# FRuDA: Framework for Distributed Adversarial Domain Adaptation

Shaoduo Gan*, Akhil Mathur*, Anton Isopoussu, Fahim Kawsar, Nadia Berthouze, Nicholas D. Lane

**Abstract**—Breakthroughs in unsupervised domain adaptation (uDA) can help in adapting models from a label-rich source domain to unlabeled target domains. Despite these advancements, there is a lack of research on how uDA algorithms, particularly those based on adversarial learning, can work in distributed settings. In real-world applications, target domains are often distributed across thousands of devices, and existing adversarial uDA algorithms – which are centralized in nature – cannot be applied in these settings. To solve this important problem, we introduce FRuDA: an end-to-end framework for distributed adversarial uDA. Through a careful analysis of the uDA literature, we identify the design goals for a distributed uDA system and propose two novel algorithms to increase adaptation accuracy and training efficiency of adversarial uDA in distributed settings. Our evaluation of FRuDA with five image and speech datasets show that it can boost target domain accuracy by up to 50% and improve the training efficiency of adversarial uDA by at least $11\times$.

**Index Terms**—Distributed Domain Adaptation, Domain Shift, Adversarial Learning

## 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, *adversarial domain adaptation* has emerged as a prominent method for uDA, wherein the core idea is to learn domain-invariant feature representations from the data using adversarial learning [1], [2], [3], [4], [5], [6], [7], [8]. This paper focuses on exploring adversarial domain adaptation in a *distributed* setting. Prior works [4], [5], [6], [7] have taken an algorithmic viewpoint to adversarial domain adaptation and assumed that datasets from the source and target domains are available on the same machine and can be accessed freely during the adaptation process. While this assumption has made it easy to research uDA algorithms, it can be easily violated in real-world scenarios where the domain datasets reside on massively distributed devices and are not allowed to be shared for privacy reasons.

As an example, let us consider the task of developing personalized health monitoring solutions on mobile devices, such as the prediction of COVID-19 from cough sounds collected from a smartphone [9]. A company (*i.e., a source domain*) can collect a labeled dataset for the task and train a prediction model on it. Later, this source model needs to be deployed for smartphone users around the world (*i.e., target domains*) and will require adapting to each user's personal health condition and biomarkers. As collecting labels from the target domains is challenging in this setting, unsupervised domain adaptation could become a promising approach to adapt the source model and tailor it to each user's data.

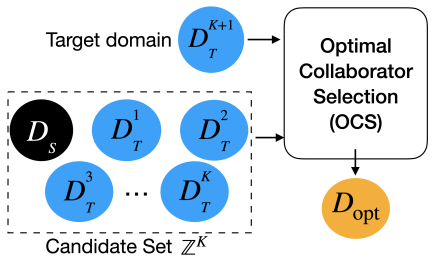
However, since the health data records from target users are distributed across thousands of devices and cannot be uploaded on a central server due to potential privacy reasons, we cannot use the centralized uDA techniques proposed in the literature. Performing adversarial domain adaptation in such distributed settings remains an under-explored problem and is the main focus of this paper. Clearly, if adversarial uDA can be extended to such distributed settings, it will further enhance the potential for real-world impact of these algorithms.

**Setup and Challenges.** We consider a distributed system where each domain dataset resides on a different node of the system. We assume that one domain dataset is labeled and represents the *source domain*. Other domain datasets are unlabeled and represent the *target domains* which need to undergo domain adaptation to learn a model tailored to their data distribution. New target domains can join the distributed system at any time. The nodes are able to communicate with each other over a network. Based on this novel but realistic setup, we highlight three key challenges in building a distributed adversarial uDA system:

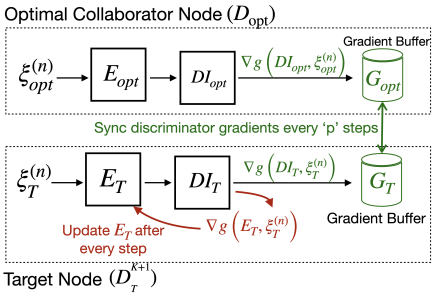
- The first challenge relates to *system design*: how do we design distributed adversarial neural network architectures that can perform uDA without exchanging any raw data between domains. Distributed training techniques have been studied extensively for supervised learning [10], [11], but their investigation for adversarial uDA remains under-explored.
- Next is the challenge of *efficiency*. In distributed machine learning, any communication between the nodes incurs a cost, both in terms of training time and data transfer expenses. As such, the design of a distributed adversarial uDA system should be as efficient as possible.
- Finally, a key challenge is to obtain the highest possible


- *Shaoduo Gan is with ETH Zurich and did this work while on an internship with Nokia Bell Labs, Cambridge, UK.*
- *Akhil Mathur and Fahim Kawsar are with Nokia Bell Labs, Cambridge, UK.*
- *Anton Isopoussu is with Invenia Labs, Cambridge, UK.*
- *Nadia Berthouze is with University College London, London, UK.*
- *Nicholas D. Lane is with University of Cambridge, Cambridge, UK.*

** denotes joint primary authors*




**(a)** Optimal Collaborator Selection (OCS)




**(b)** Distributed uDA using DILS

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

*accuracy* for each target domain after adaptation. Let us define a *collaborator* as the domain with which a target domain undergoes adaptation, e.g., in X→Y adaptation, X is the collaborator for the target domain Y. Seminal theoretical works [12] in domain adaptation have proven that the accuracy obtained for a given target domain is highly dependent on the characteristics of the collaborator with which the adaptation is performed. Hence, it becomes critical to select the *optimal collaborator* for each target domain in a distributed setting, to ensure that the target domain can achieve the highest accuracy through adaptation.

**Contributions.** In this paper, we propose an end-to-end learning framework, named **F**ramework for **R**ealistic uDA (FRuDA) which addresses all the above challenges in a unified manner, and makes adversarial uDA accurate, efficient, and privacy-preserving in a distributed setting. Figure 1 demonstrates the system overview of FRuDA. The source code of FRuDA can be found in Appendix A.1. At the core of FruDA are two novel contributions:

1) A distributed collaborator selection algorithm called Optimal Collaboration Selection (OCS) which finds the best adaptation *collaborator* for each unlabeled target domain in the system. OCS is built on a novel theoretical formulation, which selects the optimal collaborator based on the collaborator's own in-domain error and the Wasserstein distance between the collaborator and target domain. Our results show that OCS can lead to an increase in the target domain accuracy in distributed uDA systems by as much as 50% over various baselines.
2) A distributed training strategy called Discriminator-based Lazy Synchronization (DILS) which decomposes the adversarial learning architecture across distributed nodes and performs uDA by exchanging the gradients of the domain discriminator between nodes. DILS ensures that unlabeled target domains can learn a prediction model through adaptation from other domains in a privacy-preserving manner, without revealing or exchanging their raw data. DILS also allows for setting a trade-off between the *accuracy* and *efficiency* objectives of distributed uDA, using a tunable parameter.

The paper is structured as follows. In §2, we provide a brief primer on adversarial domain adaptation and further motivate the design requirements of a distributed uDA system. In §4, we present our end-to-end learning framework FruDA, describe the two core algorithms on which FruDA is built, and provide theoretical justifications for our design. In §5, we provide a comprehensive evaluation of FruDA on multiple vision and speech datasets. We compare FruDA against various baselines for selecting adaptation collaborators and observe that it significantly outperforms them on target domain accuracy, by as much as 50%. We also illustrate that FruDA can reduce the amount of data exchanged during training by at least 11× when compared to state-of-the-art baselines, without significantly compromising the adaptation accuracy. Finally, we show that FruDA can co-exist with various types of adversarial uDA algorithms proposed in the literature, thus making it a generalizable framework for supporting distributed adversarial uDA.

## 2 Preliminaries and Design Goals

### 2.1 Primer on Adversarial uDA

We first provide a brief primer on adversarial uDA with one source and one target domain. In the subsequent sections, we will explain how we extend adversarial uDA to distributed systems with multiple target domains.

