Accelerated Dynamic Importance Weighting with Versatile Divergence-Minimizing Estimators

Tongtong Fang[1], Nan Lu[2,3], Gang Niu[3], Kenji Fukumizu[1], Masashi Sugiyama[3,4]

***Abstract*—*Importance weighting* (IW) is a golden solver for *joint distribution shift*, where the joint distributions differ between the training and test data. To solve this problem, IW estimates test-to-training density ratios as importance weights and reweights the training losses accordingly. Recent advances in *dynamic IW* (DIW) integrate weight estimation into model training, enabling scalable IW for deep models and achieving strong performance on large modern datasets. Despite its promise, DIW remains limited in two aspects. First, it incurs substantial computational overhead by solving a *kernel mean matching* (KMM)-induced optimization problem to convergence in every mini-batch. Second, it relies solely on KMM for weight estimation, whereas the IW literature contains diverse estimation methods based on different divergence measures. In this paper, we propose *accelerated DIW* (ADIW), a unified and efficient IW framework for deep learning under joint distribution shift. ADIW performs a few lightweight *projected gradient descent* updates that warm-start from previously updated weights, substantially improving efficiency. Moreover, ADIW generalizes DIW into a unified divergence-minimization framework that supports diverse weight-estimation methods in a plug-and-play manner, including those based on the *Kullback–Leibler divergence*, *squared distance*, and *Wasserstein-1 distance*. We establish convergence guarantees for ADIW under mild conditions, and empirical results demonstrate that ADIW achieves state-of-the-art IW performance while being substantially more efficient.**

***Index Terms*—Distribution Shift, Importance Weighting, Divergence Minimization, Deep Learning.**

I. Introduction

DISTRIBUTION shift [1, 2] poses a fundamental challenge for deep learning, as models trained on one distribution often fail to generalize to test data drawn from a different distribution. The most general form of distribution shift is *joint distribution shift*, where the joint densities of the training and test data are different without additional assumptions on the shift structure. A classic approach to joint distribution shift is *importance weighting* (IW) [3, 4], which, in the classification setting considered in this paper, consists of two steps: (i) *weight estimation* (WE), where test-to-training density ratios are estimated using a tiny validation set from the test distribution; and (ii) *weighted classification* (WC), where training losses are reweighted accordingly.

While effective for low-dimensional data and linear models, IW often degrades on high-dimensional and complex data and deep models [5]. To address this limitation, *dynamic IW* (DIW) [6] introduced a *feature extractor* from the deep model and performed WE on feature representations in every mini-batch. Despite its effectiveness on modern datasets, DIW remains limited in computational efficiency and the coverage of WE methods, as detailed below.

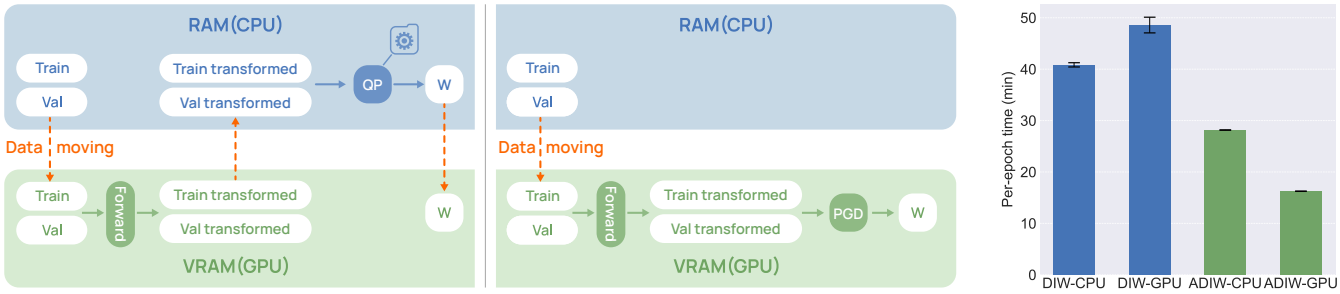
Computational Limitations. DIW has two major computational bottlenecks. (i) *CPU-GPU data transfer overhead.* DIW relies on quadratic programming (QP) solvers for *kernel mean matching* (KMM) [7] in WE, which are typically CPU-centric and thus incur frequent CPU-GPU data transfers (see Figure 1((a)); CPU and GPU denote the central and graphics processing units, respectively). Most mainstream QP solvers (e.g., CVX, CVXOPT, CPLEX, and Gurobi) [1] offer limited or no GPU support, and even GPU-enabled solvers such as cuOSQP [8] typically require CPU-side inputs for problem setup and thus still have CPU-GPU data transfers (see Figure 1((b))). Moreover, even if QP-based WE could be performed entirely on GPU, not all WE objectives are QP problems (e.g., *Kullback–Leibler (KL) importance estimation procedure* (KLIEP) [9]), making a unified implementation difficult. (ii) *Over-solving WE.* In DIW, a KMM problem is solved until convergence in each mini-batch, which is unnecessary since the representations used for WE change consecutively with model update.

Coverage Limitations. Classic two-step IW methods are each built upon a specific *divergence* in the WE step, such as *maximum mean discrepancy* (MMD) in KMM, *KL divergence* in KLIEP, and *squared distance* in *least-squares importance fitting* (LSIF) [10]. The choice of divergence determines how distribution shift is measured and affects both WE and WC through the estimated weights. This motivates a unified framework that supports diverse WE methods based on different divergences within a single implementation, enabling systematic comparison and selection. More broadly, Sugiyama et al. [11] unified many IW methods via *Bregman divergence* [12] minimization, but it is restricted to the Bregman family, does not provide a unified implementation pipeline for different methods, and does not readily support end-to-end deep learning. Although DIW is well suited to deep learning, it remains tightly coupled to a single WE method, i.e., KMM. Recent extensions to alternative WE methods are either application-specific (i.e., language model alignment [13]) or setting-specific (i.e., positive-unlabeled learning [14]), while still inheriting the computational inefficiency of DIW.

In summary, there is still no principled and general IW framework that can flexibly integrate diverse WE methods into

[1] The Institute of Statistical Mathematics, Tokyo, Japan.
[2] University of Bristol, Bristol, U.K.
[3] RIKEN Center for Advanced Intelligence Project, Tokyo, Japan.
[4] The University of Tokyo, Tokyo, Japan.

[1] See https://cvxr.com/cvx/ for CVX, https://cvxopt.org for CVXOPT, https://www.ibm.com/products/ilog-cplex-optimization-studio for CPLEX, and https://www.gurobi.com for Gurobi.



(a) Illustrations of DIW (left) and ADIW (right). (b) Training time.

(a) In DIW, weight estimation (WE) is implemented as a quadratic program (QP) solved until convergence in each mini-batch on the CPU using an external solver, incurring overhead from CPU-GPU data transfers and over-solving WE at each iteration. In contrast, ADIW optimizes the weights via a few *projected gradient descent* (PGD) updates, enabling efficient, fully GPU-accelerated end-to-end training without external QP solvers. (b) Per-epoch end-to-end training time on ImageNet-1K with 0.4 symmetric label noise (minutes). Error bars show standard deviation over 3 runs. DIW-CPU solves WE with CVXOPT on the CPU; DIW-GPU uses cuOSQP on GPU but involves CPU-side preprocessing and additional CPU-GPU data transfers; ADIW-CPU and ADIW-GPU perform PGD-based WE on CPU and fully on GPU, respectively. All methods employ kernel mean matching for WE.

Fig. 1. Comparison of DIW and the proposed ADIW.

deep learning pipelines in a computationally efficient manner.

To address these challenges, we propose *accelerated DIW* (ADIW), a unified and efficient IW framework for deep learning (see Figure 1((a))). ADIW formulates WE as density matching between transformed training and test distributions, where the transformation is induced by a feature extractor from the deep model, and density matching is done by minimizing a specified divergence measure. Instead of solving WE until convergence on CPUs, ADIW performs WE using a small number of lightweight *projected gradient descent* (PGD) [15] updates (typically a single step) directly on GPUs, with each update warm-started from the previously updated weights. This design removes the dependency on external QP solvers and improves efficiency by avoiding unnecessary CPU-GPU data transfers. Moreover, ADIW supports versatile WE methods in a plug-and-play manner, enabling seamless switching and fair comparison within a unified pipeline, which is difficult to achieve with existing heterogeneous implementations.

Overall, ADIW offers a unified and computationally efficient IW framework for deep learning under joint distribution shift. Our contributions can be summarized as follows:

- **Coverage.** We propose ADIW, a unified IW framework that supports a variety of divergences used in classic IW (including the aforementioned MMD, KL divergence, and squared distance) within an end-to-end deep learning pipeline in a plug-and-play manner.
- **Algorithmic.** We analyze the computational bottleneck of DIW, develop an *efficient* PGD-based implementation of ADIW, and prove convergence of the ADIW algorithm to a stationary point of the WC objective.
- **Practical.** We instantiate several classic IW methods within ADIW, introduce a *novel* Wasserstein-1 distance variant, and provide systematic empirical comparisons.
- **Empirical.** We demonstrate that ADIW achieves comparable or better performance than DIW while substantially reducing training time (see Figure 1((b))).

The remainder of this paper is organized as follows. Section II introduces the problem setting and reviews IW and DIW. Section III presents ADIW, followed by instantiations of IW methods in Section IV. Experimental results are reported in Section V. Additional details on the convergence analysis, discussions, and experiments are provided in the appendices.

II. Background

This section provides the necessary background for developing the proposed ADIW framework. First, we introduce the problem setting and then review classic IW and its deep learning extension, DIW.

A. Problem Setting

Let $\boldsymbol{x}$ and y denote the input and output random variables, respectively. In this work, we consider the problem of *joint distribution shift*, where the joint densities of the training and test data are different, i.e., $p_{\mathrm{tr}}(\boldsymbol{x}, y) \neq p_{\mathrm{te}}(\boldsymbol{x}, y)$.

Let $\mathcal{X} \subset \mathbb{R}^d$ be the input domain and $\mathcal{Y} = \{1, \ldots, C\}$ be the label set for a C-class classification problem, where d denotes the input dimension. The objective is to minimize the classification risk:

$$R(\boldsymbol{f}_\theta) = \mathbb{E}_{p_{\mathrm{te}}}[\ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)], \tag{1}$$

where $\boldsymbol{f}_\theta : \mathcal{X} \to \mathbb{R}^C$ is a classifier to be trained parameterized by θ, $\ell : \mathbb{R}^C \times \mathcal{Y} \to (0, +\infty)$ is a surrogate loss function for training $\boldsymbol{f}_\theta$, such as the *softmax cross-entropy loss*, and $\mathbb{E}_{p_{\mathrm{te}}}$ is the expectation over $p_{\mathrm{te}}(\boldsymbol{x}, y)$.

In this setting, we are given a set of training data $\mathcal{D}_{\mathrm{tr}} = \{(\boldsymbol{x}_i^{\mathrm{tr}}, y_i^{\mathrm{tr}})\}_{i=1}^{n_{\mathrm{tr}}} \overset{\text{i.i.d.}}{\sim} p_{\mathrm{tr}}(\boldsymbol{x}, y)$ and a small validation set $\mathcal{D}_{\mathrm{v}} = \{(\boldsymbol{x}_i^{\mathrm{v}}, y_i^{\mathrm{v}})\}_{i=1}^{n_{\mathrm{v}}} \overset{\text{i.i.d.}}{\sim} p_{\mathrm{te}}(\boldsymbol{x}, y)$, where n_{tr} and n_{v} are the sample sizes for $\mathcal{D}_{\mathrm{tr}}$ and $\mathcal{D}_{\mathrm{v}}$, and typically $n_{\mathrm{tr}} \gg n_{\mathrm{v}}$. The goal is to reliably estimate the risk using $\mathcal{D}_{\mathrm{tr}}$ and $\mathcal{D}_{\mathrm{v}}$, and to train $\boldsymbol{f}_\theta$ by minimizing a certain empirical risk on $\mathcal{D}_{\mathrm{tr}}$ that performs better than training $\boldsymbol{f}_\theta$ solely on $\mathcal{D}_{\mathrm{v}}$.

B. Importance Weighting

Importance weighting (IW) is a widely used technique for addressing distribution shift by reweighting training samples

TABLE I
COMPARISON OF CLASSIC IMPORTANCE WEIGHTING (IW), DYNAMIC IW (DIW), AND THE PROPOSED ACCELERATED DIW (ADIW) METHOD.

Category	Divergence	Method	Weight model	Deep learning compatibility	WE optimization	Overall optimization	Comp. cost	Practical adapt.
IPM	MMD	KMM [7]	None	✗	Convex QP (all data, once)	Two-step (WE → WC)	–	Low
		DIW [6]	None	✓	Convex QP (per batch)	End-to-end (WE ↻ WC)	High	High
		ADIW-KM (ours)	None	✓	PGD (per batch)	End-to-end (WE ↻ WC)	Low	High
	Wasserstein-1	ADIW-WA (ours)	None	✓	PGD* (per batch)	End-to-end (WE ↻ WC)	Low	High
f-divergence & Bregman	KL	KLIEP [9]	Parametric	✗	Convex (all data, once)	Two-step (WE → WC)	–	Low
		ADIW-KL (ours)	Parametric	✓	PGD (per batch)	End-to-end (WE ↻ WC)	Low	High
Bregman	Squared Loss	LSIF [10]	Parametric	✗	Convex QP (all data, once)	Two-step (WE → WC)	–	Low
		ADIW-LS (ours)	Parametric	✓	PGD (per batch)	End-to-end (WE ↻ WC)	Low	High

Abbreviations: WE = weight estimation; WC = weighted classification; Comp. cost = per-iteration computational cost (comparable only across end-to-end methods); Practical adapt. = ability to handle complex datasets; QP = quadratic programming; PGD = projected gradient descent; LS = least squares.
Optimization Schemes: Two-step methods optimize WE first and then WC, denoted as "WE → WC", while end-to-end methods jointly optimize WE and WC, denoted as "WE ↻ WC". * For ADIW-WA, PGD requires solving an auxiliary maximization problem to obtain the optimal critic for gradient computation.
Notes: This comparison is conceptual only. The improved adaptability of DIW and ADIW arises from their compatibility with deep models in the WE step. Empirical comparisons with classic IW methods are reported in Fang et al. [6], while comparisons between DIW and ADIW are provided in Section V.

to match the test distribution. Assuming $p_{\mathrm{te}}(\boldsymbol{x}, y)$ is *absolutely continuous* w.r.t. $p_{\mathrm{tr}}(\boldsymbol{x}, y)$, i.e., $p_{\mathrm{te}}(\boldsymbol{x}, y) > 0$ implies $p_{\mathrm{tr}}(\boldsymbol{x}, y) > 0$, the risk in Eq. (1) can be rewritten as an importance-weighted expectation of ℓ over $p_{\mathrm{tr}}(\boldsymbol{x}, y)$:

$$\mathbb{E}_{p_{\mathrm{te}}}\left[\ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)\right] = \mathbb{E}_{p_{\mathrm{tr}}}\left[w^*(\boldsymbol{x}, y)\ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)\right], \qquad (2)$$

where $w^*(\boldsymbol{x}, y) = \frac{p_{\mathrm{te}}(\boldsymbol{x}, y)}{p_{\mathrm{tr}}(\boldsymbol{x}, y)}$ is defined as the *weight* or *importance*, compensating for the difference between the training and test distributions.

