D˜2: Decentralized training over decentralized data. *arXiv preprint arXiv:1803.07068*, 2018.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7167–7176, 2017.

Praneeth Vepakomma, Abhishek Singh, Otkrist Gupta, and Ramesh Raskar. Nopeek: Information leakage reduction to share activations in distributed deep learning. *arXiv preprint arXiv:2008.09161*, 2020.

Jindong Wang, Vincent W Zheng, Yiqiang Chen, and Meiyu Huang. Deep transfer learning for cross-domain activity recognition. In *proceedings of the 3rd International Conference on Crowd Science and Engineering*, pp. 1–8, 2018.

Jindong Wang, Yiqiang Chen, Wenjie Feng, Han Yu, Meiyu Huang, and Qiang Yang. Transfer learning with dynamic distribution adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(1): 1–25, 2020.

Markus Wulfmeier, Alex Bewley, and Ingmar Posner. Incremental adversarial domain adaptation for continually changing environments. In *2018 IEEE International conference on robotics and automation (ICRA)*, pp. 1–9. IEEE, 2018.

Yuguang Yan, Wen Li, Hanrui Wu, Huaqing Min, Mingkui Tan, and Qingyao Wu. Semi-supervised optimal transport for heterogeneous domain adaptation.

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):12, 2019.

Han Zhao, Shanghang Zhang, Guanhang Wu, José MF Moura, Joao P Costeira, and Geoffrey J Gordon. Adversarial multiple source domain adaptation. In *Advances in Neural Information Processing Systems*, pp. 8559–8570, 2018.

Jinyu Zhao, Yichen Zhang, Xuehai He, and Pengtao Xie. Covid-ct-dataset: a ct scan dataset about covid-19. *arXiv preprint arXiv:2003.13865*, 2020.

Chuansheng Zheng, Xianbo Deng, Qing Fu, Qiang Zhou, Jiapei Feng, Hui Ma, Wenyu Liu, and Xinggang Wang. Deep learning-based detection for covid-19 from chest ct using weak label. *medRxiv*, 2020. doi: 10.1101/2020.03.12.20027185. URL https://www.medrxiv.org/content/early/2020/03/26/2020.03.12.20027185.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*, pp. 14747–14756, 2019.

Han Zou, Yuxun Zhou, Jianfei Yang, Huihan Liu, Hari Prasanna Das, and Costas J Spanos. Consensus adversarial domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5997–6004, 2019.

Appendix

In §A, we discuss the broader impact of this work on and beyond machine learning literature. In §B, we provide a tabular summary of the related work to assist the reader in understanding the contributions of this paper. §C presents the additional results supporting our analysis in §4.3. In §D, we discuss some implementation details, including how Wasserstein Distance is computed in a completely distributed manner. We also discuss how we modify the baselines for a fair comparison with FRUDA. In §E, we provide the proofs and theoretical justifications behind our proposed solutions. Finally, in §F, we share the complete experiment details to aid reproducibility.

A Broader Impact

Overview. As machine learning algorithms mature, we as a research community should aspire to see them deployed in realistic settings, in order to solve both technical and broader global challenges. It is also essential that ML algorithms work universally for users around the world and not just be limited to a subset of the population in a specific part of the world, otherwise we risk creating a world of ML haves and have-nots, i.e., people who will gain advanced capabilities in their lives with ML and others who will not. One of the hurdles in fulfilling this vision is the reliance of today's deep learning algorithms on large-scale labeled datasets. Collecting realistic, *labeled* data is already expensive and time-consuming; moreover, scaling this data collection process to users and communities around the world could be infeasible for most developers.

The field of Unsupervised Domain Adaptation (uDA) carries enormous promise in this context. If we can successfully adapt pre-trained models to new users, new communities, new countries (i.e., target domains) using only *unlabeled data* collected from them, we can significantly reduce our reliance on labeled data, and still ensure that the benefits of ML reach users around the globe.

Positive Impact. Undoubtedly, in the last 5 years, we have seen exciting advancements and hundreds of research papers that have proposed new adversarial uDA algorithms to increase *adaptation accuracy* in a target domain. However, for real-world impact, these accuracy improvements in uDA need to go hand-in-hand with practical settings in which ML systems will be deployed. We observe that there is not much research on understanding the bottlenecks of uDA in practical ML systems (e.g., scalability, privacy) that may hinder the adoption of these algorithms.

As such, one significant impact of our work is in providing a complementary and practical perspective on uDA research, by critically evaluating uDA techniques from a cross-disciplinary lens at the intersection of Machine Learning and Distributed Systems. By identifying the bottlenecks of scalability and privacy, and proposing solutions for them, this paper takes a step towards making uDA techniques more practical and usable in realistic settings.

Ethical Considerations. As with any distributed algorithm that sends data over the communication network, we need to remain vigilant about privacy attacks. Although our strategy protects against gradient leakage attacks currently known in the literature, we are mindful that more sophisticated privacy attacks will be developed in the future. Hence, we need to be constantly aware of the developments in the ML privacy literature and be ready to make amendments to our proposed technique.

This paper also argues for more *cross-disciplinary* research inside the uDA community. While developing and evaluating new uDA algorithms, we should evaluate them not just on 'accuracy' as a metric, but holistically from the perspective of privacy, security, fairness, and their adherence to practical systems realities. This would mean broadening our research scope to allow complementary views to contribute to uDA, including those from Systems research, Privacy, Ethics and Fairness research. This type of cross-disciplinary investigation of uDA would certainly enrich it and lead to much broader impact.

We hope that this paper, at the intersection of uDA and Distributed Systems, is considered a positive step in this direction.

B Overview of Related Work

In Table 4, we provide a tabular summary of the related work in unsupervised domain adaptation. This builds upon §5 in the main paper, and highlights how our framework, FRUDA, provides novel contributions to the uDA literature.