Let $D_S$ be a source domain with labeled training set $\{X_S, Y_S\}$ and $D_T$ be a target domain with unlabeled training set $\{X_T\}$. We can train a feature extractor, $E_S$, and a classifier, $C_S$ for the source domain using supervised learning by optimizing a classification loss $\mathcal{R}_{cls}$ as follows:

$$\mathcal{R}_{cls} = \mathbb{E}_{(x_s,y_s)\sim(X_S,Y_S)} l_{cls}[C_S(E_S(x_s)), y_s)]$$

, where $l_{cls}(\cdot)$ is a loss function such as categorical cross-entropy.

The goal of adversarial 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 target classifier $C_T$. To learn $E_T$, two losses are optimized using adversarial learning, namely the Discriminator Loss $\mathcal{R}_{dis}$ and the Mapping Loss $\mathcal{R}_{map}$, as follows:

$$\mathcal{R}_{dis} = \mathbb{E}_{x_t\sim X_T, x_s\sim X_S} l_{dis}[DI(E_S(x_s)), DI(E_T(x_t))] \quad (1)$$

$$\mathcal{R}_{map} = \mathbb{E}_{x_t\sim X_T, x_s\sim X_S} l_{map}[DI(E_S(x_s)), DI(E_T(x_t))] \quad (2)$$

Here $DI$ represents a domain discriminator tasked with separating data from source and target domains.$l_{dis}(\cdot)$ and

$l_{map}(\cdot)$ are the adversarial loss functions, which have been studied by several previous works. For example, DANN [4] uses a cross-entropy loss to compute $l_{dis}$, where the labels indicate the domain of the data. For the Mapping loss, they simply compute $l_{map}(\cdot) = -l_{dis}(\cdot)$ using a Gradient Reversal Layer (GRL). Instead, ADDA [5] computes the mapping loss using the label inversion trick. Other works such as [7] use Wasserstein distance as the metric to compute $l_{dis}$.

The important takeaway here is that even though different algorithms employ different loss functions, the general training paradigm of adversarial uDA remains similar as shown in Eq. 1 and 2. This simple yet important insight means that it is possible to design a generalized domain adaptation framework for a distributed setting that can work for many uDA algorithms. This intuition will be confirmed in §5 where we show that our proposed framework can work with four types of uDA optimization objectives.

### 2.2 Design Requirements for a Distributed Domain Adaptation System

In this section, we describe two key design requirements for a distributed domain adaptation system, which will serve as a guide for the algorithms and training strategies proposed in the paper.

**Training Efficiency.** Efficiency is a key design metric for any distributed system. Compared with the powerful computation capabilities of modern hardware, communication tends to be the main bottleneck in distributed training. Especially, in distributed domain adaptation, a domain dataset could possibly reside on a smartphone, a laptop or a wearable device, whose communication capabilities are far inferior than data center machines. In such massively distributed scenarios, an optimal communication strategy is crucial for system's efficiency and user experience.

Recall the model components $E_S$, $E_T$, $C_S$, and $DI$ mentioned in §2.1 that are involved in adversarial uDA. In a non-distributed setting, these components reside on the same machine and passing data from one to another has negligible cost. However, in a distributed setup, each domain has to keep a copy of these components on their machine and any exchange of information between them takes time and incurs a communication cost.

Since raw data is private and cannot be be exchanged in our problem setup, we exchange the gradients of these model components to facilitate distributed training. Hence, the key challenge is: *what is the most communication-efficient strategy to exchange model gradients across domains?* More specifically, this question can be decomposed into three sub-questions: 1) *how many domains should be involved in the communication?* 2) *which model components are necessary to be communicated?* and 3) *how frequently do they need to be communicated?*

To answer these questions, we analyze various types of adversarial uDA approaches in the literature. Based on the number of domains involved in training, adversarial uDA algorithms can be categorized into pairwise adaptation [4], [5], [7], [8], multi-source adaptation [13], [14], and multi-target adaptation [15]. Clearly, as the latter two approaches require either multiple source or target domains, they will incur higher communication costs and are less efficient than pairwise adaptation. Another way to classify adversarial uDA algorithms is tied [4] or untied [5] algorithms, depending on whether the feature extractors of source and target domains share weights (i.e., tied) or not (i.e., untied). Untied algorithms are communication-efficient because the feature extractors could be trained separately and their gradients do not need to be exchanged during training. Only the gradients of domain discriminator $DI$ need to be shared across nodes. For these reasons, the design of our proposed framework will be based on the **pairwise and untied** adversarial uDA algorithms.

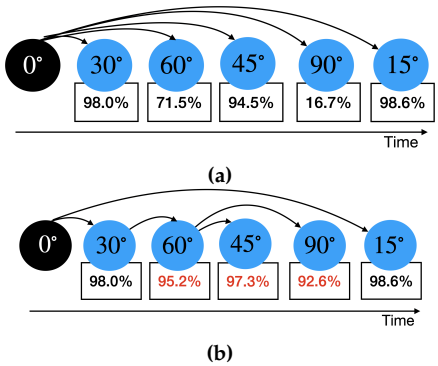



**Fig. 2:** 0° is the labeled source domain while the domains in blue are unlabeled target domains appearing sequentially in the system. The numbers in rectangle denote the post-adaptation accuracy for a domain. (a) Static Design: Labeled Source acts the collaborator for each target domain. (b) Flexible 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).

After narrowing down the scope of uDA algorithms from an efficiency perspective, we need to decide how frequently should we communicate gradients between nodes. Naively, we can exchange gradients of the discriminator between nodes after training *each* batch of data, which is common in data-parallel distributed training. However, this approach comes at the expense of a significant communication cost. Instead, we will propose a **lazy synchronization** strategy that exchanges the gradients every $p$ steps, and further reduces the communication costs of adversarial uDA with negligible impact on adaptation accuracy. Our proposed training strategy is presented in detail in §4.2.

FADA [13] is a recently proposed technique for performing adversarial uDA in a federated learning setting. Based on the above analysis, FADA can be characterized as a multi-source uDA approach which exchanges the gradients of feature extractors after every training step. Although FADA originally solves a different problem than ours and assumes that the system has multiple labeled domains, we implement a special case of FADA in §5 to provide a fair comparison of its efficiency with our technique.

**Target Domain Accuracy.** Obtaining a high accuracy in the target domain is arguably the most important metric of success for a uDA solution. Prior works in learning theory [12] have shown that the classification error obtained in a target domain depends on the characteristics of the collaborator with which the adaptation is performed. This means that if we select the right (or optimal) collaborator for each target

domain, it could significantly reduce the target domain error and boost the target domain accuracy.

To better explain this aspect, 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 adversarial uDA. The naïve approach adopted by many existing uDA approaches is to always choose the labeled source domain as the collaborator for a target domain. Figure 2a shows the performance of such existing approaches if each target domain always chooses the labeled source 0° as its collaborator. While 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 2b 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%, which is almost 75% higher than what could be achieved by adapting from 0°, as shown in Figure 2.

In summary, this example provides a clear insight that we can significantly increase the target domain accuracy if we adapt from the right (or the optimal) collaborator. **How do we choose such an optimal collaborator?** In some aspects, this problem is similar to the source selection problem [16] where a metric such as A-distance is employed to compute a distance measure between domains, and the domain with the least distance from the target is chosen as its collaborator. However, none of the prior works are designed for a distributed setup and they require access to raw data samples from both domains to compute the distance metric for source selection. Instead, we propose a fully-distributed approach which selects an optimal collaborator based on the collaborator's in-domain error and the Wasserstein distance between the collaborator and target.

## 3 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 [17], Gromov-Wasserstein Discrepancy [18], A-distance [19], and subspace mapping [20]. However, our results in §5 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 relies on Wasserstein distance which could be computed in a distributed setting without exchanging raw data between domains.