Classic IW is usually formulated as a two-step approach:

(i) *Weight Estimation* (WE). In the WE step, $w^*(\boldsymbol{x}, y)$ is estimated from $\mathcal{D}_{\mathrm{tr}}$ and $\mathcal{D}_{\mathrm{v}}$.

(ii) *Weighted Classification* (WC). In the WC step, we plug-in the estimated weights to train $\boldsymbol{f}_\theta$.

Representative classic IW methods include *kernel mean matching* (KMM) [7], the *Kullback–Leibler importance estimation procedure* (KLIEP) [9], and *least-squares importance fitting* (LSIF) [10], as shown in Table I. Additional discussion of related approaches for learning under distribution shift is provided in Appendix A.

C. Dynamic Importance Weighting

Dynamic importance weighting (DIW) [6] was proposed as an end-to-end solution that adapts classic IW to deep learning. In DIW, a feature extractor derived from the current deep model is used to apply a *nonlinear transformation* $\pi : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ that maps data into a d_{r}-dimensional vector space $\mathcal{Z}$, where $d_{\mathrm{r}} \ll d$. The transformed random variable, denoted by $\boldsymbol{z} = \pi(\boldsymbol{x}, y)$, is induced by $(\boldsymbol{x}, y)$ and inherits its randomness. WE on $\boldsymbol{z}$ is considerably easier than on $(\boldsymbol{x}, y)$. The goal of DIW is to learn a set of weights $\mathcal{W} = \{w_i = w(\boldsymbol{z}_i^{\mathrm{tr}})\}_{i=1}^{n_{\mathrm{tr}}}$ such that

$$\mathbb{E}_{p_{\mathrm{te}}}\left[\ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)\right] = \mathbb{E}_{p_{\mathrm{tr}}}\left[w(\boldsymbol{z})\ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)\right].$$

In DIW, two practical choices of π were proposed:

(i) *Hidden-layer-output Transformation.* First, we estimate a class-prior ratio $r_y^* = p_{\mathrm{te}}(y)/p_{\mathrm{tr}}(y)$. Next, $\{(\boldsymbol{x}_i^{\mathrm{tr}}, y_i^{\mathrm{tr}})\}_{i=1}^{n_{\mathrm{tr}}}$ and $\{(\boldsymbol{x}_i^{\mathrm{v}}, y_i^{\mathrm{v}})\}_{i=1}^{n_{\mathrm{v}}}$ are partitioned by the class label y. Finally, WE is performed separately for each class, i.e., C times over C class partitions, according to

$$\begin{aligned}\frac{p_{\mathrm{te}}(\boldsymbol{x}, y)}{p_{\mathrm{tr}}(\boldsymbol{x}, y)} &= \frac{p_{\mathrm{te}}(y) \cdot p_{\mathrm{te}}(\boldsymbol{x} \mid y)}{p_{\mathrm{tr}}(y) \cdot p_{\mathrm{tr}}(\boldsymbol{x} \mid y)} \\ &= \frac{p_{\mathrm{te}}(y)}{p_{\mathrm{tr}}(y)} \cdot \frac{p_{\mathrm{te}}(\boldsymbol{x} \mid y)}{p_{\mathrm{tr}}(\boldsymbol{x} \mid y)} = r_y^* \cdot \frac{p_{\mathrm{te}}(\boldsymbol{z} \mid y)}{p_{\mathrm{tr}}(\boldsymbol{z} \mid y)},\end{aligned}$$

where $\boldsymbol{z}$ denotes the transformed data obtained from the hidden-layer-output transformation, assuming a fixed, deterministic, and invertible π as in Fang et al. [6].

(ii) *Loss-value Transformation.* π transforms each sample into its one-dimensional loss value, i.e., $\pi : (\boldsymbol{x}, y) \mapsto \ell(\boldsymbol{f}_\theta(\boldsymbol{x}), y)$. DIW aims to learn weights such that, under the current model parameters θ_t,

$$\frac{1}{n_{\mathrm{v}}}\sum_{i=1}^{n_{\mathrm{v}}} \ell(\boldsymbol{f}_\theta(\boldsymbol{x}_i^{\mathrm{v}}), y_i^{\mathrm{v}})\big|_{\theta=\theta_t} \approx \frac{1}{n_{\mathrm{tr}}}\sum_{i=1}^{n_{\mathrm{tr}}} w_i\, \ell(\boldsymbol{f}_\theta(\boldsymbol{x}_i^{\mathrm{tr}}), y_i^{\mathrm{tr}})\big|_{\theta=\theta_t}.$$

After applying π, KMM is performed on $\boldsymbol{z}$ using a quadratic programming (QP) solver. Once the weights are estimated, they are fixed, and the feature extractor and the deep model are updated by optimizing the WC objective. By iterating between WE and WC within each mini-batch, DIW provides a modern and conceptually elegant realization of classic IW.

III. ACCELERATED DYNAMIC IMPORTANCE WEIGHTING

Although DIW adapts classic IW to deep learning, it remains limited in computational efficiency and support for di-

verse WE methods, as discussed in Section I. To address these limitations, we now introduce *accelerated dynamic importance weighting* (ADIW), including its unified framework, efficient implementation, algorithm, and convergence analysis.

A. A Unified IW Framework for Deep Learning

Let $\mathcal{P}(\mathcal{Z})$ denote the set of probability distributions on $\mathcal{Z}$ that have densities. Following DIW [6], π is applied to transform the original joint densities $p_{\mathrm{tr}}(\boldsymbol{x}, y)$ and $p_{\mathrm{te}}(\boldsymbol{x}, y)$ into $p_{\mathrm{tr}}(\boldsymbol{z})$ and $p_{\mathrm{te}}(\boldsymbol{z})$, respectively.

For any two densities $p, q \in \mathcal{P}(\mathcal{Z})$, a *probability divergence* is defined as a function $D : \mathcal{P}(\mathcal{Z}) \times \mathcal{P}(\mathcal{Z}) \to \mathbb{R}$ satisfying:

(i) Non-negativity: $D(p \parallel q) \geq 0, \forall p, q \in \mathcal{P}(\mathcal{Z})$,

(ii) Positivity: $D(p \parallel q) = 0$ iff (i.e., if and only if) $p = q$.

Common families of divergences include *integral probability metrics* (IPMs) [16], *f-divergences* [17], and *Bregman divergences* [12], which are briefly introduced below.

IPMs cover a wide range of well-known statistical distances, including the *Wasserstein-1 distance* [18], the *total variation* (TV) *distance* [19] (which is also an f-divergence), and the *maximum mean discrepancy* (MMD) [20, 21]. Formally, given a class $\mathcal{F}$ of real-valued functions on $\mathcal{Z}$, an IPM between distributions p and q is defined as

$$D_{\mathcal{F}}(p \parallel q) = \sup_{f \in \mathcal{F}} \left| \int_{\mathcal{Z}} f \mathrm{d}p - \int_{\mathcal{Z}} f \mathrm{d}q \right|. \tag{3}$$

IPMs are generally *pseudometrics* [22] since most instances of $D_{\mathcal{F}}$ do not satisfy condition (ii). However, some notable exceptions, including the Wasserstein-1 distance, the TV distance, and MMD with *characteristic kernels* [23] (e.g., Gaussian and Laplacian kernels), satisfy both conditions (i) and (ii).

The family of f-divergences includes, but is not limited to, the *Kullback–Leibler* (KL) *divergence* [24], the *Hellinger distance* [25], and the TV distance. Assuming that p is absolutely continuous with respect to q, the f-divergence between p and q, induced by a convex function f, is defined as

$$D_f(p \parallel q) = \int_{\mathcal{Z}} f\left(\frac{\mathrm{d}p}{\mathrm{d}q}\right) \mathrm{d}q. \tag{4}$$

According to Khosravifard et al. [26], D_f satisfies condition (i) if and only if $f(1) \geq 0$, and satisfies condition (ii) if and only if $f(1) = 0$ and f is strictly convex at $x = 1$.

The Bregman divergence is another important family of divergences, with the *squared Euclidean distance* being a representative example. Let $F : \mathcal{Z} \to \mathbb{R}$ be a continuously differentiable and strictly convex function. The Bregman divergence induced by F between p and q is defined as

$$D_F(p \parallel q) = F(p) - F(q) - \langle \nabla F(q), p - q \rangle, \tag{5}$$

which satisfies both conditions (i) and (ii).

After applying π, WE is formulated as a density-matching problem between the transformed test density $p_{\mathrm{te}}(\boldsymbol{z})$ and the weighted transformed training density $w(\boldsymbol{z})\, p_{\mathrm{tr}}(\boldsymbol{z})$. Specifically, for $p_{\mathrm{te}}, w \cdot p_{\mathrm{tr}} \in \mathcal{P}(\mathcal{Z})$, the goal of WE is to find w within a feasible set that minimizes a probability divergence $J(w)$ between p_{te} and $w \cdot p_{\mathrm{tr}}$:

$$J(w) := D\left(p_{\mathrm{te}} \parallel w \cdot p_{\mathrm{tr}}\right), \tag{6}$$

Algorithm 1 Accelerated dynamic importance weighting.

Require: training minibatch $\mathcal{S}^{\mathrm{tr}}$ with index set $\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}$, validation minibatch $\mathcal{S}^{\mathrm{v}}$, current model $\boldsymbol{f}_\theta$, current weight vector $\mathcal{W}$, WE objective $\widehat{J}$ with constraint set Q, step size η_t, number of WE iterations T

`retrieve` initial weights $\mathcal{W}_1 \leftarrow \mathcal{W}[\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}]$
`forward` the input parts of $\mathcal{S}^{\mathrm{tr}}$ & $\mathcal{S}^{\mathrm{v}}$
`compute` the loss values as $\mathcal{L}^{\mathrm{tr}}$ & $\mathcal{L}^{\mathrm{v}}$
for t = 1 to T **do**
  `compute` gradients $\nabla_{\mathcal{W}_t} \widehat{J}$ from $\mathcal{L}^{\mathrm{tr}}$ & $\mathcal{L}^{\mathrm{v}}$
  `update` $\mathcal{W}_{t+1} \leftarrow \mathcal{W}_t - \eta_t \nabla_{\mathcal{W}_t} \widehat{J}$
  `project` $\mathcal{W}_{t+1}$ onto Q
end for
`update` weights in $\mathcal{W}$ for $\mathcal{S}^{\mathrm{tr}}$: $\mathcal{W}[\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}] \leftarrow \mathcal{W}_T$
`weight` the empirical risk $\widehat{R}(\boldsymbol{f}_\theta)$ by $\mathcal{W}_T$
`backward` $\widehat{R}(\boldsymbol{f}_\theta)$ and `update` θ

where w may represent either a weight vector or a parametric weight model. Let $\widehat{J}(\mathcal{W})$ denote its empirical approximation. Then, the empirical WE objective is

$$\min_{\mathcal{W} \in Q} \widehat{J}(\mathcal{W}), \tag{7}$$

where Q denotes the feasible constraint set for $\mathcal{W}$.

B. Efficient Implementation of ADIW

Building upon the above unified framework, we now present the efficient implementation of ADIW.

In DIW, the WE optimization is solved to convergence at every iteration, which is often unnecessary for two reasons. First, since WE operates on transformed representations, it only approximates true weights, making full convergence overly restrictive. Second, WE and WC alternate at the mini-batch level: once WC updates the classifier, the transformed representations used by WE change accordingly, rendering the previous exact WE solution obsolete. As a result, an inexact but feasible update may be sufficient to ensure progress.

These insights lead us to the design of ADIW, where WE is performed using a lightweight *projected gradient descent* (PGD), which is also known as *gradient projection* [15]. Specifically, if $\widehat{J}(\mathcal{W})$ is a differentiable and convex function with respect to $\mathcal{W}$, we can update $\mathcal{W}$ by a *gradient descent* step followed by a projection onto the constraint set. Assume $\widehat{J}(\mathcal{W})$ is continuously differentiable and Q is a convex and compact set encoding the constraints. Then the empirical WE objective in Eq. (7) can be solved as

$$\mathcal{W}_{t+1} \leftarrow \Pi_Q\left(\mathcal{W}_t - \eta_t \nabla \widehat{J}(\mathcal{W}_t)\right), \tag{8}$$

where $\mathcal{W}_t$ is the set of weights at iteration t, $\eta_t \in (0, \infty)$ is the step size, and Π_Q is the projection operator onto Q to satisfy the constraints. By performing only a few PGD-based updates per mini-batch, ADIW performs WE entirely on GPUs without QP solvers, greatly improving efficiency.

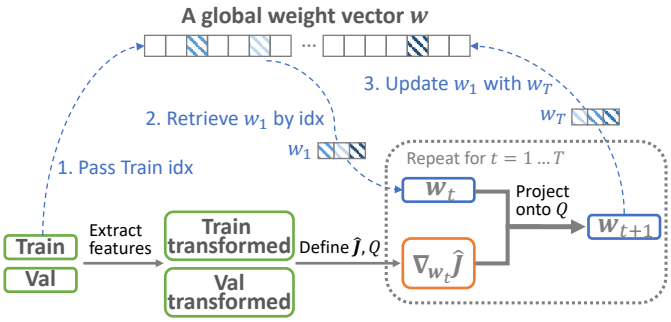


Fig. 2. Illustration of WE in ADIW. "idx" denotes the index; "Train" and "Val" denote training and validation data, respectively.

C. The ADIW Algorithm

The ADIW algorithm is summarized in Algorithm 1, using the loss-value transformation as π. The variant based on the hidden-layer-output transformation is provided in Algorithm 2 in Appendix D. For simplicity, we focus on the loss-value transformation in the main text, while Section V-C3 presents an ablation study comparing the two transformations. Although our discussion focuses on classification, the concept of ADIW may also be applied to other tasks (e.g., regression), provided that an appropriate transformation π is designed for the task.

The above algorithm applies to non-parametric IW methods (e.g., those based on MMD or the Wasserstein-1 distance). For parametric methods (e.g., those based on the KL divergence or the squared distance), $\mathcal{W}$ denotes the parameters of the weight model, whose dimensionality is determined by the validation set. In this case, the algorithm updates the weight model parameters rather than individual weights, using validation minibatch indices. Section IV presents several instantiations of classic IW methods within ADIW.