Property \ Method	**ADDA, DANN, CADA**	**Multi Source uDA**	**Federated uDA (FADA)**	**Tan et al. (2015)**	**FRUDA (Ours)**
Adversarial Domain Adaptation	✓	✓	✓		✓
Collaborator Selection before Adaptation				✓	✓
Collaborator Selection Metric				A-Distance, Domain Complexity	Estimated Target Cross-Entropy Error
Supports Distributed uDA			✓		✓
Privacy Protection in uDA Against Gradient Leakage					✓
Communication Efficient Distributed uDA					✓
Framework to support multiple uDA algorithms					✓
Number of labeled source domains	1	Multiple	Multiple	1	1
Dynamic Weighting of Multiple Source Domains	N/A	✓	✓	N/A	N/A

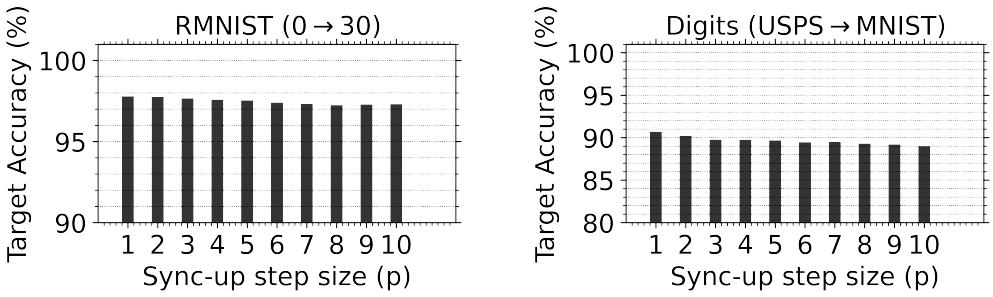
Table 4: Overview of the related work in unsupervised domain adaptation and the novelty of FRUDA over prior works. Check marks denote the core property of different methods. FRUDA is unique in providing a framework to scale multiple adversarial uDA algorithms using optimal collaborator selection and privacy-preserving, communication-efficient distributed training.

C Additional Results

C.1 Effect of sync-up step p

In DILS, p controls the frequency of gradient exchange between the nodes. In Figures 3a and 3b, we vary p from 1 and 10 and calculate the adaptation accuracy in the target domain for two adaptation tasks. The results show that DILS is fairly resilient to step-size up to $p = 10$. The difference in adaptation accuracy between $p = 10$ and $p = 1$ is just 0.5% for the RMNIST task and 1.7% for the Digits task. This accuracy loss is offset by the gain in training speed; when p is high, DILS exchanges less gradients over the communication network, hence increasing the training speed.

Effectively, p could be considered as a tunable parameter to trade-off adaptation accuracy and training time. For applications where it is important to minimize the training time, p could be set to a high value. Empirically, we find that $p = 4$ provides a good tradeoff between accuracy and training speed for all datasets studied in this paper.



(a) **(b)**

Figure 3: Effect of varying the sync-up step p from 1 to 10 on target domain accuracy.

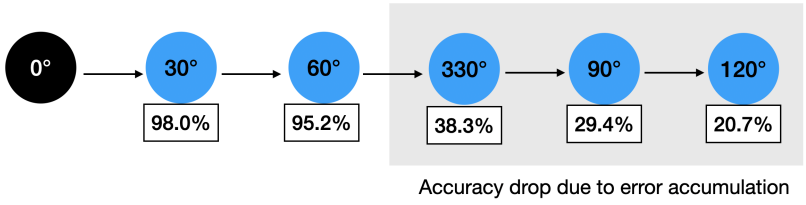
C.2 Error propagation and negative transfer in sequential adaptation

In sequential adaptation where a target domain adapts from previous domain(s), error propagation and negative transfer is a possibility, especially if an unrelated domain appears in the sequence.

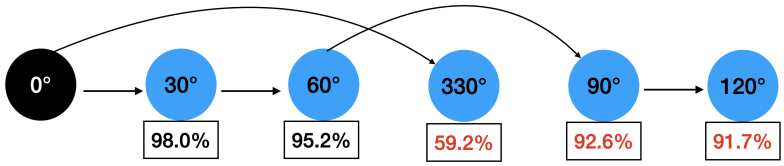
To demonstrate this phenomenon, Figure 4a shows a sequence of one source domain (0°) and 5 target domains (30°, 60°, 300°, 90°, 120°) from the RMNIST dataset. If we simply do a sequential adaptation where each domain i adapts from the $(i-1)^{th}$ domain, we see that 60°→300° results in high error and poor adaptation accuracy for 300°. This behavior is caused by the high divergence between the two domains. More critically however, we see that this error propagates in all the subsequent adaptation tasks (for 90°, 120°) and causes poor adaptation performance in them as well. In fact, for 120° we also observe negative transfer, as its post-adaptation accuracy (20.7%) is worse than the pre-adaptation accuracy obtained with source domain model.

The ability of OCS to flexibly choose a collaborator for each target domain inherently counters this problem. Firstly, Figure 4b shows that we can choose a better collaborator for 300° and obtain almost 20% higher adaptation accuracy. More importantly, the subsequent target domains are no longer reliant on 300° as their collaborator and can adapt from any available candidate domain, e.g., 90° adapts from 60° and obtains the best possible adaptation accuracy of 92.6% in this setup. Similarly, for 120°, we can prevent the negative transfer and achieve 91.7% accuracy by adapting from 90° (which itself underwent adaptation previously with 60°).

In summary, for a given sequence of domains, OCS enables each target domain to flexibly find its optimal collaborator and achieve the best possible adaptation accuracy.



(a) Sequential Adaptation without OCS. Each domain i adapts from the $(i-1)^{th}$ domain.



(b) Sequential Adaptation with OCS. Each domain flexibly chooses its optimal collaborator and obtains the best possible adaptation accuracy (indicated in red).

Figure 4: OCS prevents negative transfer and error propagation in sequential adaptation caused by the presence of unrelated domains. 0° is the labeled source domain while the domains in blue are unlabeled target domains. The numbers in rectangle denote the post-adaptation accuracy for a domain.

	Sources = 0°, 30°, 60° **Target = 90°**	**Sources = 0°,30°,60°,300°,270°,180°,45°,15°** **Target = 90°**
Vanilla MDAN (Without OCS)	82.1%	76.3 %
MDAN with OCS	88.4%	89.3 %

Table 5: OCS can improve the performance of multi-source uDA algorithms by selecting multiple collaborators for each target domain.

C.3 CAN OCS BENEFIT MULTI-SOURCE SETTINGS?

OCS can also boost the adaptation accuracy in settings with multiple labeled sources. We present an experiment with MDAN (Zhao et al. (2018)) which is an algorithm for multi-source domain adaptation.

Table 5 shows the performance of MDAN in two adaptation tasks on RMNIST with 3 and 8 source domains, and one target domain. Each source domain is labeled. In vanilla MDAN, we use all available source domains for adaptation without selecting any optimal collaborators. In MDAN with OCS, we first select multiple optimal collaborators using OCS and then perform MDAN between the collaborators and the target domain. The OCS formulation in Equation 5 can be trivially extended to select the top-n collaborators instead of only one collaborator.

For the first task, we set n=2 and the two collaborators returned by OCS are 30° and 60°. For the second task, we set n=3 and the collaborators returned by OCS are 30°, 60° and 45°. The results in Table 5 show that by selecting the optimal collaborators for the target domain, we can achieve 6.3% and 13% gains over vanilla MDAN.