There are also works on selecting or generating intermediate domains for uDA. [16] 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. [21] and [22] 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, [23], [24] 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 can appear in random order, 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 [25], [26], decentralized training [10], [11], and asynchronous training [27]. Similarly, with the goal of preserving data privacy, *Federated Learning* proposes sharing model parameters between distributed nodes instead of the raw data [28], [29]. However, these distributed and federated training techniques are primarily designed for supervised learning and do not *extend directly to adversarial uDA architectures*. A notable exception is FADA by [13] which extends uDA to federated learning. As we discussed earlier, FADA is designed for a multi-source setting and assumes that all source domains are labeled. Moreover, it exchanges features and gradients of the *feature extractor* between nodes after every step. Instead, DILS operates by doing pairwise adaptation through lazy synchronization of *discriminator gradients* between nodes, which brings significant efficiency gains over FADA.

**Source-free uDA.** [30] and [31] 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 in real-world settings.

## 4 FRuDA: Framework for Distributed Adversarial Domain Adaptation

We first present our two algorithmic contributions on Optimal Collaborator Selection (named OCS) and Discriminator-based Lazy Synchronization (named DILS). Later in §4.3, we explain how these two algorithms work in conjunction with each other in an end-to-end framework called FRuDA, and enable adversarial uDA algorithms to work in a distributed machine learning system.

### 4.1 Optimal Collaborator Selection (OCS)

Recall from §2.2 that the goal of collaborator selection is to find the optimal collaborator for each target domain. We first formulate the technical problem and then present our solution.

#### 4.1.1 Problem Formulation.

In our problem setting shown in Figure 1a, there is one labeled source domain $D_S$ and multiple unlabeled target domains. Let $\{D_T^j | j = 1, \ldots, K\}$ be the $K$ unlabeled target

| Property \ Method | ADDA [5], DANN [4], CADA [8] | Multi-Source uDA [14], [32], [33] | Federated uDA [13] | Tan et al. [16] | FRuDA (Ours) |
|---|---|---|---|---|---|
| Adversarial Domain Adaptation | ✓ | ✓ | ✓ | | ✓ |
| Collaborator Selection before Adaptation | | | | ✓ | ✓ |
| Supports Distributed uDA | | | ✓ | | ✓ |
| Communication Efficient Distributed uDA | | | | | ✓ |
| Framework to support multiple uDA algorithms | | | | | ✓ |
| Number of labeled source domains | 1 | Multiple | Multiple | 1 | 1 |

**TABLE 1:** 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.

domains for which we want to learn a prediction model using uDA. We assume that target domains join the distributed system sequentially.

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 $\tau$. When the 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 can only adapt from $D_S$ and learn a model $E_T^1$. Having learned the model, $D_T^1$ is now added to the candidate set $\mathbb{Z}_\tau$ (along with its unlabeled data) and can act as a collaborator for future domains.

In general, at step $\tau = K$, $\mathbb{Z}_K = \{D_S\} \cup \left\{D_T^j | j = 1, \ldots, K\right\}$ as shown in Figure 1a. 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:

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

where $\mathbb{Z}_K^i$ is the $i^{th}$ candidate domain in $\mathbb{Z}_K$ and $\Phi$ is a metric that quantifies the error of collaboration between $\mathbb{Z}_K^i$ and $D_T^{K+1}$. In other words, OCS aims to select a candidate domain from $\mathbb{Z}_K$ which has the least error of collaboration with the target domain.

#### 4.1.2 Solution

Our key idea is quite intuitive: the optimal collaborator $D_{\text{opt}}$ 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 $\theta$-Lipschitz if it satisfies the inequality $\|f(x) - f(y)\| \leq \theta\|x - y\|$ for some $\theta \in \mathbb{R}_+$. The smallest such $\theta$ 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 $\theta_{\text{CE}}$ denote the Lipschitz constant of the cross-entropy loss function in $D_1$. For any two $\theta$-Lipschitz hypotheses $h, h'$, we can derive the following error bound for the cross-entropy (CE) error in $D_2$:

$$\varepsilon_{\text{CE,D}_2}(h, h') \leq \theta_{\text{CE}} \left(\varepsilon_{\text{L}_1,\text{D}_1}(h, h') + 2\theta W_1(D_1, D_2)\right) \quad (3)$$

where $W_1(D_1, D_2)$ denote the first Wasserstein distance between the domains $D_1$ and $D_2$, and $\varepsilon_{\text{L}_1,\text{D}_1}$ denotes the $L_1$ error in $D_1$. A full proof is provided 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 cross-entropy (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, this bound is more useful than the one proposed by [7] which estimates the $L_1$ error in the target domain. Secondly, Theorem 1 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 our distributed problem setup and differentiates it from other distance metrics such as A-distance or Maximum Mean Discrepancy (MMD) which cannot be computed in a distributed 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:

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

We can tighten this bound to get a more reliable estimate of the target CE error. This is achieved by reducing the Lipschitz constant ($\theta$) 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 as implemented in [34]. Our empirical results show that the upper bound in Eq. 4 is a good approximation of the target domain error for the purpose of collaborator selection. In our implementation, we make a simplifying assumption about the availability of a small test set on which the collaborator error can be computed. In future work, we will study more sophisticated error propagation strategies across target domains.

Now that we have a way to estimate the target CE error, we 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 $\theta^k_{\text{CE}}$ and $\theta^k$. 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 $D_{\text{opt}}$:

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

#### 4.1.3 Computing Wasserstein Distance across Distributed Datasets

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 [7], the $W_1$ distance can be computed as:

$$W_1(X_C, X_T) = \frac{1}{n_C} \sum_{x_s \sim \mathcal{X}_C} DI(E_C(x_c)) - \frac{1}{n_T} \sum_{x_t \sim \mathcal{X}_T} DI(E_T(x_t)) \tag{6}$$

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 6 has a similar structure to the optimization objectives for ADDA and other uDA algorithms discussed in the paper. Hence, we can use the same principle as DILS (described in 4.2) 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:

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

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

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

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

$$\nabla g(DI_{\text{agg}}, x_c, x_t) = \nabla g(DI_C, x_c) - \nabla g(DI_T, x_t) \tag{7}$$

**Algorithm 1:** DILS

**Result:** $E_T$

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

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

- Sample a batch of data on both nodes, $\xi^{(n)}_{\text{opt}}$ and $\xi^{(n)}_T$. Then feed $\xi^{(n)}_{\text{opt}}$ and $\xi^{(n)}_T$ into the respective Extractor-Discriminator model separately on both nodes;
- Based on different loss functions, calculate the gradients locally. On collaborator node, calculate $\nabla g(DI_{\text{opt}}, \xi^{(n)}_{\text{opt}})$; On target node, calculate $\nabla g(E_T, \xi^{(n)}_T)$ and $\nabla g(DI_T, \xi^{(n)}_T)$;
- Add $\nabla g(DI_{\text{opt}}, \xi^{(n)}_{\text{opt}})$ to gradient buffer $G_{\text{opt}}$, add $\nabla g(DI_T, \xi^{(n)}_T)$ to target gradients buffer $G_T$ ;
- **if** *isTargetNode* **then**
  - Apply $\nabla g(E_T, \xi^{(n)}_T)$ 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 $G_{\text{opt}}$ and $G_T$ ;
- Apply $g_{\text{sync}}$ to $DI_{\text{opt}}$ and $DI_T$ separately;

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:

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

### 4.2 Distributed uDA using DIscriminator-based Lazy Synchronization (DILS)

#### 4.2.1 Algorithm description

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

As shown in Figure 1b, we split the adversarial architecture across the distributed nodes. The feature encoders of the collaborator ($E_{\text{opt}}$) and target ($E_T$) reside on their respective nodes, while the discriminator $DI$ is split into two components $DI_{\text{opt}}$ and $DI_T$. As we highlighted in §2, our framework is based on the untied adversarial uDA algorithms and exchanges information between distributed nodes using gradients of the discriminators. Our training process works in two steps: 1) update the target feature extractor $E_T$ to optimize the mapping loss $\mathcal{R}_{map}$, 2) update the discriminators $DI_T$ and $DI_{opt}$ to optimize the discriminator loss $\mathcal{R}_{dis}$. Note that $E_{opt}$ is assumed to be pre-trained in the past and is not updated.