An illustration of WE in ADIW is given in Figure 2. ADIW maintains a global weight vector $\mathcal{W}$, initialized to all ones in the first epoch and updated thereafter. In each iteration, mini-batches of training and validation data are sampled, and a subset $\mathcal{W}_1$ corresponding to the sampled training indices is retrieved to initialize WE. Starting from $\mathcal{W}_1$, the WE loop performs T steps of gradient descent on the transformed data, where each step is followed by a projection onto Q, yielding the updated weights $\mathcal{W}_T$. The resulting weights are then written back into the global vector $\mathcal{W}$, so that subsequent mini-batches start from the most recently updated values. The same weights $\mathcal{W}_T$ are also used in the WC step to update the model parameters θ.

We note that ADIW defines the validation data loader, used to repeatedly sample validation mini-batches, once outside the training loop and recreates it only when exhausted [2], instead of reconstructing it at every iteration as in DIW. This eliminates redundant loader construction, which has little impact on small datasets but yields noticeable efficiency gains for large-scale datasets (see Section V-C1).

To highlight the advantages of ADIW, Table I presents a systematic comparison of classic IW methods, DIW and ADIW. Classic two-step IW methods are limited by their incompatibility with deep learning, whereas DIW mitigates this by performing WE on the transformed data in a mini-batch manner, but remains coverage restricted and computationally expensive. ADIW overcomes these drawbacks by extending the framework to support diverse divergences and replacing the costly QP optimization with lightweight, GPU-accelerated PGD updates, thereby achieving both broader coverage and scalability for modern deep learning applications.

[2]All methods in Figure 1((b)) adopt this improvement.

D. Convergence Analysis

Here we present the convergence analysis of ADIW. We consider a slightly modified version of Algorithm 1, where the stochastic gradient generator returns an unbiased estimate of the true gradient at each iteration. Let the empirical weighted objective be

$$R_{\mathrm{tr}}(\theta, w) = \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} w(\boldsymbol{z}_i)\ell(\boldsymbol{f}_\theta(\boldsymbol{x}_i), y_i), \tag{9}$$

where each data point $(\boldsymbol{x}_i, y_i)$ is mapped to a representation $\boldsymbol{z}_i = \pi_\theta(\boldsymbol{x}_i, y_i)$ via a feature extractor π_θ. $w^*(\boldsymbol{z}) = \frac{p_{\mathrm{te}}(\boldsymbol{z})}{p_{\mathrm{tr}}(\boldsymbol{z})}$ is the true importance weight, and $w(\boldsymbol{z})$ is its learned approximation via Eq. (7). We next present the main convergence result, with the proof deferred to Appendix C.

Theorem III.1 (Convergence of ADIW)**.** *Suppose Assumptions B.1–B.3 in Appendix B hold. Let the surrogate loss ℓ be bounded as $\ell < M$ and $M > 0$, the learned weights satisfy $w \le B$, $B > 0$, the number of inner-loop steps be T, and the number of outer-loop epochs be S. Set the outer-loop learning rate as $\alpha_s = \frac{c}{\sqrt{S}}$ for some constant $c > 0$. Then, the output of ADIW after S epochs satisfies*

$$\mathbb{E}[\|\nabla R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)\|^2] \le \frac{\Delta}{\sqrt{S}}, \tag{10}$$

where $\|\cdot\|$ denotes L_2 norm, $\Delta = \frac{2BM}{c} + 2cL\sigma^2 + \frac{4cMT\sigma^2}{n_{\mathrm{tr}}}$. Equivalently, ADIW finds an ϵ-stationary point, i.e.,

$$\mathbb{E}[\|\nabla R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)\|^2] \le \epsilon$$

within $O\left(1/\epsilon^2\right)$ iterations.

The theorem establishes that ADIW converges to a stationary point of the weighted training objective in Eq. (9) at the standard $O(1/\sqrt{S})$ rate. The constant in the leading-order term includes a term that decreases with the training sample size n_{tr} and vanishes for large datasets. Consequently, for sufficiently large datasets, convergence is governed primarily by optimization complexity, whereas for smaller datasets statistical estimation error also plays a role.

IV. Instantiations of Classic IW Methods

In this section, we illustrate how classic IW methods can be instantiated within the unified ADIW framework, each corresponding to a particular divergence. We also introduce a novel IW variant based on the Wasserstein-1 distance.

A. Maximum Mean Discrepancy

We first consider ADIW with the *maximum mean discrepancy* (MMD), which recovers KMM [7]. Let $\mathcal{H}$ be a Hilbert space of real-valued functions $f : \mathbb{R}^{d_r} \to \mathbb{R}$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, or $\mathcal{H}$ be a *reproducing kernel Hilbert space* (RKHS), where $k : (\boldsymbol{z}, \boldsymbol{z}') \mapsto \langle \phi(\boldsymbol{z}), \phi(\boldsymbol{z}') \rangle_{\mathcal{H}}$ is the reproducing kernel of $\mathcal{H}$ with a feature map $\phi : \mathbb{R}^{d_r} \to \mathcal{H}$. Then, the objective of weight estimation in Eq. (6) with MMD is given by

$$D_{\mathrm{MMD}}(p_{\mathrm{te}} \parallel w \cdot p_{\mathrm{tr}}) = \sup_{\|f\|_{\mathcal{H}} \leq 1} \mathbb{E}_{p_{\mathrm{te}}}[f(\boldsymbol{z})] - \mathbb{E}_{p_{\mathrm{tr}} \cdot w(\boldsymbol{z})}[f(\boldsymbol{z})].$$

Let $\widehat{J}_{\mathrm{KM}}$ denote the empirical objective of KMM, which minimizes the empirical estimate of the squared MMD with sample averages:

$$\widehat{J}_{\mathrm{KM}}(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{K} \boldsymbol{w} - 2\boldsymbol{k}^\top \boldsymbol{w} \quad \text{subject to } \ w_i \geq 0 \text{ and } |\frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} w_i - 1| \leq \epsilon, \tag{11}$$

where $\boldsymbol{w} \in \mathbb{R}^{n_{\mathrm{tr}}}$ be the weight vector, $\boldsymbol{K}_{ij} := k(\boldsymbol{z}_i^{\mathrm{tr}}, \boldsymbol{z}_j^{\mathrm{tr}})$, $\boldsymbol{k}_i := \frac{n_{\mathrm{tr}}}{n_{\mathrm{v}}} \sum_{j=1}^{n_{\mathrm{v}}} k(\boldsymbol{z}_i^{\mathrm{tr}}, \boldsymbol{z}_j^{\mathrm{v}})$, and $\epsilon > 0$ is a slack variable.

B. Kullback–Leibler Divergence

Next, we consider ADIW with the *Kullback–Leibler* (KL) *divergence*, as used in KLIEP [9]. Let the weight $w(\boldsymbol{z})$ be modeled by a linear-in-parameter model $g(\boldsymbol{z}) = \sum_{l=1}^{b} \beta_l \psi_l(\boldsymbol{z}) = \boldsymbol{\beta}^\top \boldsymbol{\psi}(\boldsymbol{z})$, where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_b)^\top \in \mathbb{R}^b$ are learnable parameters and $\boldsymbol{\psi}(\boldsymbol{z})$ is a vector of b *basis functions*. That is, $\boldsymbol{\psi}(\boldsymbol{z}) = (\psi_1(\boldsymbol{z}), \ldots, \psi_b(\boldsymbol{z}))^\top \in \mathbb{R}^b$, where $\psi_l(\boldsymbol{z}) := \exp\left(-\frac{\|\boldsymbol{z} - \boldsymbol{z}_l^{\mathrm{v}}\|^2}{2\sigma^2}\right)$, and $\{\boldsymbol{z}_l^{\mathrm{v}}\}_{l=1}^b$ are kernel centers from the validation data. With the KL divergence, Eq. (6) becomes

$$\begin{aligned} D_{\mathrm{KL}}(p_{\mathrm{te}} \parallel w \cdot p_{\mathrm{tr}}) &= \mathbb{E}_{p_{\mathrm{te}}}\left[\log \frac{p_{\mathrm{te}}(\boldsymbol{z})}{g(\boldsymbol{z}) p_{\mathrm{tr}}(\boldsymbol{z})}\right] \\ &= \underbrace{\mathbb{E}_{p_{\mathrm{te}}}\left[\log \frac{p_{\mathrm{te}}(\boldsymbol{z})}{p_{\mathrm{tr}}(\boldsymbol{z})}\right]}_{:=\text{const.}} - \mathbb{E}_{p_{\mathrm{te}}}[\log g(\boldsymbol{z})]\,. \end{aligned}$$

After dropping the first constant term and plugging in the definition of $g(\boldsymbol{z})$, the empirical objective of KLIEP corresponds to the empirical counterpart of the second term, which is defined as

$$\widehat{J}_{\mathrm{KL}}(\boldsymbol{\beta}) = -\frac{1}{n_{\mathrm{v}}} \sum_{i=1}^{n_{\mathrm{v}}} \log \boldsymbol{\beta}^\top \boldsymbol{\psi}(\boldsymbol{z}_i^{\mathrm{v}}) \quad \text{subject to } \ \boldsymbol{\beta}^\top \bar{\boldsymbol{\psi}}_{\mathrm{tr}} = 1, \quad \boldsymbol{\beta} \geq \boldsymbol{0}_b, \tag{12}$$

where $\bar{\boldsymbol{\psi}}_{\mathrm{tr}} := \frac{1}{n_{\mathrm{tr}}} \sum_{j=1}^{n_{\mathrm{tr}}} \boldsymbol{\psi}(\boldsymbol{z}_j^{\mathrm{tr}})$ and $\boldsymbol{0}_b$ are the b-dimensional vectors with all zeros. $\boldsymbol{\beta} \geq \boldsymbol{0}_b$ is applied in the element-wise manner.

C. Squared Distance

We then consider the case of the *squared distance*, as used in *least-squares importance fitting* (LSIF) [10]. Let us model $w(\boldsymbol{z})$ by $g(\boldsymbol{z})$ in the same way as KLIEP. Then, the objective is to minimize the squared distance between w and g:

$$\begin{aligned} D_{\mathrm{SQ}}(w \parallel g) &= \tfrac{1}{2}\mathbb{E}_{p_{\mathrm{tr}}}\left[(w(\boldsymbol{z}) - g(\boldsymbol{z}))^2\right] \\ &= \tfrac{1}{2}\mathbb{E}_{p_{\mathrm{tr}}}\left[g^2(\boldsymbol{z})\right] - \mathbb{E}_{p_{\mathrm{te}}}[g(\boldsymbol{z})] + \underbrace{\tfrac{1}{2}\mathbb{E}_{p_{\mathrm{tr}}}\left[w^2(\boldsymbol{z})\right]}_{:=\text{const.}}. \end{aligned}$$

After approximating the first two terms with empirical averages and adding a regularizer $\boldsymbol{1}_b^\top \boldsymbol{\beta} := \sum_{l=1}^{b} |\beta_l|$, the empirical objective of LSIF is given by

$$\widehat{J}_{\mathrm{LS}}(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^\top \widehat{\boldsymbol{H}} \boldsymbol{\beta} - \widehat{\boldsymbol{h}}^\top \boldsymbol{\beta} + \lambda \boldsymbol{1}_b^\top \boldsymbol{\beta} \quad \text{subject to } \ \boldsymbol{\beta} \geq \boldsymbol{0}_b, \tag{13}$$

where $\widehat{\boldsymbol{H}}_{l,l'} = \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} \psi_l(\boldsymbol{z}_i^{\mathrm{tr}}) \psi_{l'}(\boldsymbol{z}_i^{\mathrm{tr}})$, $\widehat{\boldsymbol{h}}_l = \frac{1}{n_{\mathrm{v}}} \sum_{j=1}^{n_{\mathrm{v}}} \psi_l(\boldsymbol{z}_j^{\mathrm{v}})$, and $\lambda \geq 0$ is a regularization parameter. Note that for KLIEP and LSIF, the optimization is performed with respect to $\boldsymbol{\beta}$, and the importance weights are subsequently computed as $\boldsymbol{\beta}^\top \boldsymbol{\psi}(\boldsymbol{z})$.

D. Wasserstein-1 Distance

Finally, we introduce a novel IW variant based on the *Wasserstein-1 distance* (also known as the *Earth Mover's Distance* [27]). For the test density p_{te} and the weighted training density $w \cdot p_{\mathrm{tr}}$ defined on a compact metric space, the Wasserstein-1 distance is defined as

$$D_{\mathrm{W}_1}(p_{\mathrm{te}} \parallel w \cdot p_{\mathrm{tr}}) = \inf_{\gamma \in \Pi(p_{\mathrm{te}}, w \cdot p_{\mathrm{tr}})} \mathbb{E}_{(\boldsymbol{z}, \boldsymbol{z}') \sim \gamma}[\|\boldsymbol{z} - \boldsymbol{z}'\|],$$

where $\Pi(p_{\mathrm{te}}, w \cdot p_{\mathrm{tr}})$ denotes the set of all couplings γ of p_{te} and $w \cdot p_{\mathrm{tr}}$, i.e., joint distributions whose marginals are p_{te} and $w \cdot p_{\mathrm{tr}}$. Its dual representation can be derived from the *Kantorovich–Rubinshtein Theorem* [28]:

$$\begin{aligned} D_{\mathrm{W}_1}(p_{\mathrm{te}} \parallel w \cdot p_{\mathrm{tr}}) &= \sup_{\|\Phi\|_{\mathrm{Lip}} \leq 1} \mathbb{E}_{p_{\mathrm{te}}}[\Phi(\boldsymbol{z})] - \mathbb{E}_{w \cdot p_{\mathrm{tr}}}[\Phi(\boldsymbol{z})] \\ &= \sup_{\|\Phi\|_{\mathrm{Lip}} \leq 1} \mathbb{E}_{p_{\mathrm{te}}}[\Phi(\boldsymbol{z})] - \mathbb{E}_{p_{\mathrm{tr}}}[w(\boldsymbol{z})\Phi(\boldsymbol{z})], \end{aligned}$$

where $\|\Phi\|_{\mathrm{Lip}}$ denotes the smallest constant L (with $L = 1$ for 1-Lipschitz functions) such that $|\Phi(\boldsymbol{z}) - \Phi(\boldsymbol{z}')| \leq L\|\boldsymbol{z} - \boldsymbol{z}'\|$ holds for all $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^{d_{\mathrm{r}}}$ and the supremum is taken over all 1-Lipschitz *critic* functions $\Phi : \mathbb{R}^{d_{\mathrm{r}}} \to \mathbb{R}$.