We believe that the accuracy drop in vanilla MDAN is caused by negative transfer due to the presence of source domains unrelated to the target domain, e.g., in the second adaptation task, there are 8 source domains many of which are far-off from the target domain (90°). On the other hand, OCS filters out these unrelated domains before initiating the adaptation, thereby preventing any negative transfer due to them.

D IMPLEMENTATION DETAILS

D.0.1 LOSS FORMULATIONS OF UDA ALGORITHMS

We now discuss the adversarial training formulation of the various uDA algorithms with which we evaluated the efficacy of FRUDA. Recall that we evaluate our method with four domain adaptation techniques: ADDA (Tzeng et al. (2017)), Gradient Reversal (Ganin et al. (2016)) and Wasserstein DA (Shen et al. (2018)), and CADA (Zou et al. (2019)). Below are the adversarial training formulations of these techniques as proposed in their original papers.

ADDA. Following the same notations used earlier in the paper, the adversarial loss formulations of ADDA can be represented mathematically as:

$$\min_{DI} \mathcal{L}_{adv_{DI}} = -\mathbb{E}_{x_s \sim \mathcal{X}_S}[\log(DI(E_S(x_s))] - \mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(1 - DI(E_T(x_t))] \tag{6}$$

$$\min_{E_T} \mathcal{L}_{adv_M} = -\mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(DI(E_T(x_t))] \tag{7}$$

The Discriminator DI is optimized using $L_{adv_{DI}}$ where the domain data from source and target domains are assigned different domain labels (0 and 1). To update the feature extractor E_T, ADDA proposes to invert the domain labels, which results in the loss formulation given in L_{adv_M}.

In order to run ADDA in a distributed manner, we decompose the discriminator DI into DI_S and DI_T which results in the following loss functions:

$$\mathcal{L}_{adv_{DI_S}} = -\mathbb{E}_{x_s \sim \mathcal{X}_S}[\log(DI_S(E_S(x_s))] \tag{8}$$

$$\mathcal{L}_{adv_{DI_T}} = -\mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(1 - DI_T(E_T(x_t))] \tag{9}$$

$$\min_{E_T} \mathcal{L}_{adv_M} = -\mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(DI_T(E_T(x_t))] \tag{10}$$

Thereafter, we compute local gradients for DI_S and DI_T,

$$\nabla g(DI_S, x_s) = \frac{\delta L_{adv_{DI_S}}}{\delta DI_S} \tag{11}$$

$$\nabla g(DI_T, x_t) = \frac{\delta L_{adv_{DI_T}}}{\delta DI_T} \tag{12}$$

and aggregate them in the sync-up step as shown in Algorithm 1. The aggregated gradients are used to optimize DI_S and DI_T, while E_T is optimized using the loss function in Equation 10.

Gradient Reversal. The Gradient Reversal approach uses the same loss formulation for the discriminators DI_S and DI_T as ADDA as shown in Equations 6, 8, 9.

However, to update the target extractor E_T, it leverages the gradient reversal strategy, resulting in the following loss function.

$$\begin{aligned} \min_{E_T} \mathcal{L}_{adv_M} &= -\mathcal{L}_{adv_{DI_T}} \\ &= \mathbb{E}_{x_t \sim \mathcal{X}_T}[\log(1 - DI_T(E_T(x_t))] \end{aligned} \tag{13}$$

The computation of local gradients for DI_S and DI_T follow the same process as shown in Equations 11 and 12.

Wasserstein DA. In this technique, Shen et al. (2018) use the Wasserstein distance as the loss function for the discriminator. Wasserstein distance between two datasets is defined as

$$\text{Wasserstein}(X_s, X_t) = \frac{1}{n_s} \sum_{x_s \sim \mathcal{X}_S} DI(E_S(x_s)) - \frac{1}{n_t} \sum_{x_t \sim \mathcal{X}_T} DI(E_T(x_t)) \tag{14}$$

where n_s and n_t are the number of samples in the dataset. The discriminator loss is computed as:

$$\min_{DI} \mathcal{L}_{adv_D I} = -\mathbb{E}_{x_s \sim \mathcal{X}_S, x_t \sim \mathcal{X}_T}[\text{Wasserstein}(x_s, x_t)] + \gamma L_{grad} \tag{15}$$

where L_{grad} is the gradient penalty used to enforce the Lipschitz constraint on the discriminator. Further, the target extractor is optimized using the following loss function:

$$\min_{E_T} \mathcal{L}_{adv_M} = \mathbb{E}_{x_s \sim \mathcal{X}_S, x_t \sim \mathcal{X}_T}[\text{Wasserstein}(x_s, x_t)] \tag{16}$$

We use the same strategy to compute local gradients for DI_S and DI_T as shown in Eq 11 and 12.

CADA. Consensus Adversarial Domain Adaptation is a technique proposed by Zou et al. (2019) which enforces the source and target extractors to arrive at a consensus in the feature space through adversarial training. It uses the same loss formulation for the discriminators DI_S and DI_T as ADDA as shown in Equations 6, 8, 9. However, the key difference is that CADA optimizes both the source and target feature extractors in the training process, until the discriminator can no longer distinguish the features from source and target domains.

D.1 COMPUTING WASSERSTEIN DISTANCE ACROSS DISTRIBUTED DATASETS IN A PRIVACY PRESERVING MANNER

Our optimal collaborator selection algorithm requires computing an estimate of the Wasserstein (W_1) distance between a candidate domain (D_C) and the target domain (D_T). Let X_C and X_T denote the unlabeled datasets from the two domains. As shown by Shen et al. (2018), the W_1 distance can be computed as:

$$W_1(X_C, X_T) = \frac{1}{n_C} \sum_{x_s \sim \mathcal{X}_C} DI(E_C(x_c)) - \frac{1}{n_T} \sum_{x_t \sim \mathcal{X}_T} DI(E_T(x_t)) \tag{17}$$

where n_C and n_T are the number of samples in the dataset, E_C and E_T are the feature encoders of each domain, and DI is an optimal discriminator trained to distinguish the features from the two domains. To train the optimal discriminator, following loss is minimized:

$$\min_{DI} \mathcal{L}_{adv_{DI}} = -\mathbb{E}_{x_c \sim \mathcal{X}_C, x_t \sim \mathcal{X}_T}[W_1(x_c, x_t) + \gamma L_{grad}]$$

where L_{grad} is the gradient penalty used to enforce 1-Lipschitz continuity on the discriminator.

Interestingly, Equation 17 has a similar structure to the optimization objectives for ADDA and other uDA algorithms discussed above. Hence, we can use the same principle as DILS and exchange discriminator gradients between nodes to compute the Wasserstein Distance in a distributed manner, without requiring any exchange of raw data.