We present our DIscriminator-based Lazy Synchronization (DILS) strategy in Algorithm 1. Specifically, at each training step $n$, both nodes feed their domain data $\xi^n_{\text{opt}}$

and $\xi_T^n$ into their extractors and discriminators, and compute the gradients of $E_T$, $DI_T$ and $DI_{opt}$, i.e., $\nabla g(E_T, \xi_T^n)$, $\nabla g(DI_{\text{opt}}, \xi_{\text{opt}}^n)$ and $\nabla g(DI_T, \xi_T^n)$ respectively. Since $DI_T$ and $DI_{opt}$ are supposed to be shared between nodes, how to synchronize these two components is crucial to adaptation accuracy and communication cost.

If we exchange gradients of $DI_T$ and $DI_{opt}$ after every training step, we can keep the discriminators strictly synchronized and ensure that distributed training converges to the same loss as the non-distributed case. However, this approach has a major downside from an efficiency perspective, as exchanging gradients in every step incurs significant communication costs and increases the overall uDA training time. To boost the training efficiency, we propose a *Lazy Synchronization* approach, wherein instead of every step, the discriminators are synchronized every $p$ training steps, thereby reducing the total communication amount 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*.

In effect, DILS uses synced stale gradients ($\nabla g_{\text{sync}}$) from the latest sync-up step to update the discriminators (line 11), instead of using their local gradients. There are two reasons behind this design choice. Firstly, $DI_T$ and $DI_{opt}$ are two replicas of the same component and are intended to be consistent. Updating them with different local gradients can cause them to diverge. Secondly, the local gradients of $DI_T$ or $DI_{opt}$ are derived from data of only one domain, and are likely to be biased to that domain. Our design choice of applying the latest synced gradients ($\nabla g_{\text{sync}}$) circumvents these limitations of local gradients and guarantees the convergence of adversarial uDA algorithms. Our experiment results also confirm that this approach significantly reduces the uDA communication cost without degrading the target accuracy.

#### 4.2.2 Convergence Analysis

The network structure of *Lazy Synchronization* can be taken as a typical Generative Adversarial Net (GAN) [35] where the generative model is target encoder $E_T$ and the discriminative model is discriminator $DI$ ($DI_S$ and $DI_T$). $E_S$ and $E_T$ can separately define two probability distributions $E_S(x_s), x_s \sim \mathcal{X}_S$ 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 [35], we know that:

**Proposition 1.** 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 updating strategy of DILS, convergence behaviours of $DI_S$ and $DI_T$ are close enough to the non-distributed case, then the $E_T$ should be able to converge as well. Note that although $DI_S$ and $DI_T$ are lazily synced, their weights are always identical because they are equally initialized and apply same gradients (latest synced one) for every training step. Therefore, we can take it as one discriminator $DI_{lazy}$ in the convergence analysis. Then we have the following theorem.

**Theorem 2.** In DILS, given a fixed target encoder, under certain assumptions, we have the following convergence rate for the discriminators $DI_S$ and $DI_T$ (full proof is in Appendix B):

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}[\frac{f(x_0)-f(x_{t+1})}{\mu T}+ (2p-1)L\mu\sigma^2 + \frac{\mu L\sigma^2}{2} + \frac{\mu^3 L^3\sigma^2(2p-1)}{2}] \quad (8)$$

Where $p$ is the sync-up step, $\mu$ 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.

#### 4.2.3 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 [36], therefore it is critical to examine: *can the discriminator gradients also indirectly leak training data of a domain*?

We study the performance of DILS under a state-of-the-art gradient leakage attack proposed by [37]. They 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 $\nabla W$ be the gradients with respect to a private input pair $(\mathbf{x}, y)$. During distributed training, $\nabla W$ 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:

$$\begin{aligned}\mathbf{x}'^*, \mathbf{y}'^* &= \arg\min_{\mathbf{x}', \mathbf{y}'} \|\nabla W' - \nabla W\|^2 \\ &= \arg\min_{\mathbf{x}', \mathbf{y}'} \left\| \frac{\partial \ell\left(F\left(\mathbf{x}', W\right), \mathbf{y}'\right)}{\partial W} - \nabla W \right\|^2 \end{aligned} \quad (9)$$

**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 ($\nabla W$) 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 extractor and the discriminator) during the adversarial training process. As shown in Algorithm 1, the target feature extractor 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 extractor 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 ($\nabla W$) of all the layers between the input $x$ and output $y$ in the source domain. This includes gradients of $E_S$ and $DI_S$. 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. 9 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.

### 4.3 The End-to-End View of FRuDA

We now discuss how OCS and DILS work together to address the challenges of distributed adversarial uDA introduced in §1. As shown in Figure 1a, a new target domain $D_T^{K+1}$ first performs OCS with all candidate domains in $\mathbb{Z}^K$ to find its optimal collaborator $D_{\text{opt}}$. 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 1b, $D_T^{K+1}$ and $D_{\text{opt}}$ use DILS to engage in distributed uDA. This step ensures that private raw data is not exposed during adaptation and yet the target domain is able to learn a model for its distribution in an **efficient** manner. 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.

## 5 Evaluation

### 5.1 Setup

**Datasets.** We evaluate FRuDA on five image and speech datasets:

- **Rotated MNIST**: A variant of MNIST with digits rotated clockwise by different degrees. Each rotation is considered a separate domain.
- **Digits**: Five domains of digits: MNIST (M), USPS (U), SVHN (S), MNIST-M (MM) and SynNumbers (SYN), each consisting of digit classes ranging from 0-9.
- **Office-Caltech**: Images of 10 classes from four domains: Amazon (A), DSLR (D), Webcam (W), and Caltech-256 (C).
- **Mic2Mic**: A speech keyword detection dataset recorded with four microphones: Matrix Creator (C), Matrix Voice (V), ReSpeaker (R) and USB (U). Each microphone represents a domain.
- **DomainNet**: A new challenge dataset from which we use four labeled image domains containing 345 classes each: Real (R), QuickDraw (Q), Infograph (I), and Sketch (S).

**Evaluation Metrics and Implementation.** FRuDA is designed for a distributed system where each node represents a domain. Initially, one labeled source domain $\mathbf{D_S}$ is given in the system and thereafter unlabeled target domains appear sequentially in a random order. The overall accuracy of the system is measured by the mean adaptation accuracy obtained over all target domains. The communication cost is measured by the amount of traffic due to gradient exchange between nodes. For our experiments, we use a Nvidia V100 GPU to represent a distributed node. All nodes are connected via TCP network. We use Message Passing Interface (MPI) as the communication primitive between nodes. The system is implemented with TensorFlow 2.0. We use the implementation provided by [34] for calculating Lipschitz constants of a neural network. More details on network architectures, hyper-parameters and downloading the source code are provided in the Appendix.

### 5.2 Adaptation Accuracy of FRuDA

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 ($p = 4$) to perform distributed adversarial uDA between the target domain and the collaborator, and compute the test accuracy $\text{Acc}_T^i$ in the target domain. We report the mean adaptation accuracy obtained over all target domains, i.e., $\frac{1}{K}\sum_{i=1}^{K} \text{Acc}_T^i$.

For each dataset, we choose two random orderings of source and target domains as shown in Table 3. The optimization objectives of ADDA [5] as discussed in §2.1 are used for adaptation in this experiment.