In practice, the critic Φ is approximated by a parameterized family $\{\Phi_\nu\}_{\nu \in \mathcal{V}}$, with each Φ_ν implemented as a neural network with parameters $\nu \in \mathcal{V}$. The optimal critic Φ^* is obtained by solving

$$\max_{\nu \in \mathcal{V},\, \|\Phi_\nu\|_{\mathrm{Lip}} \leq 1} \mathbb{E}_{p_{\mathrm{te}}}[\Phi_\nu(\boldsymbol{z})] - \mathbb{E}_{w \cdot p_{\mathrm{tr}}}[\Phi_\nu(\boldsymbol{z})]. \tag{14}$$

To minimize D_{W_1} with respect to the weights $w(\boldsymbol{z})$, we compute its gradient as

$$\nabla_w D_{\mathrm{W}_1} = -\Phi^*(\boldsymbol{z}) p_{\mathrm{tr}}(\boldsymbol{z}). \tag{15}$$

Directly enforcing the Lipschitz constraint for neural networks is known to be challenging [29]. Fortunately, the optimal critic Φ^* satisfies the property of exhibiting a unit gradient norm almost everywhere under the optimal coupling π between p_{te} and $w \cdot p_{\mathrm{tr}}$ [30]. Leveraging this property, we adopt the *gradient penalty* approach [30], which encourages the gradient norm of

the critic to stay close to one along paths between samples from p_{te} and $w \cdot p_{\mathrm{tr}}$. The resulting objective is given by

$$J_{W_1}(\nu) = \mathbb{E}_{w \cdot p_{\mathrm{tr}}}[\Phi_\nu(\boldsymbol{z})] - \mathbb{E}_{p_{\mathrm{te}}}[\Phi_\nu(\boldsymbol{z})] + \kappa\, \mathbb{E}_{p'}\left[(\|\nabla_{\boldsymbol{z}} \Phi_\nu(\boldsymbol{z})\| - 1)^2\right], \tag{16}$$

where p' denotes the distribution of points uniformly sampled along straight-line interpolations between pairs of samples $\boldsymbol{z} \sim p_{\mathrm{te}}$ and $\boldsymbol{z}' \sim w \cdot p_{\mathrm{tr}}$, i.e., $\tilde{\boldsymbol{z}} = \epsilon \boldsymbol{z} + (1-\epsilon)\boldsymbol{z}'$ with $\epsilon \sim \mathrm{Uniform}(0,1)$. The penalty coefficient $\kappa > 0$ controls the strength of regularization. Before optimizing the full objective, we warm-start the critic Φ_ν by training it with only the first two terms for a few iterations (i.e., without the gradient penalty), which provides a stable initialization.

We note that the optimization procedure in the previous examples typically involves bi-level optimization: an outer optimization that minimizes the weighted classification risk with respect to the classifier parameters $\boldsymbol{f}_\theta$, and an inner optimization that learns the importance weights $w(\boldsymbol{z})$. However, employing the Wasserstein-1 distance introduces an additional level of optimization complexity. Specifically, within each update step of the weight estimation (inner optimization), another maximization problem (Eq. (14)) must be solved to obtain the optimal critic Φ^* for evaluating the Wasserstein-1 distance. Nevertheless, the additional maximization is performed using only a few lightweight updates on mini-batch representations, resulting in computational cost comparable to other ADIW variants in practice (see Figure 4).

V. Experiments

This section presents an empirical evaluation of ADIW on benchmark datasets. The ADIW with KMM, KLIEP, LSIF, and using Wasserstein-1 distance for weight estimation (WE) is named as ADIW-KM, ADIW-KL, ADIW-LS, and ADIW-WA respectively.

A. Experimental Setup

The experiments were conducted under two types of distribution shift, class-prior shift and label noise, using four benchmark datasets: *Fashion-MNIST* [31], *CIFAR-10*, *CIFAR-100* [32], and *ImageNet-1K*, also known as *ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC 2012)* [33]. We adopted LeNet-5 [34] as the base model for Fashion-MNIST, trained by SGD [35], and ResNet-18 [36] as the base model for CIFAR-10/100 and ImageNet-1K, trained by Adam [37]. The baselines included:

- *Val-Only* trains the model from scratch using only the limited validation data;
- *Uniform* uses the weights of all ones to weight the training data;
- *Random* uses random weights drawn from the *rectified Gaussian distribution*;
- *L2R* is the learning to reweight examples method [38];
- *MW-Net* is short for meta-weight-net [39], a parametric version of L2R;
- *DIW* is dynamic importance weighting [6].

TABLE II
Accuracy (%) and time on Fashion-MNIST under class-prior shift.

Method	$\mu = 0.5, \rho = 50$		$\mu = 0.5, \rho = 100$	
	Acc. (std)	Time	Acc. (std)	Time
Val-Only	65.55 (1.08)	0.02	65.55 (1.08)	0.02
DIW	**78.09 (0.46)**	22.97	**74.19 (0.39)**	17.88
ADIW-KM	**77.69 (0.22)**	6.84	**74.23 (0.52)**	5.81
ADIW-KL	**77.86 (0.11)**	8.39	**74.25 (0.30)**	5.83
ADIW-LS	**76.93 (1.25)**	7.67	**74.13 (0.55)**	5.70
ADIW-WA	**77.37 (0.43)**	7.64	**73.14 (1.50)**	7.79

Time denotes the training time in seconds. Results are reported as mean accuracy (standard deviation) over the last ten epochs (3 trials). Best and statistically comparable methods (t-test at 1% level) are shown in bold.

Additional details of the datasets, models, and experimental setups are provided in Appendix E. We next describe the specific setups for class-prior shift and label noise, respectively.

1) Experiments under Class-prior Shift: To simulate class-prior shift, we modified the training set of Fashion-MNIST following Buda et al. [40]. All classes were randomly partitioned into majority and minority groups, with a fraction μ of the classes designated as minority. All classes within each group were assigned equal sample sizes, but the sample size per class in the minority group was reduced relative to that of the majority group, with the imbalance ratio defined as ρ. For validation set, we randomly sampled 10 instances per class from the training data. The original test set of Fashion-MNIST was left unchanged for evaluation.

2) Experiments under Label Noise: The label-noise experiments were conducted on Fashion-MNIST, CIFAR-10, CIFAR-100, and ImageNet-1K. We imposed label noise to the training set while keeping the validation/test set intact. Two types of label noise were imposed on the training data: *pair flip* [41], where a label may flip to its neighborhood class with a probability, and *symmetric flip* [42], where a label may flip to all other classes with equal probability (i.e., the *noise rate*). For ImageNet-1K, we trained the model from scratch on the original training set with synthetic label noise, randomly selected 20,000 training data as a clean validation set, and evaluated on the original ImageNet-1K validation set. For other datasets, we created the clean validation set by randomly sampling 1,000 data from the training set.

B. Experimental Results

In this section, we show the empirical results on benchmark datasets under class-prior shift and label noise, respectively.

1) Class-prior Shift Experiments: Table II reports mean accuracy and average per-epoch training time under two class-prior shift settings: $\rho = 50$ and $\rho = 100$. Across both settings, all ADIW variants achieve accuracy comparable to DIW while consistently requiring substantially less training time per epoch.

2) Label-noise Experiments: Figure 3 shows classification accuracy of DIW and ADIW variants on Fashion-MNIST, CIFAR-10, and CIFAR-100 under 0.3 pair, 0.4 and 0.5 symmetric label noise, and a complete accuracy summary across all baselines is provided in Table VIII. Overall, the ADIW methods achieve performance that is competitive with, or

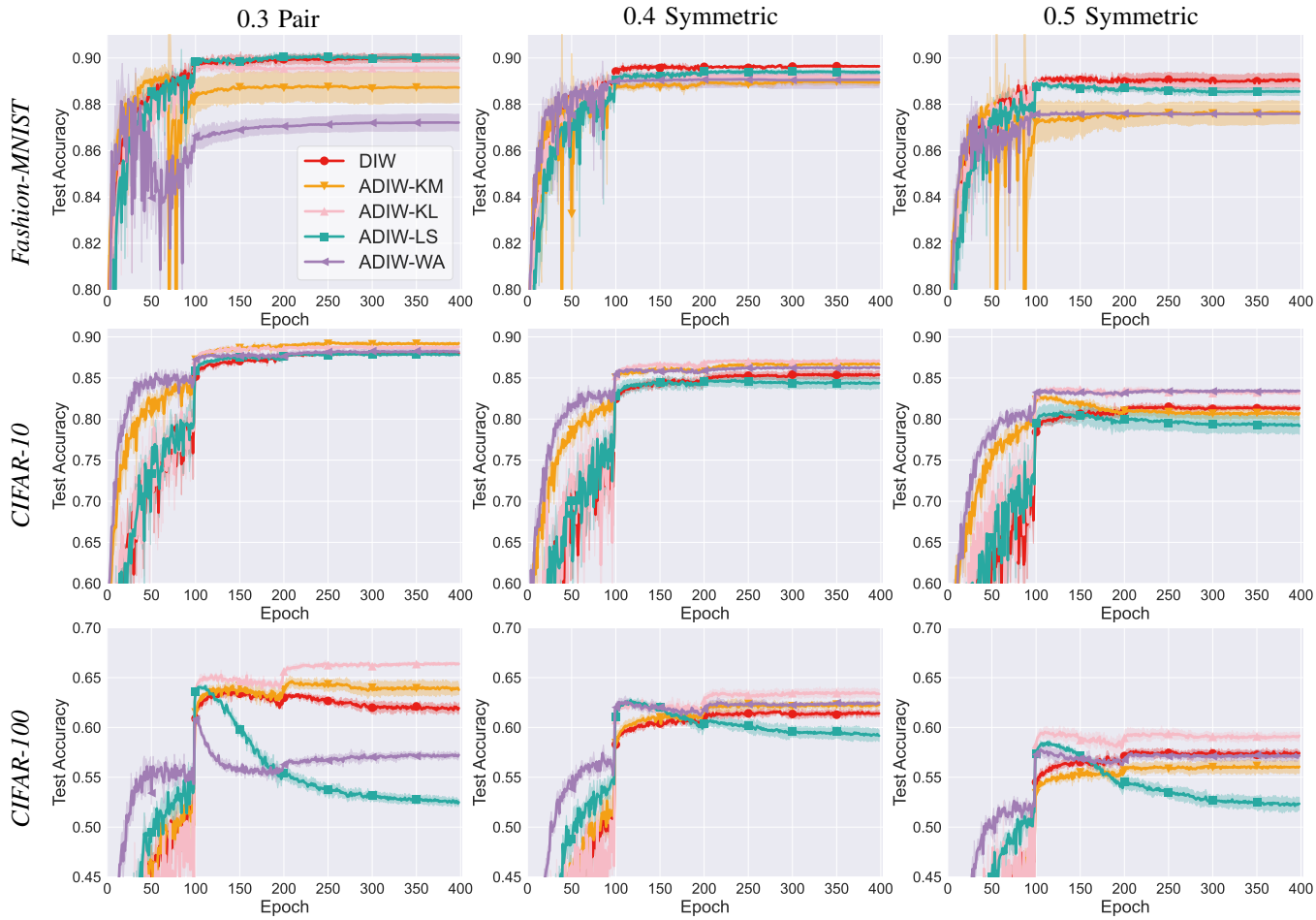


Fig. 3. Experimental results on Fashion-MNIST and CIFAR-10/100 under label noise (3 trials).

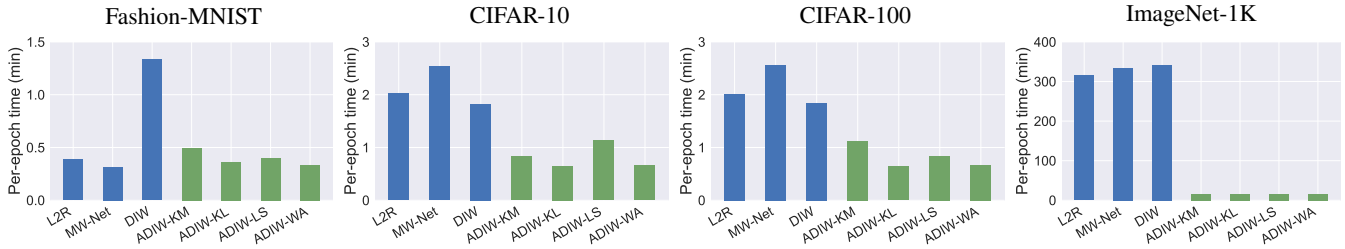


Fig. 4. Training time (minutes) under 0.4 symmetric label noise.

superior to, DIW and other baselines. Both DIW and ADIW also tend to be robust as the label noise rate increases. An exception is that ADIW-WA underperforms under 0.3 pair-flip noise on Fashion-MNIST and CIFAR-100, and ADIW-LS exhibits lower robustness than other methods on the more challenging CIFAR-100 dataset.

Table III reports results on ImageNet-1K validation data under 0.4 symmetric label noise. Baselines relying on differentiating through model parameters or quadratic programming solvers (i.e., L2R, MW-Net, and DIW) are computationally infeasible at this scale and are thus omitted. In contrast, ADIW enables efficient end-to-end GPU training and substantially outperforms naive baselines (i.e., Val-Only and Uniform). For ADIW, ADIW-KL achieves the best top-1 accuracy, while all variants achieve comparable top-5 accuracy.

TABLE III
TOP-1/5 ACCURACY (%) ON IMAGENET-1K UNDER LABEL NOISE.

	Val-Only	Uniform	ADIW-KM	ADIW-KL	ADIW-LS	ADIW-WA
Top-1	11.60 (0.03)	57.66 (0.20)	64.73 (0.08)	**65.86 (0.16)**	64.16 (0.07)	63.29 (0.12)
Top-5	25.77 (0.27)	78.89 (0.10)	**84.66 (0.03)**	**84.82 (0.04)**	**84.65 (0.08)**	**83.59 (0.18)**

Results are reported as mean accuracy (standard deviation) over the last ten epochs (3 trials). Best and statistically comparable methods (t-test at the 1% significance level) are shown in bold.
Note: L2R, MW-Net, and DIW are computationally expensive and unable to complete training on ImageNet-1K. Their training times are reported separately in Figure 4 and Table IX in Appendix F-D.

In addition, Figure 4 visualizes per-epoch end-to-end training time under 0.4 symmetric label noise, with complete

TABLE IV
STAGE-WISE BREAKDOWN OF TOTAL CPU AND CUDA TIME ON IMAGENET-1K (SECONDS; PERCENTAGES IN PARENTHESES).