We initialize E_T with E_C and decompose the discriminator DI into two parts (DI_C and DI_T) which reside on the respective nodes. The raw data from both nodes is fed into their respective encoders and discriminators, and we compute the gradients of each discriminator as follows:

$$\mathcal{L}^C_{DI} = \frac{1}{n_C} \sum_{x_s \sim \mathcal{X}_C} DI_C(E_C(x_c))$$

$$\mathcal{L}^T_{DI} = \frac{1}{n_T} \sum_{x_t \sim \mathcal{X}_T} DI_T(E_T(x_t))$$

$$\nabla g(DI_C, x_c) = \frac{\delta \mathcal{L}^C_{DI}}{\delta DI_C}$$

$$\nabla g(DI_T, x_t) = \frac{\delta \mathcal{L}^T_{DI}}{\delta DI_T}$$

Both nodes exchange their discriminator gradients during a synchronization step and compute aggregated gradients:

$$\nabla g(DI_{\text{agg}}, x_c, x_t) = \nabla g(DI_C, x_c) - \nabla g(DI_T, x_t) \tag{18}$$

Finally, both discriminators DI_C and DI_T are updated with these aggregated gradients, and gradient penalty is applied to enforce the 1-Lipschitz continuity on the discriminators. This process continues until convergence and results in an optimal discriminator. Once the discriminators are trained to convergence, we can calculate the Wasserstein distance as:

$$W_1(X_C, X_T) = \mathcal{L}^C_{DI} - \mathcal{L}^T_{DI}$$

To the best of our knowledge, this approach of computing Wasserstein distance in a distributed manner has not been explored before. Moreover, this ability to compute Wasserstein distance in a distributed manner makes it an ideal metric for collaborator selection in a privacy-preserving setting.

D.2 MODIFICATIONS DONE TO MDAN

As we noted in §4.2, we use a Multi-Collaborator baseline for OCS wherein all available candidate domains contribute in domain adaptation with the target. To this end, we employ the MDAN method

proposed by Zhao et al. (2018) for multi-source domain adaptation. However, one key difference between MDAN and our problem setting is that MDAN assumes that there are multiple source domains and all of them are labeled. With this assumption, MDAN optimizes the following objective during uDA:

$$\text{minimize} \ \frac{1}{\gamma} \sum_{i \in [k]} \exp(\gamma(\hat{\epsilon}_{S_i}(h) - \min_{h' \in H\Delta H} \hat{\epsilon}_{T,S_i}(h')))$$

where $\hat{\epsilon}_{S_i}$ is the classification error for source domain S_i obtained using supervised learning (assuming the availability of labels), and $\hat{\epsilon}_{T,S_i}$ is the error of a domain discriminator trained to separate S_i and target T.

However, in our problem setting, there is only one labeled source and all other domains are unlabeled. As such, we do not have a way to optimize the classification error $\hat{\epsilon}_{S_i}$ for all candidate domains. Therefore, we only optimize the discriminator error proposed in MDAN.

E PROOFS

E.1 CONVERGENCE OF DISCRIMINATOR-BASED LAZY SYNCHRONIZATION (DILS)

The network structures of adversarial uDA methods resemble a GAN, where the target encoder E_T and discriminators DI (DI_S and DI_T) play a minimax game similar to a GAN's generator and discriminator. E_S and E_T can separately define two probability distributions on the feature representations $E_S(x_s), x_s \sim \mathcal{X}_S$ and $E_T(x_t), x_t \sim \mathcal{X}_T$, noted as p_s and p_t respectively. Then according to the theoretical analysis in Goodfellow et al. (2014), we know that:

Proposition 2. For a given E_T, if DI is allowed to reach its optimum, and p_t is updated accordingly to optimize the value function, then p_t converges to p_s, which is the optimization goal of E_T.

In other words, if we can guarantee that under the training strategy of *Lazy Synchronization*, convergence behaviour of DI_S and DI_T is similar to the non-distributed case, then E_T should also converge. We have the following theorem.

Theorem 1. In *Lazy Synchronization*, given a fixed target encoder, under certain assumptions, we have the following convergence rate for the discriminators DI_S and DI_T.

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}[\frac{f(x_0)-f(x_{t+1})}{\mu T}+(2p-1)L\mu\sigma^2+\frac{\mu L\sigma^2}{2}+\frac{\mu^3L^3\sigma^2(2p-1)}{2}] \tag{19}$$

Where p is the sync-up step, μ is the learning rate. Set $\mu = O(1/\sqrt{T})$. When $p << L\sigma^2$, the impact of stale update will be very small, and thus it can converge with rate $O(1/\sqrt{T})$, which is same as the classic SGD algorithm.

Proof. As we mentioned in the paper, DI_S and DI_T are lazily synced, such that their weights are always identical, because they are initialized with the same weights and always apply the same synced gradients at every training step. Therefore, we can consider them as one discriminator DI_{lazy} for our convergence analysis.

Notations. Throughout the proof, we use following notations and definitions:

- x denotes model weights of DI_{lazy}.
- f denotes the loss function of DI_{lazy}.
- $\mathcal{D}_s$ and $\mathcal{D}_t$ denote the datasets of source and target feature representations (i.e., outputs of the respective feature extractors).
- ξ denotes one batch of training instances sampled from $\mathcal{D}_s \cup \mathcal{D}_t$.
- $f(x, \xi)$ denotes empirical loss of model x on batch ξ.
- ∇f$(\cdot)$ denotes gradients of function f.

- σ denotes the gradient bound.
- μ denotes learning rate.
- T denotes the number of training steps.
- p denotes the size of sync-up step in our proposed Lazy Synchronization approach.

We can formalize the optimization goal of the discriminator as:

$$\min_{x\sim\mathbb{R}^N} f(x) = \mathbb{E}_{r^{(s)}\sim\mathcal{D}_s} \log x(r^{(s)}) + \mathbb{E}_{r^{(t)}\sim\mathcal{D}_t} \log[1 - x(r^{(t)})] \tag{20}$$

Assumption 1. $f(x)$ is with L-Lipschitz gradients:

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2^2 \leq L \|x_1 - x_2\|_2^2 \tag{21}$$

Equivalently, we can get:

$$f(x_2) \leq f(x_1) + \nabla f(x_1)^T (x_2 - x_1) + \frac{L}{2} \|x_2 - x_1\|_2^2 \tag{22}$$