**Collaborator Selection Baselines.** We use four baselines for collaborator selection: (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 [38] from the target; (iv) *Multi-Collaborator* is based on MDAN [14], where all available candidate domains contribute to the adaptation in a weighted-average way. While this baseline obviously is less efficient than pairwise adaptation, we are interested in comparing its accuracy with our system. Note that MDAN was originally developed assuming that all candidate domains are *labeled*, which is not the case in our setting. Hence, for a fair comparison, we modify MDAN by only optimizing its adversarial loss during adaptation.

**Results.** Table 2 reports the mean accuracy obtained across all target domains for two orderings in each dataset. We can observe that collaborator selection techniques have a crucial impact on the adaptation accuracy, and FRuDA outperforms the baseline techniques in almost all cases. Below we describe our key findings:

Firstly, we observe that *Random Collaborator* and *Multi-Collaborator* are in general the worst among all the methods. We surmise that this is due to the fact that these methods do not consider the distances between domains, and end up with collaborator domains that are too different from the target domain. Compared with *Proxy A-distance (PAD)*,

**TABLE 2:** Mean adaptation accuracy obtained over all target domains in a given order. The sync-up step size $p = 4$ is used for this experiment.

| | RMNIST | | Digits | | Office-Caltech | | Mic2Mic | | DomainNet | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $Order_1$ | $Order_2$ | $Order_1$ | $Order_2$ | $Order_1$ | $Order_2$ | $Order_1$ | $Order_2$ | $Order_1$ | $Order_2$ |
| No Adaptation | 34.65 | 35.54 | 59.59 | 72.09 | 66.40 | 85.25 | 76.45 | 75.83 | 26.12 | 28.32 |
| 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 | 20.66±3.10 | 28.15±2.89 |
| 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 | 33.51±0.09 | **36.44±0.14** |
| 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 | 18.41±1.04 | 25.46±0.61 |
| 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 | 35.03±0.21 | 36.0±0.49 |
| 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** | **37.10±0.28** | 35.46±0.31 |

FRuDA has better performance because we jointly consider the in-domain error and the Wasserstein distance between domains during collaborator selection.

Next, for *Labeled Source* (LS), we observe that its relative performance against FRuDA depends on the characteristics of the tasks, and the number and nature of domains. 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 leads to higher accuracy gains over LS in this dataset. For Office-Caltech (Order 2), the performance of FRuDA is the same as the Labeled Source baseline because here the the labeled source domain **W** turned out to be the optimal collaborator for all target domains. Overall, out of the 10 different orderings across 5 datasets, we found that LS outperforms FRuDA in only one setting (DomainNet Order 2). For this order, we found that the labeled source: Sketch (S) is the best adaptation collaborator for all subsequent domains, hence the LS performs the best. FRuDA makes one error in collaborator selection here: for target domain = Infograph (I), it picks Quickdraw (Q) as the collaborator instead of picking the labeled source Sketch (S), which leads to a 1% mean accuracy drop.

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.

**Generalization to other uDA optimization objectives.** The results presented in Table 2 used the optimization objectives of ADDA [5]. However, as we discussed earlier, FRuDA is intended to be a general framework for distributed uDA and not limited to one specific uDA algorithm. We now evaluate FRuDA with three other uDA loss formulations (i) [4], which uses a Gradient Reversal Layer (GRL) to compute the mapping loss, (ii) *WassDA* [7], which uses Wasserstein Distance as a loss metric for the domain discriminator and (iii) CADA [8] which operates by enforcing consensus between source and target features. In Table 4, we observe that different uDA techniques yield different target accuracies after adaptation, depending on their optimization objective. Regardless, FRuDA can work in conjunction with all of them to improve the target accuracy over the *Labeled Source* baseline because our proposed framework is designed to be agnostic to the learning algorithm.

**TABLE 3:** 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.

| Dataset | Order 1 | Order 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 |
| Digits | **MM**, Syn, M, U, S | **Syn**, M, MM, U, S |
| Office-Caltech | **D**, W, C, A | **W**, C, D, A |
| Mic2Mic | **V**, U, C, R | **U**, C, V, R |
| DomainNet | **S**, R, I, Q | **S**, Q, I, R |

**TABLE 4:** Mean target accuracy for four uDA methods. FRuDA can be used in conjunction with various uDA methods, and improves mean 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 |
| Proxy A-distance | 93.51 | 94.91 | 89.04 | 78.70 | 70.09 | 67.14 | 72.66 | 67.0 |
| FRuDA | **97.08** | **97.35** | **91.15** | **83.37** | **73.01** | **69.80** | **75.36** | **70.19** |

**Takeaways.** This section highlighted the accuracy gains achieved by FRuDA in a distributed uDA setting by selecting the optimal collaborator for each domain. We also showed that FRuDA can act as a general framework for distributed adversarial uDA by incorporating the optimization objectives of various uDA algorithms.

### 5.3 Communication Efficiency of FRuDA

To evaluate the communication efficiency of FRuDA, we use the amount of data communicated during pairwise adversarial training as a metric. Indeed, if a method exchanges less amount of data between two nodes during distributed training, it will be communication-efficient. Note that in addition to the adversarial training costs, we also incur communication costs during collaborator selection. However, those costs are negligible as compared to adversarial uDA and are not included in our analysis.

Since no previous work has studied the communication costs of adversarial uDA, we choose the most related work FADA [13] as our baseline. As discussed in §4.2, FADA was originally designed for multiple sources in a federated learning setup, which is not the case in our 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 (referred to as 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.

**Results.** In Figure 3, we plot the test accuracy with the amount of data transmission during pairwise training between two nodes. For DILS, we have two curves with

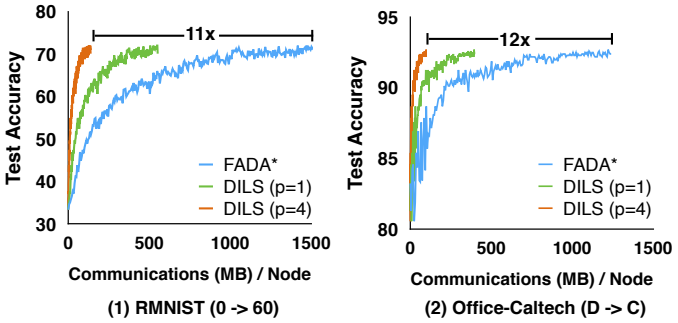



**Fig. 3:** Test accuracy vs. communication amount per node. DILS (p=4) provides up to 12× reduction in amount of data exchanged during training.

different value of $p$. The results show that with the lazy synchronization strategy ($p = 4$), DILS can achieve the same test accuracy as FADA* or strict synchronized case ($p = 1$), but consume much less amount of communication data. On RMNIST and Office dataset, we save up to 12× traffic compared with FADA*. We also present the similar results for Digits, Mic2Mic and DomainNet in Table 5.

There are two key reasons why DILS can be much more efficient than FADA*: i) FADA (and FADA*) exchange gradients of the feature extractor during the training, while DILS only exchanges gradients of the discriminator. As the discriminator has significantly fewer parameters than the feature extractor, DILS is inherently more communication-friendly. ii) In DILS ($p = 4$), the synchronization between two nodes happens every $p$ steps. Instead, FADA* requires data exchange after every single step. Therefore, DILS requires much less number of network synchronization. In sum, DILS is able to reduce the communication overhead over FADA* by exchanging lesser amount of data at a lesser frequency.

**Effect of sync-up step.** In DILS, the sync-up step size $p$ controls the frequency of gradient exchange between nodes. In Figure 4, 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 RMNIST and 1.7% for Digits. This accuracy loss is offset by reduction in data communication and gains in training speed. When $p$ is high, DILS exchanges gradients less frequently, which leads to less communication overhead but slightly worse accuracy. When $p$ is low, the training accuracy is improved at the expense of higher communication cost.

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

### 5.4 Analysis and Discussion

We now analyze some additional properties of FRuDA, and provide our thoughts on FRuDA's future research directions.