Stage	DIW		DIW_Val		ADIW_CPU		ADIW_GPU	
	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)
Fetch tr data	0.77 (1.02)	0.16 (0.47)	0.79 (3.47)	0.14 (0.63)	0.80 (10.65)	0.16 (2.04)	2.46 (71.96)	0.17 (8.20)
Fetch val data	43.31 (57.07)	0.14 (0.43)	0.30 (1.31)	0.14 (0.62)	0.29 (3.83)	0.15 (1.97)	0.28 (8.09)	0.16 (7.93)
Forward tr data	0.07 (0.09)	0.57 (1.72)	0.05 (0.22)	0.54 (2.35)	0.06 (0.82)	0.52 (6.60)	0.09 (2.52)	0.52 (25.70)
Forward val data	0.09 (0.11)	0.59 (1.78)	0.06 (0.25)	0.54 (2.32)	0.06 (0.78)	0.51 (6.57)	0.09 (2.74)	0.52 (25.66)
Get tr loss	0.01 (0.02)	0.00 (0.00)	0.00 (0.02)	0.00 (0.00)	0.00 (0.05)	0.00 (0.00)	0.00 (0.13)	0.00 (0.01)
Get val loss	0.01 (0.02)	0.00 (0.00)	0.00 (0.02)	0.00 (0.00)	0.00 (0.05)	0.00 (0.00)	0.00 (0.12)	0.00 (0.01)
Estimate weights	31.18 (41.07)	30.97 (93.44)	21.32 (93.28)	21.11 (91.22)	6.06 (80.56)	5.86 (75.03)	0.21 (6.22)	0.04 (2.17)
Weight tr loss	0.21 (0.28)	0.70 (2.12)	0.13 (0.58)	0.66 (2.83)	0.09 (1.25)	0.60 (7.68)	0.10 (2.81)	0.61 (29.89)
Backward data	0.20 (0.27)	0.00 (0.00)	0.17 (0.74)	0.00 (0.00)	0.12 (1.60)	0.00 (0.00)	0.15 (4.44)	0.00 (0.00)
Update model	0.04 (0.06)	0.01 (0.03)	0.03 (0.13)	0.01 (0.04)	0.03 (0.41)	0.01 (0.11)	0.03 (0.95)	0.01 (0.42)
Total	75.90 (100.00)	33.14 (100.00)	22.86 (100.00)	23.15 (100.00)	7.52 (100.00)	7.81 (100.00)	3.42 (100.00)	2.04 (100.00)

The results are averaged over four profiling windows at epochs 1, 5, 10 and 20. Each profiling window consists of 10 iterations measured by PyTorch Profiler. "tr" and "val" denote training and validation data. The "Estimate weights" stage is highlighted as the primary target for efficiency improvement.

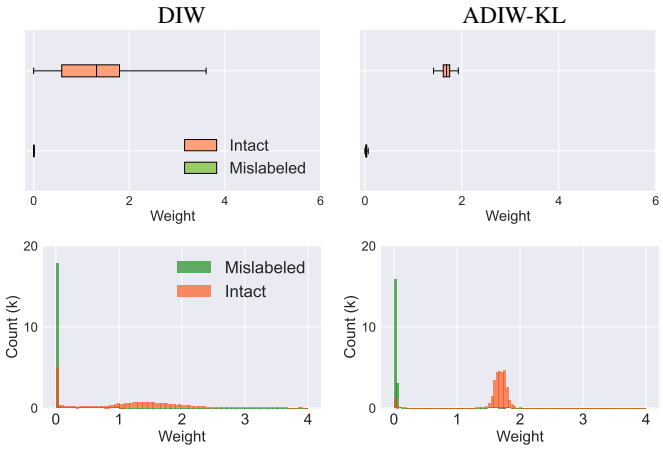


Fig. 5. Statistics of weight distributions on CIFAR-10 under 0.4 symmetric label noise (top: box plots; bottom: histogram plots).

results under 0.3 pair and 0.5 symmetric label noise reported in Table IX in Appendix F-D. On the small-scale datasets (Fashion-MNIST, CIFAR-10, and CIFAR-100), ADIW reduces per-epoch training time by approximately 20–75% compared to DIW. The efficiency gain of ADIW is even more pronounced on ImageNet-1K, where ADIW achieves over 95% reduction. These results demonstrate that ADIW scales efficiently from small to large datasets, whereas existing reweighting-based methods such as L2R, MW-Net, and DIW become less practical on the large-scale datasets.

Finally, we visualized the learned weight distributions on noisy-labeled CIFAR-10 in Figures 5, where the weights of intact samples are shown in orange and those of mislabeled samples in green. Both DIW and ADIW-KL assign near-zero weights to mislabeled data; however, ADIW-KL yields lower variance among intact-sample weights than DIW, indicating more stable weighting. Results for all ADIW variants are provided in Figure 6 in Appendix F-C.

C. Ablation Study

To gain deeper insight into the design and behavior of ADIW, we conducted several ablation studies. First, we performed fine-grained profiling of multiple implementations to trace the evolution from DIW to ADIW and quantify the resulting efficiency gains. Second, we investigated the effect of the number of validation samples used in the weight estimation. Finally, we examined the impact of adopting different data transformation choices within ADIW.

1) Key-stage Profiling from DIW to ADIW: To examine the efficiency improvements from DIW to ADIW, we profiled different implementations across key stages using the PyTorch Profiler (see Appendix E-D for configuration details). We considered the following four implementations:

- DIW: the original DIW implementation;
- DIW_Val: DIW with the validation loader defined outside the training loop as described in Section III-C;
- ADIW_CPU: PGD-based WE performed on CPU;
- ADIW_GPU: PGD-based WE performed on GPU (i.e., the proposed method).

Table IV presents stage-wise profiling results on ImageNet-1K under 0.4 symmetric label noise. For DIW, validation data fetching and WE are the main bottlenecks. DIW_Val substantially reduces the cost of validation data fetching, but WE remains the dominant contributor to both CPU and CUDA time. ADIW_CPU replaces solving the quadratic program to convergence with lightweight PGD-based WE on CPU, achieving about 72% reductions in both CPU and CUDA time, though WE remains the main bottleneck. ADIW_GPU performs PGD-based WE entirely on GPU, reducing total CPU and CUDA time by approximately 95% and 94%, respectively (compared with DIW), with WE contributing only 6% of CPU time and 2% of CUDA time.

These results demonstrate how ADIW_GPU evolves from DIW through progressively improved designs toward fully GPU-accelerated and scalable training. Additional profiling results on CIFAR-10 are reported in Appendix F-A.

2) Impact of Validation Sample Size: We conducted experiments on CIFAR-10 under 0.4 symmetric label noise with different validation set sizes (3 trials). Results in Table V show that ADIW-KL and ADIW-WA remain highly robust to the validation set size, with only minor performance variations when reducing the number of validation samples from

TABLE V
EFFECT OF VALIDATION SET SIZE ON CIFAR-10 UNDER LABEL NOISE.

Size	DIW	ADIW-KM	ADIW-KL	ADIW-LS	ADIW-WA
10	61.04 (1.10)	82.06 (1.87)	86.14 (0.40)	65.00 (1.29)	86.32 (0.22)
100	83.13 (0.56)	86.29 (0.67)	86.85 (0.29)	82.82 (1.02)	86.52 (0.08)
1000	85.38 (0.45)	86.70 (0.06)	87.05 (0.18)	84.38 (0.68)	86.27 (0.06)

TABLE VI
COMPARISON OF DATA TRANSFORMATION CHOICES IN ADIW ON CIFAR-10.

Trans.	ADIW	0.3 pair		0.4 symmetric		0.5 symmetric	
		Acc. (std)	Time	Acc. (std)	Time	Acc. (std)	Time
-F	-KM	**88.57 (0.17)**	42.21	81.30 (0.21)	42.40	75.37 (0.74)	42.36
	-KL	**87.92 (0.14)**	42.24	**85.50 (0.33)**	41.37	**81.31 (0.54)**	42.21
	-LS	**87.42 (0.27)**	42.46	81.30 (0.31)	42.42	75.94 (0.12)	42.55
	-WA	67.54 (0.40)	75.74	63.09 (0.68)	75.79	56.26 (0.55)	75.80
-L	-KM	**89.16 (0.15)**	55.04	**86.70 (0.06)**	49.71	**80.64 (0.67)**	38.73
	-KL	**88.71 (0.27)**	38.48	**87.05 (0.18)**	38.82	**83.20 (0.20)**	38.61
	-LS	**87.86 (0.13)**	69.59	**84.38 (0.68)**	67.94	**79.26 (1.03)**	68.53
	-WA	**88.20 (0.30)**	39.59	**86.27 (0.06)**	39.66	**83.41 (0.12)**	39.79

"Trans." denotes the data transformation type, where "-F" and "-L" indicate hidden-layer-output and loss-value transformations, respectively. The results are reported as mean accuracy (standard deviation) over the last ten epochs (3 trials). Best and statistically comparable methods in accuracy (t-test at the 1% significance level) are shown in bold.

1000 to 10. In contrast, DIW exhibits noticeable performance degradation under extremely small validation sets, e.g., when reducing the validation size from 100 to 10 samples. Overall, ADIW variants tend to be more robust to limited validation data, although ADIW-LS shows relatively larger degradation while still outperforming DIW.

3) Comparison of Data Transformations: As discussed in Section III-C, the data transformation in ADIW can be either hidden-layer-output transformation (-F) or loss-value transformation (-L). We compared their performance on noisy-labeled CIFAR-10 in Table VI, where the time denotes the average end-to-end training time per epoch in seconds. Overall, the -L transformation achieves strong performance across all divergence choices, providing computational efficiency and a simpler algorithmic design. These advantages make it the preferred configuration adopted in all preceding experiments.

VI. CONCLUSIONS

We analyzed the limitations of existing IW-based methods for joint distribution shift and proposed ADIW, a unified and efficient IW framework for deep learning. By formulating weight estimation as a divergence-minimization problem, ADIW supports diverse weight estimation methods in a plug-and-play manner within a single implementation while improving efficiency through projected gradient descent updates. Experiments demonstrated the effectiveness, scalability, and versatility of ADIW in handling distribution shifts.

Future work includes deeper theoretical analysis of the impact of different divergence measures within ADIW, as well as empirical investigations of how they affect performance across tasks such as reinforcement learning, time-series forecasting, and natural language processing. Additionally, future work may consider handling gradual distribution shifts, such as *concept drift* [43], which involve multiple sequential shifts rather than a single training-test distribution pair.

REFERENCES

[1] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. Lawrence, *Dataset shift in machine learning*. Cambridge, MA: The MIT Press, 2009.

[2] S. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[3] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000.

[4] M. Sugiyama, M. Krauledat, and K. Müller, "Covariate shift adaptation by importance weighted cross validation," *Journal of Machine Learning Research*, vol. 8, no. 5, pp. 985–1005, 2007.

[5] J. Byrd and Z. C. Lipton, "What is the effect of importance weighting in deep learning?" in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[6] T. Fang, N. Lu, G. Niu, and M. Sugiyama, "Rethinking importance weighting for deep learning under distribution shift," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf, and A. Smola, "Correcting sample selection bias by unlabeled data," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[8] M. Schubiger, G. Banjac, and J. Lygeros, "GPU acceleration of ADMM for large-scale quadratic programming," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 55–67, 2020.

[9] M. Sugiyama, S. Nakajima, H. Kashima, P. Buenau, and M. Kawanabe, "Direct importance estimation with model selection and its application to covariate shift adaptation," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[10] T. Kanamori, S. Hido, and M. Sugiyama, "A least-squares approach to direct importance estimation," *Journal of Machine Learning Research*, vol. 10, no. 7, pp. 1391–1445, 2009.

[11] M. Sugiyama, T. Suzuki, and T. Kanamori, "Density-ratio matching under the Bregman divergence: a unified framework of density-ratio estimation," *Annals of the Institute of Statistical Mathematics*, vol. 64, pp. 1009–1044, 2012.

[12] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 3, pp. 200–217, 1967.

[13] T. Lodkaew, T. Fang, T. Ishida, and M. Sugiyama, "Importance weighting for aligning language models under deployment distribution shift," *Transactions on Machine Learning Research*, 2025.

[14] A. Kumagai, T. Iwata, H. Takahashi, T. Nishiyama, and Y. Fujiwara, "Importance-weighted positive-unlabeled learning for distribution shift adaptation," in *Proceedings*

of the International Conference on Artificial Intelligence and Statistics (AISTATS), 2025.

[15] E. S. Levitin and B. T. Polyak, "Constrained minimization methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 6, no. 5, pp. 1–50, 1966.

[16] A. Müller, "Integral probability metrics and their generating classes of functions," *Advances in Applied Probability*, vol. 29, no. 2, pp. 429–443, 1997.

[17] S. M. Ali and S. D. Silvey, "A general class of coefficients of divergence of one distribution from another," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 28, no. 1, pp. 131–142, 1966.

[18] L. N. Vaserstein, "Markov processes over denumerable products of spaces describing large system of automata," *Problems Inform. Transmission*, vol. 5, no. 3, pp. 47–52, 1969.

[19] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International Statistical Review*, vol. 70, no. 3, pp. 419–435, 2002.

[20] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola, "Integrating structured biological data by kernel maximum mean discrepancy," *Bioinformatics*, vol. 22, no. 14, pp. e49–e57, 2006.

[21] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, 2012.

[22] B. K. Sriperumbudur, A. Gretton, K. Fukumizu, B. Schölkopf, and G. R. Lanckriet, "Hilbert space embeddings and metrics on probability measures," *Journal of Machine Learning Research*, vol. 11, no. 50, pp. 1517–1561, 2010.

[23] K. Fukumizu, A. Gretton, X. Sun, and B. Schölkopf, "Kernel measures of conditional dependence," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

[24] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[25] E. Hellinger, "Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen." *Journal für die reine und angewandte Mathematik*, vol. 136, pp. 210–271, 1909.

[26] M. Khosravifard, D. Fooladivanda, and T. A. Gulliver, "Confliction of the convexity and metric properties in f-divergences," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 90, no. 9, pp. 1848–1853, 2007.

[27] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[28] C. Villani, *Topics in optimal transportation*, ser. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 2003, vol. 58.

[29] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[30] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[31] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747v2*, 2017.

[32] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[38] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[39] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[40] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, 2018.

[41] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[42] B. Van Rooyen, A. Menon, and R. C. Williamson, "Learning with symmetric label noise: The importance of being unhinged," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[43] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[44] N. Lu, T. Zhang, T. Fang, T. Teshima, and M. Sugiyama, *Rethinking Importance Weighting for Transfer Learning*.

Cham: Springer International Publishing, 2023, pp. 185–231.

[45] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[46] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[47] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky, "Domain-adversarial training of neural networks," *Journal of Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016.

[48] 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 (CVPR)*, 2017.

[49] D.-H. Lee *et al.*, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *International Conference on Machine Learning (ICML) Workshop*, 2013.

[50] K. Saito, Y. Ushiku, and T. Harada, "Asymmetric tri-training for unsupervised domain adaptation," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[51] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM Journal on Optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[52] G. Garrigos and R. M. Gower, "Handbook of convergence theorems for (stochastic) gradient methods," *arXiv preprint arXiv:2301.11235*, 2023.