In training step t, we first simplify the updating rule as:

$$x_{t+1} = x_t - \mu \Delta(x_t, \xi_t) \tag{23}$$

Combine Inequality 22 and Equation 23, at time step t we have:

$$\begin{aligned} f(x_{t+1}) &\leq f(x_t) - \mu \nabla f(x_t) \Delta(x_t, \xi_t) + \frac{\mu^2 L}{2} \|\Delta(x_t, \xi_t)\|_2^2 \\ &... \\ f(x_1) &\leq f(x_0) - \mu \nabla f(x_0) \Delta(x_0, \xi_0) + \frac{\mu^2 L}{2} \|\Delta(x_0, \xi_0)\|_2^2 \end{aligned} \tag{24}$$

Sum all inequalities:

$$f(x_{t+1}) \leq f(x_0) - \mu \sum_{t=1}^{T} \nabla f(x_t) \Delta(x_t, \xi_t) + \sum_{t=1}^{T} \frac{\mu^2 L}{2} \|\Delta(x_t, \xi_t)\|_2^2 \tag{25}$$

Take expectation on ξ_t in both sides and re-arrange terms:

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\nabla f(x_t) \Delta(x_t, \xi_t)] \leq \frac{f(x_0) - f(x_{t+1})}{\mu T} + \frac{\mu L}{2T} \sum_{t=1}^{T} \mathbb{E}[\|\Delta(x_t, \xi_t)\|_2^2] \tag{26}$$

In our proposed Lazy Synchronization algorithm, the update term $\Delta(x_t, \xi_t)$ is:

$$\begin{aligned} &\Delta(x_t, \xi_t) = \frac{1}{p} \sum_{n=0}^{p-1} \nabla f(x_{t_p - n}, \xi_{t_p - n}), t \geq p \\ &t_p = \lfloor t/p \rfloor \times p \end{aligned} \tag{27}$$

where t_p is the latest sync-up step given a certain time step t, and p is the sync-up step of our algorithm. This updating rule formalizes our approach, wherein the gradient applied to the discriminator is the averaged gradient over p steps before the latest sync up step.

We transform $\Delta(x_t, \xi_t)$ as follows:

$$\begin{aligned} \Delta(x_t, \xi_t) &= \frac{1}{p} \sum_{n=0}^{p-1} \nabla f(x_{t_p - n}, \xi_{t_p - n}) \\ &= \frac{1}{p} \sum_{n=0}^{p-1} \nabla f(x_{t_p - n}, \xi_{t_p - n}) - \frac{1}{p} \sum_{n=0}^{p-1} \nabla f(x_t, \xi_{t_p - n}) + \frac{1}{p} \sum_{n=0}^{p-1} \nabla f(x_t, \xi_{t_p - n}) - \nabla f(x_t) + \nabla f(x_t) \\ &= \nabla f(x_t) + \frac{1}{p} \sum_{n=0}^{p-1} [\nabla f(x_{t_p - n}, \xi_{t_p - n}) - \nabla f(x_t, \xi_{t_p - n})] + \frac{1}{p} \sum_{n=0}^{p-1} [\nabla f(x_t, \xi_{t_p - n}) - \nabla f(x_t)] \end{aligned} \tag{28}$$

Bring Equation 28 back to Equation 26 (Let $e(x_t,\xi_t) = \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] + \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)]$):

$$\begin{aligned}
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\nabla f(x_t)(\nabla f(x_t)+e(x_t,\xi_t))] \le \frac{f(x_0)-f(x_{t+1})}{\mu T} + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|(\nabla f(x_t)+e(x_t,\xi_t))\|_2^2] \\
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] + \frac{1}{T}\sum_{t=1}^{T}\nabla f(x_t)\mathbb{E}[e(x_t,\xi_t)] \le \frac{f(x_0)-f(x_{t+1})}{\mu T} + \frac{\mu L}{2T}\sum_{t=1}^{T}\|\nabla f(x_t)\|_2^2 + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|e(x_t,\xi_t))\|_2^2] \\
&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \le \frac{2}{2-\mu L}\left(\frac{f(x_0)-f(x_{t+1})}{\mu T} - \frac{1}{T}\sum_{t=1}^{T}\nabla f(x_t)\mathbb{E}[e(x_t,\xi_t)] + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|e(x_t,\xi_t))\|_2^2]\right)
\end{aligned} \tag{29}$$

Then we analyze $\mathbb{E}[e(x_t,\xi_t)]$ and $\mathbb{E}[\|e(x_t,\xi_t))\|_2^2]$:

$$e(x_t,\xi_t) = \underbrace{\frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})]}_{e_1} + \underbrace{\frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)]}_{e_2} \tag{30}$$

For e_1:

$$\begin{aligned}
e_1(x_t,\xi_t) =& \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] \\
=& \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_{t_p-n+1},\xi_{t_p-n}) + \nabla f(x_{t_p-n+1},\xi_{t_p-n}) - ... \\
& - \nabla f(x_{t_p},\xi_{t_p-n}) + \nabla f(x_{t_p},\xi_{t_p-n}) - ... + \nabla f(x_{t-1},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] \\
=& \frac{1}{p}\sum_{n=0}^{p-1}\left(\sum_{i=0}^{n-1}[\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})] + \sum_{j=0}^{t-t_p-1}[\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})]\right)
\end{aligned} \tag{31}$$

According to the algorithm, we know that:

$$\begin{aligned}
x_{t_p-i} &= x_{t_p-(i+1)} - \mu G_{(\lfloor t/p\rfloor-1)\times p} \\
x_{t_p+j+1} &= x_{t_p+j} - \mu G_{\lfloor t/p\rfloor\times p}
\end{aligned} \tag{32}$$

where $G_{(\lfloor t/p\rfloor-1)\times p}$ is the second last synced gradient, $G_{\lfloor t/p\rfloor\times p}$ is the latest synced gradient.

Apply L-Lipschitz gradient:

$$\begin{aligned}
\left\|\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})\right\| &\le L\mu\left\|G_{(\lfloor t/p\rfloor-1)\times p}\right\| \\
\left\|\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})\right\| &\le L\mu\left\|G_{\lfloor t/p\rfloor\times p}\right\|
\end{aligned} \tag{33}$$

Assumption 2. Bounded Gradient. We assume the stochastic gradients are uniformly bounded: (see Shalev-Shwartz et al. (2011); Nemirovski et al. (2009); Hazan & Kale (2014))

$$\mathbb{E}[\|\nabla f(x_t,\xi_t)\|_2^2] \le \sigma^2 \tag{34}$$

Therefore, we have:

$$\begin{aligned}
\left\|\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})\right\| &\le L\mu\,\sigma \\
\left\|\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})\right\| &\le L\mu\sigma
\end{aligned} \tag{35}$$

Bring Inequality 35 back to Equality 31:

$$\begin{aligned} e_1(x_t, \xi_t) &\leq \frac{1}{p}\sum_{n=0}^{p-1}(n + t - t_p)L\mu\sigma \\ &\leq (2p-1)L\mu\sigma \end{aligned} \tag{36}$$

since it is easy to see $t - t_p \leq p$.