**TABLE 5:** Communication Amount per node till convergence.

| Dataset | Digits | Mic2Mic | DomainNet |
|---|---|---|---|
| FADA* | 7.9 GB | 25 GB | 86 GB |
| $DILS(P = 1)$ | 802 MB | 2 GB | 4 GB |
| $DILS(P = 4)$ | 201 MB | 500 MB | 1 GB |

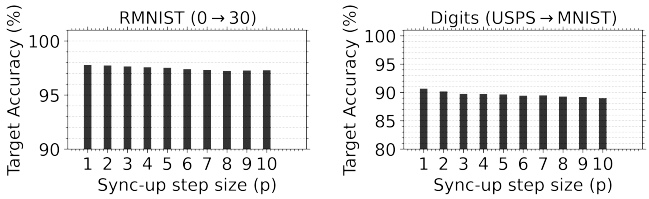



**Fig. 4:** Effect of varying the sync-up step $p$ from 1 to 10 on target domain accuracy for two adaptation tasks

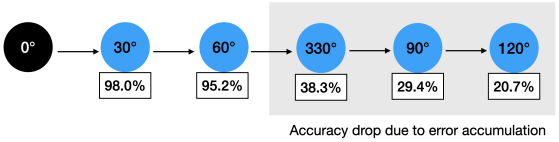
**Error accumulation and negative transfer in sequential adaptation.** FRuDA can be interpreted as a form of sequential adaptation, because new target domains can choose a collaborator from previously adapted target domains using the OCS algorithm. In sequential adaptation, error accumulation and negative transfer for downstream target domains is a possibility, especially if an unrelated domain appears in the sequence. We analyze OCS from this lens in the following text.

Figure 5a shows a sequence of one source domain (0°) and 5 target domains (30°, 60°, 330°, 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°→330° results in high error and poor adaptation accuracy for 330°. This behavior is due to 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.

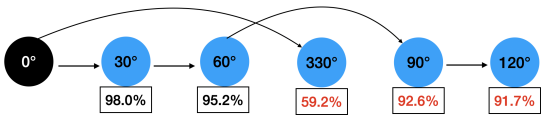
The ability of OCS to flexibly choose a collaborator for each target domain inherently counters this problem. Firstly, Figure 5b shows that we can choose a better collaborator for 330° using OCS and obtain almost 20% higher adaptation accuracy than in Figure 5a. More importantly, the subsequent target domains are no longer reliant on 330° 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.

**Future Work**. We showed that FRuDA can work with multiple uDA algorithms that follow the adversarial training paradigm introduced in §2.1. However, other uDA techniques such as those based on generative algorithms (e.g., [6]) and those with no adversarial learning component (e.g., [39]) were out of scope of this paper. Other future works in-




**(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 shows accuracy gains (in red) over the baseline.

**Fig. 5:** OCS prevents negative transfer and error accumulation in sequential adaptation caused by the presence of unrelated domains.

clude extending FRuDA to scenarios where the label spaces of source and target domains do not overlap [40] or when there is label shift between them [41].

FRuDA, in its current form, is designed as a sequential adaptation algorithm, that is, it only deals with one target domain at a time. If multiple target domains join together as a batch, FRuDA will still process them one by one, and the processing order of the domain could impact their training accuracy. In future work, we will extend FRuDA to batch-based processing of target domains. In particular, it will be interesting to explore that given a pool of target domains, how do we sequentially feed them to FRuDA such that it maximizes target domain accuracies.

## 6 Conclusion

In real-world ML applications, datasets are often distributed across thousands of users and devices. Despite the rapid advancements in adversarial uDA research, their extension to distributed settings has surprisingly remained under-explored. We introduced FRuDA, an end-to-end framework for distributed adversarial uDA which brings a novel and complementary perspective to uDA research. Through a careful analysis of the literature, we identified the key design requirements for a distributed uDA system and proposed two novel algorithms to increase adaptation accuracy and training efficiency in distributed uDA settings. A comprehensive evaluation with multiple image and speech datasets show that FRuDA can increase target accuracy over baselines by as much as 50% and improve the training efficiency of adversarial uDA by up to $11\times$. Overall, this paper contributes to both domain adaptation and distributed learning literature, by showing for the first time, how domain adaptation and adversarial training can work in a distributed setting.

## References

[1] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," *arXiv preprint arXiv:1502.02791*, 2015.

[2] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2208–2217.

[3] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Conditional adversarial domain adaptation," in *Advances in Neural Information Processing Systems*, 2018, pp. 1640–1650.

[4] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[5] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7167–7176.

[6] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "Cycada: Cycle-consistent adversarial domain adaptation," in *International Conference on Machine Learning*, 2018, pp. 1994–2003.

[7] J. Shen, Y. Qu, W. Zhang, and Y. Yu, "Wasserstein distance guided representation learning for domain adaptation," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] H. Zou, Y. Zhou, J. Yang, H. Liu, H. P. Das, and C. J. Spanos, "Consensus adversarial domain adaptation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5997–6004.

[9] C. Brown, J. Chauhan, A. Grammenos, J. Han, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta, and C. Mascolo, "Exploring automatic diagnosis of covid-19 from crowdsourced respiratory sound data," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3474–3484.

[10] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[11] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D2: Decentralized training over decentralized data," *arXiv preprint arXiv:1803.07068*, 2018.

[12] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, no. 1-2, 2010.

[13] X. Peng, Z. Huang, Y. Zhu, and K. Saenko, "Federated adversarial domain adaptation," *ICLR*, 2020.

[14] H. Zhao, S. Zhang, G. Wu, J. M. Moura, J. P. Costeira, and G. J. Gordon, "Adversarial multiple source domain adaptation," in *Advances in Neural Information Processing Systems*, 2018, pp. 8559–8570.

[15] M. Ragab, Z. Chen, M. Wu, H. Li, C.-K. Kwoh, R. Yan, and X. Li, "Adversarial multiple-target domain adaptation for fault classification," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–11, 2020.

[16] B. Tan, Y. Song, E. Zhong, and Q. Yang, "Transitive transfer learning," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1155–1164.

[17] J. Wang, Y. Chen, W. Feng, H. Yu, M. Huang, and Q. Yang, "Transfer learning with dynamic distribution adaptation," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 1, pp. 1–25, 2020.

[18] Y. Yan, W. Li, H. Wu, H. Min, M. Tan, and Q. Wu, "Semi-supervised optimal transport for heterogeneous domain adaptation."

[19] J. Wang, V. W. Zheng, Y. Chen, and M. Huang, "Deep transfer learning for cross-domain activity recognition," in *proceedings of the 3rd International Conference on Crowd Science and Engineering*, 2018, pp. 1–8.

[20] B. Gong, Y. Shi, F. Sha, and K. Grauman, "Geodesic flow kernel for unsupervised domain adaptation," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 2066–2073.

[21] R. Gong, W. Li, Y. Chen, and L. V. Gool, "Dlow: Domain flow for adaptation and generalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2477–2486.

[22] J. Choi, Y. Choi, J. Kim, J.-Y. Chang, I. Kwon, Y. Gwon, and S. Min, "Visual domain adaptation by consensus-based transfer to intermediate domain." in *AAAI*, 2020, pp. 10 655–10 662.

[23] M. Wulfmeier, A. Bewley, and I. Posner, "Incremental adversarial domain adaptation for continually changing environments," in

*2018 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–9.

[24] A. Bobu, E. Tzeng, J. Hoffman, and T. Darrell, "Adapting to continuously shifting domains," 2018.

[25] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[26] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *arXiv preprint arXiv:1802.05799*, 2018.

[27] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," *arXiv preprint arXiv:1710.06952*, 2017.

[28] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[29] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, p. 12, 2019.

[30] J. N. Kundu, N. Venkat, A. Revanur, R. V. Babu *et al.*, "Towards inheritable models for open-set domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 376–12 385.

[31] J. Liang, D. Hu, and J. Feng, "Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation," *arXiv preprint arXiv:2002.08546*, 2020.