APPENDIX A
DISCUSSION OF IW AND DA APPROACHES UNDER DISTRIBUTION SHIFT

This section briefly reviews distribution shift and representative approaches for handling it, including importance weighting and domain adaptation.

Distribution Shift. Joint distribution shift is the most general form of distribution shift, where the joint distribution $p(\boldsymbol{x}, y)$ changes between the training and test data without additional assumptions, i.e., $p_{\mathrm{tr}}(\boldsymbol{x}, y) \neq p_{\mathrm{te}}(\boldsymbol{x}, y)$. It can be specialized into the following cases under further assumptions [44]:

- Covariate shift: $p(\boldsymbol{x})$ changes while $p(y \mid \boldsymbol{x})$ remains unchanged;
- Class-prior shift: $p(y)$ changes while $p(x \mid y)$ remains unchanged;
- Class-posterior shift (i.e., label noise): $p(y \mid \boldsymbol{x})$ changes while $p(\boldsymbol{x})$ remains unchanged;
- Class-conditional shift: $p(\boldsymbol{x} \mid y)$ changes while $p(y)$ is fixed.

Importance Weighting (IW). As introduced in the main text, IW and IW-based methods can address the general joint distribution shift without assuming a specific shift form a priori. They typically rely on a small labeled validation set from the test distribution to estimate importance weights for the training data. Moreover, IW-based methods assume that $p_{\mathrm{te}}(\boldsymbol{x}, y)$ is absolutely continuous with respect to $p_{\mathrm{tr}}(\boldsymbol{x}, y)$, i.e., $p_{\mathrm{te}}(\boldsymbol{x}, y) > 0$ implies $p_{\mathrm{tr}}(\boldsymbol{x}, y) > 0$. Consequently, IW methods are usually evaluated under controlled distribution shifts, where reliable estimation of importance weights is feasible.

Domain Adaptation (DA). DA takes a different approach from IW and can be broadly categorized into supervised DA (SDA) and unsupervised DA (UDA) [2, 45]. In SDA, a small amount of labeled target-domain data is available, whereas UDA assumes access to abundant unlabeled target-domain data. While SDA is closer to the setting of IW, much of the DA literature focuses on UDA, where the objective is to reduce domain discrepancy through representation alignment [46–48] or pseudo-label-based transfer [49, 50]. These approaches typically rely on implicit assumptions, such as limited change in the conditional distribution, and therefore address only a restricted subset of joint distribution shift. When such assumptions are violated, e.g., under severe label noise, DA methods often become less effective. Overall, DA is primarily designed for more challenging cross-domain shifts with substantial domain discrepancy.

In summary, IW and DA tackle distribution shift under fundamentally different assumptions, problem setups, and objectives. IW focuses on addressing general joint distribution shift by reweighting training data using a small labeled validation set under controlled shifts, whereas DA typically addresses a restricted subset of joint shift using abundant unlabeled target-domain data to learn transferable or domain-invariant representations for large cross-domain discrepancies.

APPENDIX B
ASSUMPTIONS OF THEOREM III.1

Here we list the necessary technical assumptions for Theorem III.1.

Assumption B.1 (Lipschitz continuous gradient)**.** The learning objective $R_{\mathrm{tr}}(\theta, w)$ is twice differentiable and has an L-Lipschitz continuous gradient for all w, i.e.,

$$-LI \preceq \nabla^2_\theta R_{\mathrm{tr}}(\theta, w) \preceq LI,$$

where I is the identity matrix.

This assumption does not require convexity of the learning objective $R_{\mathrm{tr}}(\theta, w)$; it only ensures that the gradient does not change arbitrarily fast.

Assumption B.2 (Unbiased stochastic gradients with bounded variance)**.** Let $\tilde{\nabla}_\theta$ be the noisy gradient given by the gradient generator and we assume it satisfies:

$$\mathbb{E}[\tilde{\nabla}_\theta R_{\mathrm{tr}}(\theta, w)] = \mathbb{E}[\nabla_\theta R_{\mathrm{tr}}(\theta, w)],\ \mathbb{E}_{\tilde{\nabla}_\theta}\left[\left\|\tilde{\nabla}_\theta R_{\mathrm{tr}}(\theta, w) - \nabla_\theta R_{\mathrm{tr}}(\theta, w)\right\|^2\right] \leq \sigma^2$$

for some constant σ^2 and all w, where $\|\cdot\|$ denotes the L_2 norm.

This is a standard assumption in stochastic optimization [51, 52].

Assumption B.3. We assume the WE subroutine satisfies two stability conditions:

(W1) (*Nonexpansive gradient projection*) Let Π_Q be the Euclidean projection onto a closed convex set Q, then

$$\|\Pi_Q(u) - \Pi_Q(v)\|_1 \leq \|u - v\|_1,\ \forall\ u, v \in \mathbb{R}^n,$$

where $\|\cdot\|_1$ denotes the L_1 norm.

(W2) (*Learning-rate adjustment*) For each outer iteration s and all inner iterations $t = 0, \ldots, T$,

$$\left\|\eta_t \tilde{\nabla}_{\mathcal{W}} \widehat{J}(\mathcal{W}_{s,t}; \theta_s)\right\|_1 \leq \left\|\alpha_s \tilde{\nabla}_\theta R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)\right\|^2,$$

where $\mathcal{W}_{s,t}$ denotes the weight vector at inner step t and outer step s, $\mathcal{W}_s = \mathcal{W}_{s-1,T}$, and η_t and α_s are the inner-loop and outer-loop learning rates, respectively.

W1 ensures that the projection operation does not increase the magnitude of the weight updates, ensuring numerical stability. W2 enforces proportional scaling between the inner- and outer-loop updates, ensuring that the cumulative weight updates, measured in L_1 norm to capture the total effect across all samples, remain controlled relative to the classifier gradient, measured in squared L_2 norm to capture its total energy, while still allowing sufficient flexibility for WE.

Appendix C
Proof of Theorem III.1

Proof. We analyze the ADIW algorithm with inner weight updates

$$\mathcal{W}_{s,t} := (w_{i,s,t} := w(\boldsymbol{z}_{i,s,t}^{\mathrm{tr}}))_{i=1}^{n_{\mathrm{tr}}} \in \mathbb{R}^{n_{\mathrm{tr}}},\ t = 0, \cdots, T,$$

and outer classifier updates. At each outer iteration s, the classifier parameters θ are updated by

$$\theta_{s+1} = \theta_s - \alpha_s \tilde{\nabla}_{\boldsymbol{\theta}} R_{\mathrm{tr}}(\theta_s, \mathcal{W}_{s,0}),\ s = 0, \cdots, S, \tag{17}$$

where $\alpha_s > 0$ is the outer learning rate. During the outer iteration s, at each inner iteration t, the weights are updated by projected gradient descent (PGD):

$$\mathcal{W}_{s,t+1} \ = \ \Pi_Q\Big(\mathcal{W}_{s,t} - \eta_t \, \tilde{\nabla}_{\mathcal{W}} \widehat{J}(\mathcal{W}_{s,t}; \theta_s)\Big), \tag{18}$$

where $\eta_t > 0$ is the inner learning rate and Π_Q denotes projection onto Q.

a) Bounding weight changes.: Fix an outer iteration s. Given that the inner PGD runs T steps,

$$\mathcal{W}_{s,0} \xrightarrow{\text{PGD step in Eq. (18)}} \mathcal{W}_{s,1} \ \to \ \cdots \ \to \ \mathcal{W}_{s,T},$$

we have the total weight change across the inner loop

$$\mathcal{W}_{s,T} - \mathcal{W}_{s,0} = \sum_{t=0}^{T-1} \big(\mathcal{W}_{s,t+1} - \mathcal{W}_{s,t}\big). \tag{19}$$

Using the triangle inequality, we have

$$\|\mathcal{W}_{s,T} - \mathcal{W}_{s,0}\|_1 \le \sum_{t=0}^{T-1} \|\mathcal{W}_{s,t+1} - \mathcal{W}_{s,t}\|_1. \tag{20}$$

For each inner iteration,

$$\begin{aligned}
\|\mathcal{W}_{s,t+1} - \mathcal{W}_{s,t}\|_1 &= \big\|\Pi_Q(\mathcal{W}_{s,t} - \eta_t \tilde{\nabla}_{\mathcal{W}} \widehat{J}(\mathcal{W}_{s,t}; \theta_s)) - \Pi_Q(\mathcal{W}_{s,t})\big\|_1 \\
&\le \big\|(\mathcal{W}_{s,t} - \eta_t \tilde{\nabla}_{\mathcal{W}} \widehat{J}(\mathcal{W}_{s,t}; \theta_s)) - \mathcal{W}_{s,t}\big\|_1 \\
&= \ \big\|\eta_t \tilde{\nabla}_{\mathcal{W}} \widehat{J}(\mathcal{W}_{s,t}; \theta_s)\big\|_1 \\
&\le \alpha_s^2 \|\tilde{\nabla}_{\boldsymbol{\theta}} R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)\|^2,
\end{aligned} \tag{21}$$

where the first equation is due to Eq. (18) and the fact that at the current iteration, $\mathcal{W}_{s,t}$ lies in Q and thus $\mathcal{W}_{s,t} = \Pi_Q(\mathcal{W}_{s,t})$; the second inequality is due to the nonexpansiveness of Π_Q (Assumption W1); and the last inequality is due to the stability of weight gradient (Assumption W2). Then summing over $t = 0, \ldots, T-1$ yields

$$\|\mathcal{W}_{s,T} - \mathcal{W}_{s,0}\|_1 \le T\alpha_s^2 \, \|\tilde{\nabla}_s\|^2, \tag{22}$$

where $\tilde{\nabla}_s$ denotes $\tilde{\nabla}_{\boldsymbol{\theta}} R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)$ Denote $\mathcal{W}_s := \mathcal{W}_{s-1,T}$, the total weight change across the outer loop satisfies

$$\begin{aligned}
\mathcal{W}_{s+1} - \mathcal{W}_s &= \mathcal{W}_{s,T} - \mathcal{W}_{s-1,T} \\
&= (\mathcal{W}_{s,T} - \mathcal{W}_{s,0}) + (\mathcal{W}_{s,0} - \mathcal{W}_{s-1,T}).
\end{aligned} \tag{23}$$

We note that the second term of Eq. (23) is 0 as $\mathcal{W}_{s,0}$ is the starting point for the inner loop at outer iteration s, which is inherited from the last step $\mathcal{W}_{s-1,T}$. Combining the above results leads to

$$\begin{aligned}
\|\mathcal{W}_{s+1} - \mathcal{W}_s\|_1 &\le \|\mathcal{W}_{s,T} - \mathcal{W}_{s,0}\|_1 + \|\mathcal{W}_{s,0} - \mathcal{W}_{s-1,T}\|_1 \\
&\le T\alpha_s^2 \, \|\tilde{\nabla}_s\|^2.
\end{aligned} \tag{24}$$

b) Bounding risk decrease.: By *Taylor's theorem*, there exists θ' on the line segment between θ_s and θ_{s+1} such that

$$\begin{aligned} R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_s) &= R_{\mathrm{tr}}\left(\theta_s - \alpha_s \tilde{\nabla}_s, \mathcal{W}_s\right) \\ &= R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \alpha_s \tilde{\nabla}_s^T \nabla_\theta R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) + \frac{\alpha_s^2}{2} \tilde{\nabla}_s^T \nabla_\theta^2 R_{\mathrm{tr}}(\theta', \mathcal{W}_s) \tilde{\nabla}_s \\ &\leq R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \alpha_s \tilde{\nabla}_s^T \nabla_\theta R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) + \frac{\alpha_s^2 L}{2} \left\| \tilde{\nabla}_s \right\|^2, \end{aligned} \tag{25}$$

where the last inequality is due to the Lipschitz continuous gradient (Assumption B.1). Changing the weights from $\mathcal{W}_s$ to $\mathcal{W}_{s+1}$ gives

$$\begin{aligned} R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1}) - R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_s) &= \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} w_{i,s+1} \ell(\boldsymbol{f}_{\theta_{s+1}}(\boldsymbol{x}_i), y_i) - \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} w_{i,s} \ell(\boldsymbol{f}_{\theta_{s+1}}(\boldsymbol{x}_i), y_i) \\ &= \frac{1}{n_{\mathrm{tr}}} \sum_{i=1}^{n_{\mathrm{tr}}} \left[(w_{i,s+1} - w_{i,s}) \ell(\boldsymbol{f}_{\theta_{s+1}}(\boldsymbol{x}_i), y_i) \right] \\ &\leq \frac{M}{n_{\mathrm{tr}}} \|\mathcal{W}_{s+1} - \mathcal{W}_s\|_1 \\ &\leq \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \|\tilde{\nabla}_s\|^2, \end{aligned} \tag{26}$$

where the first inequality applies the upper bound of the loss, and the last inequality is due to Eq. (24). Combining Eq. (25) and (26) gives

$$R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1}) \leq R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \alpha_s \tilde{\nabla}_s^T \nabla_s + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \left\| \tilde{\nabla}_s \right\|^2, \tag{27}$$

where ∇_s denotes $\nabla_\theta R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)$.

c) Convergence via telescoping sum.: Taking the expected value gives us

$$\begin{aligned} \mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1}) | \theta_s] &\leq R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \alpha_s \mathbb{E}\left[\tilde{\nabla}_s^T \nabla_s \middle| \theta_s \right] + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \mathbb{E}\left[\left\| \tilde{\nabla}_s \right\|^2 \middle| \theta_s \right] \\ &= R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \alpha_s \|\nabla_s\|^2 + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \mathbb{E}\left[\left\| \tilde{\nabla}_s \right\|^2 \middle| \theta_s \right], \end{aligned} \tag{28}$$

where the equality is due to the unbiased stochastic gradients (Assumption B.2). Since

$$\begin{aligned} \mathbb{E}\left[\left\| \tilde{\nabla}_s \right\|^2 \middle| \theta_s \right] &= \mathbb{E}\left[\left\| \tilde{\nabla}_s - \nabla_s + \nabla_s \right\|^2 \middle| \theta_s \right] \\ &= \mathbb{E}\left[\left\| \tilde{\nabla}_s - \nabla_s \right\|^2 + \|\nabla_s\|^2 + 2 \left(\tilde{\nabla}_s - \nabla_s \right)^T \nabla_s \middle| \theta_s \right] \\ &\leq \sigma^2 + \|\nabla_s\|^2, \end{aligned} \tag{29}$$

where the last inequality comes from the unbiased stochastic gradients and bounded variance (Assumption B.2), then we have

$$\mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1}) | \theta_s] \leq R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \left(\alpha_s - \frac{\alpha_s^2 L}{2} - \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \|\nabla_s\|^2 + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \sigma^2. \tag{30}$$