In summary of e_1, we get:

$$\begin{aligned} \mathbb{E}[\nabla f(x_t)e_1(x_t, \xi_t)] \leq \mathbb{E}[\|\nabla f(x_t)\|] \times \|e_1(x_t, \xi_t)\|] \leq (2p-1)L\mu\sigma^2 \\ \mathbb{E}[\|e_1(x_t, \xi_t))\|_2^2] \leq (2p-1)L^2\mu^2\sigma^2 \end{aligned} \tag{37}$$

For $e_2 = \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t, \xi_{t_p-n}) - \nabla f(x_t)]$:

Since batch ξ is an unbiased sampled batch, it is obvious that for a certain n:

$$\mathbb{E}[\nabla f(x_t, \xi_{t_p-n}) - \nabla f(x_t)] = 0 \tag{38}$$

Also,

$$\begin{aligned} \mathbb{E}[\left\|\nabla f(x_t, \xi_{t_p-n}) - \nabla f(x_t)\right\|_2^2] &\leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \mathbb{E}[\left\|\nabla f(x_t, \xi_{t_p-n})\right\|_2^2] \\ &\leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \sigma^2 \end{aligned} \tag{39}$$

Therefore,

$$\begin{aligned} \mathbb{E}[e_2(x_t, \xi_t)] = 0 \\ \mathbb{E}[\|e_2(x_t, \xi_t))\|_2^2] \leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \sigma^2 \end{aligned} \tag{40}$$

Combine 37 and 40 with 29:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}[\frac{f(x_0) - f(x_{t+1})}{\mu T} + (2p-1)L\mu\sigma^2 + \frac{\mu L\sigma^2}{2} + \frac{\mu^3L^3\sigma^2(2p-1)}{2}] \tag{41}$$

Therefore, the expected $\|\nabla f(x_t)\|_2^2$ converge to 0 with rate $O(1/\sqrt{T})$ if $\mu = 1/\sqrt{T}$. When the function $f(x)$ is strongly convex, this is the global minimum, otherwise it can be a local minimum, or a stationary point.

E.2 Privacy Analysis of DILS

Recall that a key feature of our distributed training algorithm is to exchange information between the distributed nodes using *gradients of the discriminators*. This clearly affords certain privacy benefits over existing uDA algorithms since we no longer have to transmit raw training data between nodes. However, prior works have shown that model gradients can potentially leak raw training data in collaborative learning (Melis et al. (2019)), therefore it is critical to examine: *can the discriminator gradients also indirectly leak training data of a domain*? We begin by providing some theoretical justification on why our training algorithm prevents the reconstruction of raw data from discriminator gradients. Then in §E.2.1, we study the performance of DILS under a state-of-the-art gradient leakage attack proposed by Zhu et al. (2019).

We consider the case when the domain discriminator is made of a single neuron and later, we will explain how this analysis generalizes to deeper discriminators.

Let $x = (x_1, x_2, ...x_n) \in \mathbb{R}^n$ be a n-dimensional training point sampled from the target domain. Let the target domain feature extractor E_T be a neural network which outputs a j-dimensional feature vector $e = \left(e_1, e_2, ...e_j\right) \in \mathbb{R}^j$. The squared error discriminator loss $\mathcal{L}_D$ is expressed as:

$$\mathcal{L}_D = \left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y\right)^2 \tag{42}$$

where W_i (i = 1...j), b and g are respectively the weights, bias, and activation function of the target discriminator, and y is the target domain label. Under this formulation, the gradient with respect to discriminator weight W_j is given as:

$$\begin{aligned}
\nabla_{W_j} &= \frac{\delta L_D}{\delta W_j} \\
&= \frac{\delta \left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right)^2}{\delta W_j} \\
&= 2\left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right) \frac{\delta \left(g\left(\sum_{i=1}^{j} W_i e_i + b\right)\right)}{\delta W_j} \\
&= 2\left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right) \left(g'\left(\sum_{i=1}^{j} W_i e_i + b\right)\right) \frac{\delta \left(\sum_{i=1}^{j} W_i e_i + b\right)}{\delta W_j} \\
&= 2\left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right) \left(g'\left(\sum_{i=1}^{j} W_i e_i + b\right)\right) \cdot e_j
\end{aligned} \tag{43}$$

Similarly, the gradient with respect to the bias b is given as:

$$\begin{aligned}
\nabla_b &= \frac{\delta L_D}{\delta b} \\
&= \frac{\delta \left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right)^2}{\delta b} \\
&= 2\left(g\left(\sum_{i=1}^{j} W_i e_i + b\right) - y \right) \left(g'\left(\sum_{i=1}^{j} W_i e_i + b\right)\right) \cdot 1
\end{aligned} \tag{44}$$

Note that $\frac{\nabla_{W_j}}{\nabla_b} = e_j$, i.e., if an adversary steals the gradients of the weights and bias of the discriminator, they can potentially (and, at best) reconstruct the feature representation e of the training input. It is infeasible to reconstruct the input raw data x from the stolen features e, because our training strategy does not exchange the feature extractor model E_T between the nodes. Also note that by adding regularization during the training process (e.g., using Dropout or L2 regularization), even reconstruction of the entire feature representations e can be prevented.

Although we provided the above justification assuming that the discriminator consists of a single neuron, our conclusion extends to scenarios where the discriminator is a neural network. Here, the gradients of the first hidden layer of the discriminator network can result in feature leakage:

$$\begin{aligned}
\nabla_{W^{(1)}_{c,j}} &= \frac{\delta L_D}{\delta W^{(1)}_{c,j}} \\
&= \theta \cdot e_j
\end{aligned} \tag{45}$$

where, $W^{(1)}_{c,j}$ is the weight connecting the j^{th} encoded feature e_j with the c^{th} node in the first hidden layer of the discriminator. θ is a real number. As described earlier, the gradient in Equation 45, in conjunction with the gradient of the bias can at best result in reconstructing the feature representation e, however it does not reveal the training data x from the target domain.

E.2.1 PROTECTION AGAINST A STATE-OF-THE-ART PRIVACY ATTACK

The above analysis was conceptual in nature and intended to give an intuition on the privacy properties of DILS. We now take a recently proposed privacy attack as an example and study the behavior of DILS under it.

Zhu et al. (2019) showed that gradient matching can be a simple but robust technique to reconstruct the raw training data from stolen gradients. Let us say we are given a machine learning model $F()$ with weights W. Let ∇W be the gradients with respect to a private input pair $(\mathbf{x}, y)$. During distributed training, ∇W are exchanged between the nodes.