[32] K. Bascol, R. Emonet, and E. Fromont, "Improving domain adaptation by source selection," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 3043–3047.

[33] X. Cui and D. Bollegala, "Multi-source attention for unsupervised domain adaptation," *arXiv preprint arXiv:2004.06608*, 2020.

[34] H. Gouk, E. Frank, B. Pfahringer, and M. Cree, "Regularisation of neural networks by enforcing lipschitz continuity," *arXiv preprint arXiv:1804.04368*, 2018.

[35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[36] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 691–706.

[37] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 747–14 756.

[38] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *Advances in neural information processing systems*, 2007.

[39] B. Sun, J. Feng, and K. Saenko, "Correlation alignment for unsupervised domain adaptation," in *Domain Adaptation in Computer Vision Applications*. Springer, 2017, pp. 153–171.

[40] K. You, M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Universal domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2720–2729.

[41] Y. Wu, E. Winston, D. Kaushik, and Z. Lipton, "Domain adaptation with asymmetrically-relaxed distribution alignment," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6872–6881.

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

[43] A. Mathur, A. Isopoussu, F. Kawsar, N. Berthouze, and N. 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*, 2019, pp. 169–180.

[44] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for svm," *Mathematical programming*, vol. 127, no. 1, pp. 3–30, 2011.

[45] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.

[46] E. Hazan and S. Kale, "Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 2489–2512, 2014.

# APPENDIX A
# REPRODUCIBILITY DETAILS

## A.1 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 LeNet architecture for training the feature extractor. 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.

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. 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 extractor 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 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 [42] for each domain along with the original train/test splits. The following architecture was used for the feature extractor 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 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.

**DomainNet:** We used four labeled domains from the DomainNet dataset (Real, Quickdraw, Infograph, Sketch) along with their original train/test splits. A ResNet50-v2 pre-trained on ImageNet was employed as the base model for this task. We froze all but the last four layers of the base model and fine-tuned it for each source domain with a learning rate of 1e-5 using the Adam optimizer and a batch size of 64.

```
ResNet50V2(include_top=False, input_shape
    =(224, 224,3), avg='pool'),
Dense(345, activation='softmax')
```

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.

**Mic2Mic:** Similarly, we followed the same train/test splits as in the original dataset provided by the authors of [43]. 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 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.

# APPENDIX B
# PROOFS AND ANALYSIS

## B.1 Optimal Collaborator Selection (OCS)

***Theorem 1.*** Let $D_1$ and $D_2$ be two domains sharing the same labeling function $l$. Let $\theta_{\mathrm{CE}}$ denote the Lipschitz constant of the cross-entropy loss function in $D_1$, and let $\theta$ be the Lipschitz constant of a hypothesis learned on $D_1$. For any two $\theta$-Lipschitz hypotheses $h, h'$, we can derive the following error bound for the cross-entropy (CE) error $\varepsilon_{\mathrm{CE},\mathrm{D}_2}$ in $D_2$:

$$\varepsilon_{\mathrm{CE},\mathrm{D}_2}(h,h') \leq \theta_{\mathrm{CE}} \left(\varepsilon_{\mathrm{L}_1,\mathrm{D}_1}(h,h') + 2\theta W_1(D_1,D_2)\right) \quad (10)$$

where $W_1(D_1,D_2)$ denote the first Wasserstein distance between the domains $D_1$ and $D_2$, and $\varepsilon_{\mathrm{L}_1,\mathrm{D}_1}$ 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:

$$\varepsilon_{\mathrm{L}_1,\mathrm{D}}(h,h') = \mathbb{E}_{x\sim D}\left[|h(x) - h'(x)|\right] \quad (11)$$

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

$$\varepsilon_{\mathrm{CE},\mathrm{D}}(h) = \mathbb{E}_{x\sim D}\left[\left|\log S_{l(x)}h(x)\right|\right], \quad (12)$$

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,

$$\begin{aligned}\varepsilon_{\mathrm{CE},\mathrm{D}}(h,h') &= \mathbb{E}_{x\sim D}\left[|\log S_{l(x)}h(x) - \log S_{l(x)}h'(x)|\right] \\ &= \varepsilon_{\mathrm{L}_1,\mathrm{D}}\left(\log S_l h, \log S_l h'\right) \end{aligned} \quad (13)$$

Further, using the definition of Lipschitz continuity, we have

$$\left|\log S_{l(x)}h(x) - \log S_{l(y)}h(y)\right| \leq \theta_{\mathrm{CE}}|h(x) - h(y)|, \tag{14}$$

where $\theta_{\mathrm{CE}}$ is the Lipschitz constant of the softmax cross-entropy function.

Next, we follow the triangle inequality proof from [7, proof of Lemma 1] to find that

$$\begin{aligned}\varepsilon_{\mathrm{L}_1,\mathrm{D}_2}\left(\log S_l h, \log S_l h'\right) \leq & \varepsilon_{\mathrm{L}_1,\mathrm{D}_1}\left(\log S_l h, \log S_l h'\right) \\ & + 2\theta_{\mathrm{CE}}.\theta W_1(D_1, D_2),\end{aligned} \tag{15}$$

where $\theta$ 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 15 holds everywhere.

Then, by substituting from Eq. 13 and Eq. 14 in Eq. 15, we get Theorem 1:

$$\begin{aligned}\varepsilon_{\mathrm{CE},\mathrm{D}_2}(h, h') &\leq \varepsilon_{\mathrm{CE},\mathrm{D}_1}(h, h') + 2\theta_{\mathrm{CE}}.\theta W_1(D_1, D_2) \\ &\leq \theta_{\mathrm{CE}}(\varepsilon_{\mathrm{L}_1,\mathrm{D}_1}\left(h, h'\right) + 2\theta W_1\left(D_1, D_2\right))\end{aligned} \tag{16}$$

### B.2 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 $D$ ($D_S$ and $D_T$) 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 [35], we know that:

**Proposition 2.** For a given $E_T$, if $D$ 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 $D_S$ and $D_T$ 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 $D_S$ and $D_T$.

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}[\frac{f(x_0)-f(x_{t+1})}{\mu T} &+ (2p-1)L\mu\sigma^2 \\ &+ \frac{\mu L\sigma^2}{2} + \frac{\mu^3 L^3\sigma^2(2p-1)}{2}]\end{aligned} \tag{17}$$

Where $p$ is the sync-up step, $\mu$ 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, $D_S$ and $D_T$ 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 $D_{lazy}$ for our convergence analysis.

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

- $x$ denotes model weights of $D_{lazy}$.
- $f$ denotes the loss function of $D_{lazy}$.
- $\mathcal{D}_s$ and $\mathcal{D}_t$ denote the datasets of source and target feature representations (i.e., outputs of the respective feature extractors).
- $\xi$ 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 $\xi$.
- $\nabla f(\cdot)$ denotes gradients of function $f$.
- $\sigma$ denotes the gradient bound.
- $\mu$ 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:

$$\min_{x\sim\mathbb{R}^N} f(x) = \mathbb{E}_{r^{(s)}\sim\mathcal{D}_s}\log x(r^{(s)}) + \mathbb{E}_{r^{(t)}\sim\mathcal{D}_t}\log[1 - x(r^{(t)})] \tag{18}$$

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

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2^2 \leq L\|x_1 - x_2\|_2^2 \tag{19}$$

Equivalently, we can get:

$$f(x_2) \leq f(x_1) + \nabla f(x_1)^T(x_2 - x_1) + \frac{L}{2}\|x_2 - x_1\|_2^2 \tag{20}$$