If we set α_s small enough such that $\alpha_s < \frac{n_{\mathrm{tr}}}{2MT + nL}$ for all s, this simplifies to

$$\mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1}) | \theta_s] \leq R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s) - \frac{\alpha_s}{2} \|\nabla_s\|^2 + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \sigma^2. \tag{31}$$

Now taking the full expectation,

$$\mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1})] \leq \mathbb{E}[R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)] - \frac{\alpha_s}{2} \mathbb{E}\left[\|\nabla_s\|^2 \right] + \left(\frac{\alpha_s^2 L}{2} + \frac{MT\alpha_s^2}{n_{\mathrm{tr}}} \right) \sigma^2, \tag{32}$$

and then rearranging the terms,

$$\frac{1}{2} \mathbb{E}\left[\|\nabla_s\|^2 \right] \leq \frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_s, \mathcal{W}_s)] - \mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1}, \mathcal{W}_{s+1})]}{\alpha_s} + \left(\frac{\alpha_s L}{2} + \frac{MT\alpha_s}{n_{\mathrm{tr}}} \right) \sigma^2. \tag{33}$$

Next summing up Eq. (33) from $s = 1$ to S,

$$\begin{aligned}
\frac{1}{2}\sum_{s=1}^{S}\mathbb{E}\left[\|\nabla_s\|^2\right] &\leq \sum_{s=1}^{S}\frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_s,\mathcal{W}_s)]}{\alpha_s} - \sum_{s=1}^{S}\frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_{s+1},\mathcal{W}_{s+1})]}{\alpha_s} + \left(\frac{L}{2}+\frac{MT}{n_{\mathrm{tr}}}\right)\sigma^2\sum_{s=1}^{S}\alpha_s \\
&= \sum_{s=1}^{S}\frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_s,\mathcal{W}_s)]}{\alpha_s} - \sum_{s=2}^{S+1}\frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_s,\mathcal{W}_s)]}{\alpha_{s-1}} + \left(\frac{L}{2}+\frac{MT}{n_{\mathrm{tr}}}\right)\sigma^2\sum_{s=1}^{S}\alpha_s \\
&= \sum_{s=2}^{T}\left(\frac{1}{\alpha_s}-\frac{1}{\alpha_{s-1}}\right)\mathbb{E}[R_{\mathrm{tr}}(\theta_s,\mathcal{W}_s)] + \frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_1,\mathcal{W}_1)]}{\alpha_1} \\
&\quad - \frac{\mathbb{E}[R_{\mathrm{tr}}(\theta_{S+1},\mathcal{W}_{S+1})]}{\alpha_S} + \left(\frac{L}{2}+\frac{MT}{n_{\mathrm{tr}}}\right)\sigma^2\sum_{s=1}^{S}\alpha_s \\
&\leq \sum_{s=2}^{S}\left(\frac{1}{\alpha_s}-\frac{1}{\alpha_{s-1}}\right)BM + \frac{BM}{\alpha_1} + \left(\frac{L}{2}+\frac{MT}{n_{\mathrm{tr}}}\right)\sigma^2\sum_{s=1}^{S}\alpha_s \\
&= \frac{BM}{\alpha_S} + \left(\frac{L}{2}+\frac{MT}{n_{\mathrm{tr}}}\right)\sigma^2\sum_{s=1}^{S}\alpha_s, \qquad (34)
\end{aligned}$$

where the last inequality applies the upper bound of the weights and loss. By setting $\alpha_s = \frac{c}{\sqrt{s}}$ (with $\frac{1}{\alpha_0} \triangleq 0$), we obtain

$$\begin{aligned}
\frac{1}{2}\sum_{s=1}^{S}\mathbb{E}\left[\|\nabla_s\|^2\right] &\leq \frac{BM\sqrt{S}}{c} + \left(\frac{L}{2}\sigma^2+\frac{MT\sigma^2}{n_{\mathrm{tr}}}\right)2c\sqrt{S} \\
&= \left(\frac{BM}{c} + cL\sigma^2 + \frac{2cMT\sigma^2}{n_{\mathrm{tr}}}\right)\sqrt{S}, \qquad (35)
\end{aligned}$$

where the inequality follows since $\sum_{s=1}^{S}\frac{1}{\sqrt{S}} \leq 2\sqrt{S}$. Therefore, let $m_S = \theta_s$ with probability $\frac{1}{S}$ for all $s \in \{1,\ldots,S\}$, and the expected value of gradient at this point is

$$\begin{aligned}
\mathbb{E}[\|\nabla R_{\mathrm{tr}}(m_S,\mathcal{W}_s)\|^2] &= \sum_{s=1}^{S}\mathbb{E}\left[\|\nabla_s\|^2\right]\cdot\mathbf{P}\left(m_S=\theta_s\right) \\
&= \frac{1}{S}\sum_{s=1}^{S}\mathbb{E}\left[\|\nabla_s\|^2\right]. \qquad (36)
\end{aligned}$$

Substituting Eq. (35) into this gives us

$$\mathbb{E}[\|\nabla R_{\mathrm{tr}}(m_S,\mathcal{W}_s)\|^2] \leq \frac{2}{\sqrt{S}}\left(\frac{BM}{c} + cL\sigma^2 + \frac{2cMT\sigma^2}{n_{\mathrm{tr}}}\right), \qquad (37)$$

which concludes the proof. □

Appendix D
ADIW with Hidden-Layer-Output Transformation

The algorithm of ADIW with the hidden-layer-output transformation is presented in Algorithm 2. Compared with Algorithm 1, it involves substantially more steps, as it requires partitioning the training and validation data by class within each mini-batch and performing weight estimation in a class-wise manner. This results in fewer samples per estimation, potentially causing unstable weight estimation and requiring additional handling for cases with insufficient samples. Furthermore, a class-prior ratio is needed to be estimated before training and applied to the computed weights. Therefore, we recommend using the loss-value transformation as the default configuration in ADIW, as in Algorithm 1.

Appendix E
Supplementary Experimental Setups

This section introduces additional experimental details, including descriptions of the datasets and base models used in the main paper, and the specific hyperparameter settings for the class-prior shift, label-noise, and ablation study experiments.

Since computational efficiency is a key aspect of our contribution, we evaluated the methods not only in terms of predictive performance but also efficiency. Specifically, we assessed the computational efficiency at two levels. First, we measured the average end-to-end training time per epoch using `time.perf_counter()` from Python's `time` module, with `torch.cuda.synchronize()` invoked beforehand to ensure all GPU operations had completed. Second, we provided a

Algorithm 2 Accelerated dynamic importance weighting (with hidden-layer-output transformation, in a mini-batch).

Require: a training and validation minibatch $\mathcal{S}^{\mathrm{tr}}$ and $\mathcal{S}^{\mathrm{v}}$ with index set $\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}$ and $\mathcal{I}_{\mathcal{S}^{\mathrm{v}}}$, current model $\boldsymbol{f}_\theta$, weight vector $\mathcal{W}$, WE objective $\widehat{J}$ with constraint set Q, step size η_t, number of WE iterations T, class-prior ratio $\{r_y^*\}_{y=1}^C$

`forward` the input parts of $\mathcal{S}^{\mathrm{tr}}$ & $\mathcal{S}^{\mathrm{v}}$
`retrieve` the hidden-layer outputs as $\mathcal{Z}^{\mathrm{tr}}$ & $\mathcal{Z}^{\mathrm{v}}$
`apply` L2 normalization to $\mathcal{Z}^{\mathrm{tr}}$ & $\mathcal{Z}^{\mathrm{v}}$: $\mathcal{Z}^{\mathrm{tr}} \leftarrow \mathrm{L2}(\mathcal{Z}^{\mathrm{tr}})$, $\mathcal{Z}^{\mathrm{v}} \leftarrow \mathrm{L2}(\mathcal{Z}^{\mathrm{v}})$
`partition` $\mathcal{Z}^{\mathrm{tr}}$ & $\mathcal{Z}^{\mathrm{v}}$ into $\{\mathcal{Z}_y^{\mathrm{tr}}\}_{y=1}^C$ & $\{\mathcal{Z}_y^{\mathrm{v}}\}_{y=1}^C$
`partition` $\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}$ & $\mathcal{I}_{\mathcal{S}^{\mathrm{v}}}$ into $\{\mathcal{I}_{\mathcal{S}^{\mathrm{tr}},y}\}_{y=1}^C$ & $\{\mathcal{I}_{\mathcal{S}^{\mathrm{v}},y}\}_{y=1}^C$
for y = 1 to C **do**
  `retrieve` initial weights for class y: $\mathcal{W}_1^y \leftarrow \mathcal{W}[\mathcal{I}_{\mathcal{S}^{\mathrm{tr}},y}]$
  if $|\mathcal{Z}_y^{\mathrm{tr}}| < 2$ **or** $|\mathcal{Z}_y^{\mathrm{v}}| < 2$ **then**
    `skip` class y; retain original weights `(insufficient samples)`
    continue
  end if
  for t = 1 to T **do**
    `compute` gradients $\nabla_{\mathcal{W}_t^y}\widehat{J}$ from $\mathcal{Z}_y^{\mathrm{tr}}$ & $\mathcal{Z}_y^{\mathrm{v}}$
    `update` $\mathcal{W}_{t+1}^y \leftarrow \mathcal{W}_t^y - \eta_t \nabla_{\mathcal{W}_t^y}\widehat{J}$
    `project` $\mathcal{W}_{t+1}^y$ onto Q
  end for
  `multiply` all $w_i \in \mathcal{W}_T^y$ by r_y^*
end for
if `all classes skipped` **then**
  `set` $\mathcal{W}_T \leftarrow \mathbf{0}$ for $\mathcal{S}^{\mathrm{tr}}$ `(fallback)`
else
  `concatenate` all class-wise indices $\{\mathcal{I}_{\mathcal{S}^{\mathrm{tr}},y}\}_{y=1}^C$ and weights $\{\mathcal{W}_T^y\}_{y=1}^C$
  `update` weights in $\mathcal{W}$: $\mathcal{W}[\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}^{\mathrm{cat}}] \leftarrow \mathcal{W}_T^{\mathrm{cat}}$
  `reorder` $\mathcal{W}_T^{\mathrm{cat}}$ to match $\mathcal{I}_{\mathcal{S}^{\mathrm{tr}}}$, yielding $\mathcal{W}_T$
end if
`weight` the empirical risk $\widehat{R}(\boldsymbol{f}_\theta)$ by $\mathcal{W}_T$
`backward` $\widehat{R}(\boldsymbol{f}_\theta)$ and `update` θ

fine-grained breakdown of CPU and GPU runtimes across key algorithmic stages, presented as an ablation study in Section V-C1 and in Appendix E-D. Experiments on ImageNet-1K were run on NVIDIA A100 GPUs, while others were run on NVIDIA Tesla V100 GPUs. All experiments were implemented using PyTorch 2.5.0.

A. Datasets and Base Models

Datasets. The detailed information of the datasets used in the paper is listed below:

- *Fashion-MNIST* [31] is a benchmark dataset for the 10-class image classification task. It consists of 70,000 grayscale images of fashion products across 10 categories, each with a resolution of 28*28 pixels. The dataset is divided into 60,000 training and 10,000 test samples. See https://github.com/zalandoresearch/fashion-mnist for details.
- *CIFAR-10* and *CIFAR-100* [32] are benchmark datasets for image classification of real-world objects. Each dataset consists of 60,000 color images of size 32*32 pixels, split into 50,000 for training and 10,000 for testing. CIFAR-10 contains 10 object categories, while CIFAR-100 has 100 fine-grained classes. See https://www.cs.toronto.edu/~kriz/cifar.html for details.
- *ImageNet-1K*, short for the *ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)* [33], is a large-scale image dataset of real-world objects grouped into 1000 categories. It contains 1,281,167 training data, 50,000 validation data, and 100,000 test data. Each image is a high-resolution color photograph of real-world scenes, providing rich visual diversity and fine-grained semantic coverage. See https://www.image-net.org for details.

Base Models. In this work, we adopted LeNet-5 [34] as the base model for Fashion-MNIST, whose architecture is as follows:

0th (input) layer: (32*32)-
1st to 2nd layer: C(5*5,6)-S(2*2)-
3rd to 4th layer: C(5*5,16)-S(2*2)-
5th layer: FC(120)-
6th layer: FC(84)-10,

where C(5*5,6) denotes a convolutional layer with 6 filters of size 5*5 followed by a ReLU activation, S(2*2) represents a max-pooling layer with a 2*2 filter and stride 2, and FC(120) indicates a fully connected layer with 120 output units.

The model used for ImageNet-1K was ResNet-18 [36], a residual convolutional neural network consisting of 18 layers organized into four residual stages:

0th (input) layer: (224*224*3)-
1st to 5th layers: C(7*7, 64)-S(3*3)-[C(3*3, 64), C(3*3, 64)]*2-
6th to 9th layers: [C(3*3, 128), C(3*3, 128)]*2-
10th to 13th layers: [C(3*3, 256), C(3*3, 256)]*2-
14th to 17th layers: [C(3*3, 512), C(3*3, 512)]*2-
18th layer: Global Average Pooling-FC(1000),

where C(7*7, 64) denotes a convolution with 64 filters of size 7*7 followed by batch normalization and ReLU activation, S(3*3) represents a max-pooling layer with a 3*3 filter and stride 2, [·, ·]*2 indicates two consecutive residual blocks, each consisting of two 3*3 convolutions followed by batch normalization, with ReLU applied after the first convolution and after the residual addition, and FC(1000) denotes a fully connected layer with 1000 output units.

The model used for CIFAR-10 and CIFAR-100 was a CIFAR-style ResNet-18 variant. Unlike the ImageNet-1K version, it replaces the initial 7*7 convolution and max-pooling layers with a single 3*3 convolution of stride 1. The remaining structure follows the same four residual stages and concludes with a global average pooling layer followed by a fully connected layer producing 10 or 100 outputs for classification.

B. Class-prior-shift Experiments

Here we present the hyperparameters used in the class-prior-shift experiments. For class-prior-shift experiments, the initial learning rate was 0.1 and decayed by a factor of 0.1 every 100 epochs, for a total of 400 epochs. The learning rates for weight estimation were 0.0001 for ADIW-KM, 0.005 for ADIW-KL, 0.01 for ADIW-LS, and 0.0005 for ADIW-WA. The kernel width was 0.1 for DIW and ADIW-KM, and was 5 for ADIW-KL, and 1 for ADIW-LS.

Across all experiments, including the class-prior-shift and the label-noise experiments, the batch size was fixed at 256, and the number of weight estimation iterations was set to 1. For ADIW-LS, the regularization parameter λ was set to 1e-05. For ADIW-WA, the gradient penalty coefficient κ was set to 10. The critic Φ_ν was trained using the Adam optimizer with a learning rate of 1e-04. The first 50 minibatches were used for pre-training, and each minibatch was used for 3 critic updates.