A reconstruction attack happens as follows: an attacker first randomly initializes a dummy input $\mathbf{x}'$ and label input y'. This data is fed into the model $F()$ to compute dummy gradients as follows:

$$\nabla W' = \frac{\partial \ell \left(F\left(\mathbf{x}', W\right), \mathbf{y}'\right)}{\partial W}$$

Finally, the attacker minimizes the distance between the dummy gradients and the actual gradients using gradient descent to reconstruct the private data as follows:

$$\mathbf{x}'^*, \mathbf{y}'^* = \arg\min_{\mathbf{x}', \mathbf{y}'} \left\| \nabla W' - \nabla W \right\|^2 = \arg\min_{\mathbf{x}', \mathbf{y}'} \left\| \frac{\partial \ell \left(F\left(\mathbf{x}', W\right), \mathbf{y}'\right)}{\partial W} - \nabla W \right\|^2 \tag{46}$$

Zhu et al. (2019) demonstrate the success of this attack on a number of image datasets.

Can this attack succeed on DILS? There are two key assumptions in this attack: (i) the weights W of the end-to-end machine learning model are available to an adversary in order for them to compute the dummy gradients, (ii) the gradients of all the layers (∇W) between the input x and output y are available to the adversary.

DILS never exchanges the weights of the target domain model (i.e., the feature encoder and the discriminator) during the adversarial training process. As shown in Algorithm 1, the target feature encoder is trained locally and only discriminator gradients are exchanged. Without the knowledge of the model weights W, an attacker can not generate the dummy gradients $\nabla W'$ necessary to initiate the attack on the target domain.

Looking at the source or collaborator domain, we do exchange its feature encoder with the target domain in the initialization step of uDA, which could be used by the attacker to generate the dummy gradients $\nabla W'$. However, for the attack to succeed, the attacker also needs the real gradients (∇W) of all the layers between the input x and output y in the source domain. This includes gradients of the feature encoder E_S and the domain discriminator DI_S. In DILS however, we only exchange the gradients of the domain discriminator DI_S during training; the gradients of E_S are never exchanged. Without the knowledge of the gradients of E_S, an attacker cannot use Eq. 46 to reconstruct the training data of the source domain.

In summary, we have proven that our strategy of distributed uDA based on discriminator gradients does not allow an attacker to reconstruct the private data of either the source or the target domain.

E.3 Optimal Collaborator Selection (OCS) algorithm

Theorem 2 *Let D_1 and D_2 be two domains sharing the same labeling function l. Let θ_{CE} denote the Lipschitz constant of the cross-entropy loss function in D_1, and let θ be the Lipschitz constant of a hypothesis learned on D_1. For any two θ-Lipschitz hypotheses h, h', we can derive the following error bound for the cross-entropy (CE) error $\varepsilon_{\mathrm{CE,D_2}}$ in D_2:*

$$\varepsilon_{\mathrm{CE,D_2}}(h, h') \leq \theta_{\mathrm{CE}} \left(\varepsilon_{\mathrm{L_1,D_1}}(h, h') + 2\theta W_1(D_1, D_2) \right) \tag{47}$$

where $W_1(D_1, D_2)$ denote the first Wasserstein distance between the domains D_1 and D_2, and $\varepsilon_{\mathrm{L_1,D_1}}$ denotes the L_1 error in D_1.

Proof. The L_1 error between two hypotheses h, h' on a distribution D is given by:

$$\varepsilon_{\mathrm{L_1,D}}(h, h') = \mathbb{E}_{x \sim D} \left[|h(x) - h'(x)| \right] \tag{48}$$

We define softmax cross-entropy on a given distribution D as

$$\varepsilon_{\mathrm{CE,D}}(h) = \mathbb{E}_{x \sim D} \left[\left| \log S_{l(x)} h(x) \right| \right], \tag{49}$$

where S is the softmax function $\mathbb{R}^n \longrightarrow \mathbb{R}^n$, l is the labelling function, and $S_{l(x)}$ denotes the projection of S to the $l(x)$-component.

Then we have,

$$\begin{aligned}\varepsilon_{\mathrm{CE,D}}(h, h') &= \mathbb{E}_{x\sim D}\left[\left|\log S_{l(x)}h(x) - \log S_{l(x)}h'(x)\right|\right] \\ &= \varepsilon_{\mathrm{L_1,D}}\left(\log S_l h, \log S_l h'\right)\end{aligned} \tag{50}$$

Further, using the definition of Lipschitz continuity, we have

$$\left|\log S_{l(x)}h(x) - \log S_{l(y)}h(y)\right| \leq \theta_{\mathrm{CE}}|h(x) - h(y)|, \tag{51}$$

where θ_{CE} is the Lipschitz constant of the softmax cross-entropy function.

Next, we follow the triangle inequality proof from (Shen et al., 2018, proof of Lemma 1) to find that

$$\varepsilon_{\mathrm{L_1,D_2}}\left(\log S_l h, \log S_l h'\right) \leq \varepsilon_{\mathrm{L_1,D_1}}\left(\log S_l h, \log S_l h'\right) + 2\theta_{\mathrm{CE}}.\theta W_1(D_1, D_2), \tag{52}$$

where θ is a Lipschitz constant for h and h', if the label $l(x)$ were constant. Since $l(x)$ is constant outside of a measure 0 subset where the labels change, and h and h' are Lipschitz, so in particular measurable, Equation 52 holds everywhere.

Then, by substituting from Eq. 50 and Eq. 51 in Eq. 52, we get Theorem 1:

$$\begin{aligned}\varepsilon_{\mathrm{CE,D_2}}(h, h') &\leq \varepsilon_{\mathrm{CE,D_1}}(h, h') + 2\theta_{\mathrm{CE}}.\theta W_1(D_1, D_2) \\ &\leq \theta_{\mathrm{CE}}(\varepsilon_{\mathrm{L_1,D_1}}\left(h, h'\right) + 2\theta W_1\left(D_1, D_2\right))\end{aligned} \tag{53}$$

□

In effect, Theorem 1 enables the estimation of target CE error, given a hypothesis (i.e., a classifier) from a collaborator domain. Given a collaborator domain D_c, a learned hypothesis h^c and a labeling function l, we can estimate the CE error for a target domain D_T as:

$$\varepsilon_{\mathrm{CE,D_T}}\left(h^c, l\right) \leq \theta_{\mathrm{CE}}(\varepsilon_{\mathrm{L_1,D_c}}\left(h^c, l\right) + 2\theta W_1\left(D_c, D_T\right)) \tag{54}$$

Now that we have a way to estimate the target CE error, we can use it to select an optimal collaborator that yields the minimum target CE error. Let $\mathbb{Z} = \{D^k | k = 1, \ldots, K\}$ be a set of candidate domains each with a pre-trained model h^k with Lipschitz constants θ^k_{CE} and θ^k. Let D_T^{K+1} be a target domain for which an optimal collaborator is to be chosen. We use Eq. 54 to select the optimal collaborator domain D_{opt} as:

$$D_{\mathrm{opt}} = \underset{k=1,\ldots,K}{\operatorname{argmin}}\ \theta^k_{\mathrm{CE}}(\varepsilon_{\mathrm{L_1,D^k}}(h^k, l) + 2\theta^k W_1(D^k, D_T^{\mathrm{K+1}})) \tag{55}$$

Our empirical results confirm that this formulation provides a robust metric to choose optimal collaborators and increase the accuracy of uDA algorithms in multi-target scenarios.