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

$$x_{t+1} = x_t - \mu\Delta(x_t, \xi_t) \tag{21}$$

Combine Inequality 20 and Equation 21, at time step $t$ we have:

$$\begin{aligned}&f(x_{t+1}) \leq f(x_t) - \mu\nabla f(x_t)\Delta(x_t, \xi_t) + \frac{\mu^2 L}{2}\|\Delta(x_t, \xi_t)\|_2^2 \\ &\ldots \\ &f(x_1) \leq f(x_0) - \mu\nabla f(x_0)\Delta(x_0, \xi_0) + \frac{\mu^2 L}{2}\|\Delta(x_0, \xi_0)\|_2^2\end{aligned} \tag{22}$$

Sum all inequalities:

$$f(x_{t+1}) \leq f(x_0) - \mu\sum_{t=1}^{T}\nabla f(x_t)\Delta(x_t, \xi_t) + \sum_{t=1}^{T}\frac{\mu^2 L}{2}\|\Delta(x_t, \xi_t)\|_2^2 \tag{23}$$

Take expectation on $\xi_t$ in both sides and re-arrange terms:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\nabla f(x_t)\Delta(x_t, \xi_t)] \leq \frac{f(x_0) - f(x_{t+1})}{\mu T} + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|\Delta(x_t, \xi_t)\|_2^2] \tag{24}$$

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

$$\begin{aligned}&\Delta(x_t, \xi_t) = \frac{1}{p}\sum_{n=0}^{p-1}\nabla f(x_{t_p-n}, \xi_{t_p-n}), t \geq p \\ &t_p = \lfloor t/p \rfloor \times p\end{aligned} \tag{25}$$

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:

$$\begin{aligned}\Delta(x_t,\xi_t) &= \frac{1}{p}\sum_{n=0}^{p-1}\nabla f(x_{t_p-n},\xi_{t_p-n}) \\ &= \frac{1}{p}\sum_{n=0}^{p-1}\nabla f(x_{t_p-n},\xi_{t_p-n}) - \frac{1}{p}\sum_{n=0}^{p-1}\nabla f(x_t,\xi_{t_p-n}) + \\ &\frac{1}{p}\sum_{n=0}^{p-1}\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t) + \nabla f(x_t) \\ &= \nabla f(x_t) + \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] + \\ &\frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)]\end{aligned} \tag{26}$$

Bring Equation 26 back to Equation 24 (Let $e(x_t,\xi_t) = \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] + \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)]$):

$$\begin{aligned}&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\nabla f(x_t)(\nabla f(x_t) + e(x_t,\xi_t))] \leq \\ &\frac{f(x_0)-f(x_{t+1})}{\mu T} + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|(\nabla f(x_t) + e(x_t,\xi_t))\|_2^2], \\ &\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] + \frac{1}{T}\sum_{t=1}^{T}\nabla f(x_t)\mathbb{E}[e(x_t,\xi_t)] \leq \\ &\frac{f(x_0)-f(x_{t+1})}{\mu T} + \frac{\mu L}{2T}\sum_{t=1}^{T}\|\nabla f(x_t)\|_2^2 + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|e(x_t,\xi_t)\|_2^2], \\ &\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{2}{2-\mu L}\bigg(\frac{f(x_0)-f(x_{t+1})}{\mu T} - \\ &\frac{1}{T}\sum_{t=1}^{T}\nabla f(x_t)\mathbb{E}[e(x_t,\xi_t)] + \frac{\mu L}{2T}\sum_{t=1}^{T}\mathbb{E}[\|e(x_t,\xi_t)\|_2^2]\bigg).\end{aligned} \tag{27}$$

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

$$e(x_t,\xi_t) = \underbrace{\frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})]}_{e_1} + \underbrace{\frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)]}_{e_2} \tag{28}$$

For $e_1$:

$$\begin{aligned}e_1(x_t,\xi_t) =& \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_t,\xi_{t_p-n})] \\ =& \frac{1}{p}\sum_{n=0}^{p-1}[\nabla f(x_{t_p-n},\xi_{t_p-n}) - \nabla f(x_{t_p-n+1},\xi_{t_p-n}) + \\ & \nabla f(x_{t_p-n+1},\xi_{t_p-n}) - ... - \nabla f(x_{t_p},\xi_{t_p-n}) \\ & + \nabla f(x_{t_p},\xi_{t_p-n}) - ... + \nabla f(x_{t-1},\xi_{t_p-n}) \\ & - \nabla f(x_t,\xi_{t_p-n})] \\ =& \frac{1}{p}\sum_{n=0}^{p-1}\bigg(\sum_{i=0}^{n-1}[\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})] + \\ & \sum_{j=0}^{t-t_p-1}[\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})]\bigg)\end{aligned} \tag{29}$$

According to the algorithm, we know that:

$$\begin{aligned}x_{t_p-i} &= x_{t_p-(i+1)} - \mu G_{(\lfloor t/p\rfloor-1)\times p} \\ x_{t_p+j+1} &= x_{t_p+j} - \mu G_{\lfloor t/p\rfloor\times p}\end{aligned} \tag{30}$$

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:

$$\begin{aligned}\left\|\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})\right\| &\leq L\mu\left\|G_{(\lfloor t/p\rfloor-1)\times p}\right\| \\ \left\|\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})\right\| &\leq L\mu\left\|G_{\lfloor t/p\rfloor\times p}\right\|\end{aligned} \tag{31}$$

*Assumption 2.* Bounded Gradient. We assume the stochastic gradients are uniformly bounded: (see [44], [45], [46])

$$\mathbb{E}[\|\nabla f(x_t,\xi_t)\|_2^2] \leq \sigma^2 \tag{32}$$

Therefore, we have:

$$\begin{aligned}\left\|\nabla f(x_{t_p-(i+1)},\xi_{t_p-n}) - \nabla f(x_{t_p-i},\xi_{t_p-n})\right\| &\leq L\mu\,\sigma \\ \left\|\nabla f(x_{t_p+j},\xi_{t_p-n}) - \nabla f(x_{t_p+j+1},\xi_{t_p-n})\right\| &\leq L\mu\sigma\end{aligned} \tag{33}$$

Bring Inequality 33 back to Equality 29:

$$\begin{aligned}e_1(x_t,\xi_t) &\leq \frac{1}{p}\sum_{n=0}^{p-1}(n+t-t_p)L\mu\sigma \\ &\leq (2p-1)L\mu\sigma\end{aligned} \tag{34}$$

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

In summary of $e_1$, we get:

$$\begin{aligned}&\mathbb{E}[\nabla f(x_t)e_1(x_t,\xi_t)] \leq \mathbb{E}[\|\nabla f(x_t)\|] \times \|e_1(x_t,\xi_t)\|] \leq (2p-1)L\mu\sigma^2 \\ &\mathbb{E}[\|e_1(x_t,\xi_t))\|_2^2] \leq (2p-1)L^2\mu^2\sigma^2\end{aligned} \tag{35}$$

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 $\xi$ is an unbiased sampled batch, it is obvious that for a certain $n$:

$$\mathbb{E}[\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)] = 0 \tag{36}$$

Also,

$$\begin{aligned}\mathbb{E}[\left\|\nabla f(x_t,\xi_{t_p-n}) - \nabla f(x_t)\right\|_2^2] &\leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \mathbb{E}[\left\|\nabla f(x_t,\xi_{t_p-n})\right\|_2^2] \\ &\leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \sigma^2\end{aligned} \tag{37}$$

Therefore,

$$\begin{aligned}\mathbb{E}[e_2(x_t,\xi_t)] &= 0 \\ \mathbb{E}[\|e_2(x_t,\xi_t))\|_2^2] &\leq \mathbb{E}[\|\nabla f(x_t)\|_2^2] + \sigma^2\end{aligned} \tag{38}$$

Combine 35 and 38 with 27:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\|\nabla f(x_t)\|_2^2] \leq \frac{1}{1-\mu L}[\frac{f(x_0)-f(x_{t+1})}{\mu T}+ (2p-1)L\mu\sigma^2+\frac{\mu L\sigma^2}{2}+\frac{\mu^3L^3\sigma^2(2p-1)}{2}] \quad (39)$$

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.