C. Label-noise Experiments

In the Fashion-MNIST experiments, the initial learning rate was set to 0.1 and decayed by a factor of 0.1 every 100 epochs, for a total of 400 epochs. The weight decay was fixed at 1e-07. The learning rates for weight estimation were 0.1 for ADIW-KM, 0.5 for ADIW-KL, 0.01 for ADIW-LS, and 0.01 for ADIW-WA.

In the CIFAR-10/100 experiments, the initial learning rate was 0.0005, decaying by a factor of 0.1 every 100 epochs, for a total of 400 epochs. The weight decay was set to 0.0001 for CIFAR-10 and 0.0005 for CIFAR-100. For CIFAR-10, the weight estimation learning rates were 0.1 for ADIW-KM, ADIW-KL, and ADIW-LS, and 0.01 for ADIW-WA. For CIFAR-100, the rates were 0.5 for ADIW-KM, 0.1 for ADIW-KL, 0.1 for ADIW-LS, and 0.005 for ADIW-WA.

In the ImageNet-1K experiments, the initial learning rate was set to 0.0002 and decayed by a factor of 0.1 every 100 epochs, for a total of 300 epochs. The weight decay was 1e-05. The learning rates for weight estimation were 0.5 for ADIW-KM, ADIW-KL, and ADIW-LS, and 0.003 for ADIW-WA.

For all experiments under the label noise, the kernel width was 0.01 for DIW, ADIW-KM, and ADIW-LS, while for ADIW-KL it was set to 0.1 on CIFAR-100 and 0.5 on the other datasets. Other hyperparameters shared across both the class-prior shift and label-noise experiments are provided in Appendix E-B.

D. Ablation Study

First we present the configuration of the PyTorch Profiler [3]. The profiler was configured with a window size of 10 iterations and activated during the selected epochs. In each activated epoch, the profiler skipped the first 10 iterations as a warm-up, used the following 2 iterations for stabilization, and then recorded results over the subsequent 10 iterations. The reported results were averaged across all profiling windows.

Then, we provide the hyperparameter configurations used in the CIFAR-10 experiments under 0.4 symmetric label noise for ADIW with the hidden-layer-output transformation, i.e., the ADIW-F variants. The kernel width was set to 0.5 for ADIW-KM-F, and 0.3 for both ADIW-KL-F and ADIW-LS-F. The regularization coefficient λ for ADIW-LS-F was 0.001. All other setups were identical to those used in the ADIW-L experiments (see the configurations for CIFAR-10 in Appendix E-C).

[3]See https://pytorch.org/docs/stable/profiler.html for PyTorch Profiler.

TABLE VII
STAGE-WISE BREAKDOWN OF TOTAL CPU AND CUDA TIME ON CIFAR-10 (SECONDS; PERCENTAGES IN PARENTHESES).

Stage	DIW		DIW_Val		ADIW_CPU		ADIW_GPU	
	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)	CPU (%)	CUDA (%)
Fetch tr data	1.37 (21.78)	0.01 (0.23)	1.44 (21.78)	0.01 (0.22)	1.14 (19.16)	0.01 (0.24)	1.60 (47.31)	0.01 (1.04)
Fetch val data	1.26 (20.07)	0.01 (0.23)	1.36 (20.58)	0.02 (0.39)	1.08 (18.07)	0.01 (0.23)	1.46 (43.34)	0.01 (1.00)
Forward tr data	0.05 (0.75)	0.34 (7.88)	0.04 (0.68)	0.33 (7.59)	0.04 (0.67)	0.34 (7.61)	0.05 (1.45)	0.34 (31.83)
Forward val data	0.05 (0.87)	0.34 (8.02)	0.05 (0.82)	0.33 (7.56)	0.04 (0.67)	0.33 (7.48)	0.06 (1.65)	0.34 (31.75)
Get tr loss	0.00 (0.06)	0.00 (0.01)	0.01 (0.08)	0.00 (0.01)	0.00 (0.06)	0.00 (0.01)	0.00 (0.11)	0.00 (0.02)
Get val loss	0.00 (0.06)	0.00 (0.01)	0.00 (0.06)	0.00 (0.01)	0.00 (0.06)	0.00 (0.01)	0.01 (0.18)	0.00 (0.04)
Estimate weights	3.33 (53.04)	3.21 (75.35)	3.47 (52.54)	3.36 (76.18)	3.49 (58.60)	3.37 (76.34)	0.02 (0.47)	0.01 (0.48)
Weight tr loss	0.06 (0.97)	0.34 (7.97)	0.08 (1.14)	0.34 (7.75)	0.06 (0.95)	0.34 (7.79)	0.07 (2.08)	0.34 (32.58)
Backward data	0.13 (2.00)	0.00 (0.00)	0.13 (1.93)	0.00 (0.00)	0.08 (1.35)	0.00 (0.00)	0.09 (2.71)	0.00 (0.01)
Update model	0.02 (0.39)	0.01 (0.31)	0.03 (0.39)	0.01 (0.30)	0.02 (0.39)	0.01 (0.30)	0.02 (0.71)	0.01 (1.25)
Total	6.28 (100.00)	4.26 (100.00)	6.61 (100.00)	4.41 (100.00)	5.96 (100.00)	4.41 (100.00)	3.38 (100.00)	1.06 (100.00)

The results are averaged over 5 profiling windows at epochs 20, 105, 205, 305 and 390. Each profiling window consists of 10 iterations measured by PyTorch Profiler. "tr" and "val" denote training and validation data. The "Estimate weights" stage is highlighted as the primary target for efficiency improvement.

TABLE VIII
MEAN ACCURACY (STANDARD DEVIATION) IN PERCENTAGE ON FASHION-MNIST, CIFAR-10/-100 OVER THE LAST TEN EPOCHS UNDER LABEL NOISE (3 TRIALS).

Data	Noise	Val-Only	Uniform	Random	L2R	MW-Net	DIW	ADIW-KM	ADIW-KL	ADIW-LS	ADIW-WA
F-MNIST	0.3 p	79.29	65.11	71.15	**88.74**	**86.56**	**89.99**	**88.73**	**89.57**	**90.00**	**87.21**
		(0.86)	(0.23)	(0.53)	**(0.47)**	**(1.79)**	**(0.17)**	**(0.66)**	**(0.18)**	**(0.11)**	**(0.36)**
	0.4 s	79.29	62.88	73.49	84.85	**85.69**	**89.64**	**88.94**	**89.19**	**89.39**	**89.06**
		(0.86)	(0.61)	(0.60)	(0.13)	**(1.03)**	**(0.04)**	**(0.14)**	**(0.11)**	**(0.09)**	**(0.35)**
	0.5 s	79.29	55.43	68.26	**83.91**	82.87	**89.05**	**87.66**	**88.80**	**88.55**	**87.60**
		(0.86)	(0.58)	(1.12)	**(0.66)**	(0.73)	**(0.28)**	**(0.49)**	**(0.25)**	**(0.16)**	**(0.05)**
CIFAR-10	0.3 p	48.09	67.84	**88.06**	**89.45**	79.31	88.00	**89.16**	**88.71**	87.86	**88.20**
		(0.27)	(0.55)	**(0.28)**	**(0.08)**	(0.76)	(0.14)	**(0.15)**	**(0.27)**	(0.13)	**(0.30)**
	0.4 s	48.09	62.92	82.74	82.39	**68.93**	**85.38**	**86.70**	**87.05**	**84.38**	**86.27**
		(0.27)	(0.34)	(0.21)	(0.50)	**(5.43)**	**(0.45)**	**(0.06)**	**(0.18)**	**(0.68)**	**(0.06)**
	0.5 s	48.09	57.09	79.07	78.61	62.65	81.31	**80.64**	**83.20**	**79.26**	**83.41**
		(0.27)	(1.30)	(0.09)	(0.35)	(1.55)	(0.30)	**(0.67)**	**(0.20)**	**(1.03)**	**(0.12)**
CIFAR-100	0.3 p	12.18	50.34	55.75	**61.42**	50.32	61.93	**63.88**	**66.39**	52.54	57.19
		(0.28)	(0.21)	(0.29)	**(1.27)**	(1.09)	(0.44)	**(0.63)**	**(0.15)**	(0.35)	(0.29)
	0.4 s	12.18	38.65	53.63	56.05	38.36	61.39	**62.26**	**63.42**	59.24	**62.38**
		(0.28)	(0.21)	(0.90)	(0.57)	(0.13)	(0.28)	**(0.30)**	**(0.43)**	(0.59)	**(0.21)**
	0.5 s	12.18	31.06	48.50	52.00	30.39	**57.35**	**56.02**	**59.09**	52.29	**57.13**
		(0.28)	(0.27)	(0.43)	(0.89)	(0.36)	**(0.37)**	**(0.62)**	**(0.40)**	(0.62)	**(0.42)**

Best and comparable methods (paired t-test at significance level 1%) are highlighted in bold. "p" and "s" denote pair and symmetric label noise, respectively.

APPENDIX F
SUPPLEMENTARY EXPERIMENTAL RESULTS

In this section, we present additional experimental results, including a profiling analysis on CIFAR-10, a summary of the classification accuracy of the compared methods, statistical plots of the importance weight distributions, and a summary of the end-to-end training time.

A. Key-stage Profiling on CIFAR-10

Table VII reports stage-wise profiling results on CIFAR-10 under 0.4 symmetric label noise. Unlike the ImageNet-1K case in the main paper, DIW, DIW_Val, and ADIW_CPU exhibited similar runtimes, likely because CIFAR-10 was small enough to fit entirely in memory, making CPU-GPU transfers and solver overhead less significant. Nevertheless, the proposed method (i.e., ADIW_GPU) achieved a substantial efficiency gain, reducing CPU time from 6.28 s to 3.38 s (about 46% reduction) and CUDA time from 4.26 s to 1.06 s (about 75% reduction) compared with DIW, with weight estimation contributing less than 0.5% on both. These results indicated that ADIW's efficiency gains were most significant at large scale but still offered clear advantages on smaller datasets.

B. Summary of Classification Accuracy

Table VIII further summarizes the mean accuracy (standard deviation) on Fashion-MNIST, CIFAR-10, and CIFAR-100 over the last ten epochs under label noise, compared with all baseline methods. According to paired t-tests, most ADIW variants

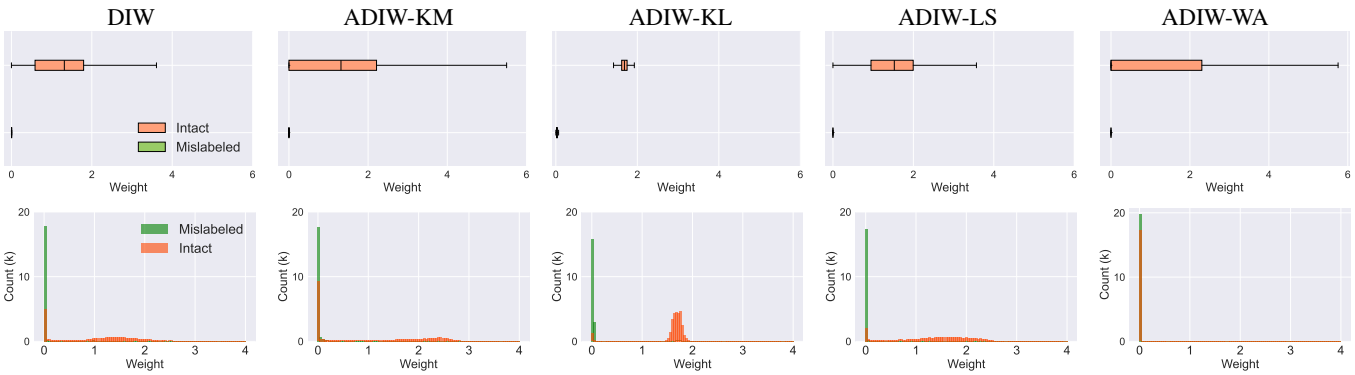


Fig. 6. Statistics of weight distributions on CIFAR-10 under 0.4 symmetric label noise (top: box plots; bottom: histogram plots).

TABLE IX
END-TO-END TRAINING TIME PER EPOCH* IN SECONDS ON FASHION-MNIST (F-MNIST), CIFAR-10, CIFAR-100, AND IMAGENET-1K.

Data	Noise	Uniform	L2R	MW-Net	DIW	ADIW-KM	ADIW-KL	ADIW-LS	ADIW-WA
F-MNIST	0.3 p	8.13	23.85	18.87	77.25	29.63	16.07	21.09	19.48
	0.4 s	8.02	23.15	18.93	80.10	29.81	21.80	24.10	19.92
	0.5 s	8.19	23.97	21.54	76.39	28.08	21.64	31.85	19.69
CIFAR-10	0.3 p	17.85	119.66	134.40	90.40	55.04	38.48	69.59	39.59
	0.4 s	17.90	121.38	152.12	109.55	49.71	38.82	67.94	39.66
	0.5 s	17.90	121.07	135.15	111.01	38.73	38.61	68.53	39.79
CIFAR-100	0.3 p	18.49	120.80	133.67	89.60	67.24	38.74	55.60	40.00
	0.4 s	18.64	120.68	153.93	110.65	67.15	38.56	50.19	39.87
	0.5 s	18.54	118.98	135.10	114.80	68.61	38.54	38.56	39.75
ImageNet-1K	0.4 s	746.99	18917.64	20087.89	20498.98	984.92	981.36	981.26	984.90

* Averaged over the full training schedule with the first epoch excluded (300 epochs for ImageNet-1K, 400 epochs for others). L2R, MW-Net, and DIW were run for only four epochs due to prohibitive cost. "p" and "s" denote pair and symmetric label noise, respectively.

achieve accuracy comparable to or better than DIW. Notable exceptions include ADIW-LS on CIFAR-10 with 0.3 pair label noise and on CIFAR-100, as well as ADIW-WA on CIFAR-100 with 0.3 pair label noise. Overall, ADIW-KM and ADIW-KL demonstrate consistently strong performance across all settings and are therefore recommended in practice.

C. Importance Weight Distributions

Figure 6 presents the weight distributions learned by DIW and all ADIW variants. Overall, both DIW and ADIW effectively assign higher weights to intact samples and lower weights to mislabeled ones. Among ADIW variants, ADIW-KL yields the most stable weighting, with weights concentrated around the median for both intact and mislabeled data, whereas other variants, especially ADIW-KM and ADIW-WA, produce broader distributions.

D. Summary of End-to-end Training Time

Table IX summarizes the end-to-end per-epoch training time under 0.3 pair, 0.4 symmetric, and 0.5 symmetric label noise settings. Figure 4 in the main text presents the 0.4 symmetric case as a representative setting, while this table confirms that similar efficiency trends hold across all label noise configurations.