F Experiment Details for Reproducibility

F.0.1 Domain Orderings

As mentioned in our problem formulation, the unlabeled target domains are introduced sequentially in a given order. To measure the performance of FRUDA with various target orders, we reported two different domain orders for each data set in our experiments. We specify these orders in Table 6.

F.0.2 Architectures, Pre-Processing and Hyperparameters

We now describe the neural architectures used for each dataset along with the pre-processing steps and hyperparameters used for supervised and adversarial learning.

Rotated MNIST: The MNIST dataset is obtained from the Tensorflow Dataset repository and is rotated by different degrees to generate different domains. The same training and test partitions as in the original dataset are used in our experiments. We employ the well-known LeNet architecture

Dataset	Ordering 1	Ordering 2
RMNIST	**0**,30,60,90,120,150,180, 210,240,270,300,330	**0**,180,210,240,270,300, 330,30,60,90,120,150
Mic2Mic	**Voice (V)**, USB (U), Creator (C), ReSpeaker (R)	**USB**, Creator, Voice, ReSpeaker
Digits	**MNIST Modified (M-M)**, Synth Digits (Syn), MNIST (M), USPS (U), SVHN (S)	**Synth Digits**, MNIST, MNIST Modified, USPS, SVHN
Office-Caltech	**DSLR (D)**, Webcam (W), Caltech (C), Amazon (A)	**Webcam**, Caltech, DSLR, Amazon

Table 6: Domain orderings used in our experiments. Domains in bold correspond to the labeled source domain, which is introduced first in the system. All other domains have no training labels.

for training the feature encoder as shown below. The model was trained for each source domain with a learning rate of 10^{-4} using the Adam optimizer and a batch size of 32.

```
Conv2D(filters = 20, kernel_size = 5, activation='relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Conv2D(filters = 50, kernel_size = 5, activation='relu'),
MaxPooling2D(pool_size = 2, strides = 2),
Flatten(),
Dense(500, activation='relu'),
Dense(10, activation='softmax')
```

In order to enforce the Lipschitz continuity on the encoder, we applied a constraint on the weights of each layer, which represents the upper bound of the Lipschitz constant for that layer Gouk et al. (2018). We did a hyperparameter search to find the optimal Lipschitz constant for each layer. In the adversarial training process, we used the ADDA loss formulations to perform domain adaptation with a learning rate of 10^{-3} for the target extractor and 10^{-2} for the discriminators.

Digits: This task consists of five domains: MNIST, SVHN, USPS, MNIST-Modified and SynthDigits Ganin & Lempitsky (2014). We used the same train/test split as in the original domain datasets. The images from all domains were normalized between 0 and 1, and resized to 32x32x3 for consistency. The following architecture was used for the feature encoder and it was trained for each source domain with a learning rate of 10^{-5} using the Adam optimizer and a batch size of 64.

```
inputs = tf.keras.Input(shape=(32,32,3), name='img')
x = Conv2D(filters = 64, kernel_size = 5, strides=2)(inputs)
x = BatchNormalization()(x, training=is_training)
x = Dropout(0.1)(x, training=is_training)
x = ReLU()(x)
x = Conv2D(filters = 128, kernel_size = 5, strides=1)(x)
x = BatchNormalization()(x, training=is_training)
x = Dropout(0.3)(x, training=is_training)
x = ReLU()(x)
x = Conv2D(filters = 256, kernel_size = 5, strides=1)(x)
x = BatchNormalization()(x, training=is_training)
x = Dropout(0.5)(x, training=is_training)
x = ReLU()(x)
x = Flatten()(x)
x = Dense(512)(x)
x = BatchNormalization()(x, training=is_training)
x = ReLU()(x)
x = Dropout(0.5)(x, training=is_training)
```

```
outputs = Dense(10)(x)
```

In order to enforce the Lipschitz continuity on the encoder, we added spectral norm regularization on the weights of each layer during the training process. In the adversarial training process, we used the ADDA losses to perform domain adaptation with a learning rate of 10^{-6} for the target extractor and 10^{-4} for the discriminator.

Office-Caltech: We used the pre-trained DeCAF features (dec (2019)) for each domain along with the original train/test splits. The following architecture was used for the feature encoder and it was trained with a learning rate of 10^{-6} using the Adam optimizer and a batch size of 32.

```
Dense(512, activation='linear')
Dropout(0.7)
Dense(256, activation='linear')
Dropout(0.7)
Dense(10, activation=None)
```

In order to enforce the Lipschitz continuity, we added spectral norm regularization during the training process. In the adversarial training process, we used the ADDA losses to perform domain adaptation with a learning rate of 10^{-6} for the target extractor and 10^{-5} for the discriminator.

Mic2Mic: Similarly, we followed the same train/test splits as in the original dataset provided by the authors of Mathur et al. (2019). The spectrogram tensors were normalized between 0 and 1 during the training and test stages. The following model was trained for each source domain with a learning rate of 10^{-5} using the Adam optimizer and a batch size of 64.

```
Conv2D(filters = 64, kernel_size = (8,20), activation='relu')
MaxPooling2D(pool_size = (2,2)),
Conv2D(filters = 128, kernel_size = (4,10), activation='relu'),
MaxPooling2D(pool_size = (1,4)),
Flatten(),
Dense(256, activation='relu'),
Dense(31)
```

In order to enforce the Lipschitz continuity, we added spectral norm regularization during the training process. In the adversarial training process, we used the ADDA losses to perform domain adaptation with a learning rate of 10^{-3} for the target extractor and 10^{-2} for the discriminator.

F.0.3 COMPUTING INFRASTRUCTURE

Our system is implemented in Tensorflow 2.0 and uses NVIDIA V100 GPUs. For distributed training, we use the Message Passing Interface (MPI) to communicate discriminator gradients between the two virtual machines in a cloud infrastructure.