Appendix Contents

A Algorithm Details

A.1 Latent Variable Model in CTOT

In this section, we provide a detailed explanation of the latent variable models used in classification and regression tasks respectively.

Classification. For the K-class classification task, we utilize K NDEs to separately model the latent dynamics of data for each category, and the observable data space is the representation space $\mathcal{Z}$. In other words, we model the temporal evolving processes of instances in the representation space over time using an NDE for each category separately. For instances belonging to the k-th ($k = 1, \cdots, K$) class:

$$\mathbf{h}_0 \sim p_{\theta_k}(\mathbf{h}_0), \tag{1}$$

$$\mathbf{h}_1, \cdots, \mathbf{h}_T = \text{NDESolve}(\theta_k, \mathbf{h}_0, (t_1, \cdots, t_T)), \tag{2}$$

$$\mathbf{z}_s = \psi_{\theta_k}(\mathbf{h}_s), s = 1, \cdots, T, \tag{3}$$

$$y_s \equiv k, s = 1, \cdots, T, \tag{4}$$

where θ_k represents the parameters for the k-th class. All learnable parameters $\theta = \{\theta_k\}_{k=1}^K$.

Regression. For regression task, we employ one NDE to control the latent dynamics of all the data, and the observable data space is the joint space of representation $\mathcal{Z}$ and label $\mathcal{Y}$. In other words, we model the temporal evolving processes of the concatenation of an instance's representation and its label:

$$\mathbf{h}_0 \sim p_{\theta}(\mathbf{h}_0), \tag{5}$$

$$\mathbf{h}_1, \cdots, \mathbf{h}_T = \text{NDESolve}(\theta, \mathbf{h}_0, (t_1, \cdots, t_T)), \tag{6}$$

$$(\mathbf{z}_s, y_s) = \psi_{\theta}(\mathbf{h}_s), s = 1, \cdots, T, \tag{7}$$

A.2 Pseudo Code of CTOT

The pseudo code of overall CTOT is given in Algorithm 1.

A.3 Prediction on the target domain

In this section, we detail the process of make predictions for the target domain data using the virtual generated data. For generating data, we first sample some $\mathbf{h}_0$ from the prior distribution $p_\theta(\mathbf{h}_0)$, then generate trajectories using NDEs with $\mathbf{h}_0$ as the initial state via Eq. (12) and Eq. (13). The final values of the trajectories at future timestamp t_{T+1} serve as the virtual data. Suppose that the generated data is $\hat{\mathcal{D}} = \left\{\hat{\mathbf{z}}^i, \hat{y}^i\right\}_{i=1}^N$ (We omit the subscript $T+1$ for simplicity).

Classification. Suppose there are totally K categories, we use $\hat{\mathcal{D}}$ to estimate the conditional distribution $p(\mathbf{z}|y)$ for each class by kernel density estimation (KDE) [Terrell and Scott, 1992]. Without loss of generality, we use a Gaussian kernel located at each $\hat{\mathbf{z}}^i$:

$$K_i(\mathbf{z}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\|\mathbf{z}-\hat{\mathbf{z}}^i\|_2^2/2\sigma^2}, \tag{8}$$

Algorithm 1 CTOT: Continuous-Time modelling Optimal Transport trajectories

Input: Source domains $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_T$ with each $\mathcal{D}_s = \left\{(\mathbf{x}_s^i, y_s^i)\right\}_{i=1}^{n_s}$, the feature extractor f, domain-specific task heads $\{g_s\}_{s=1}^T$, the latent variable model (denote all the learnable parameters by θ), the parameterized posterior model (denote the learnable parameters by ν), the batch size B, the hyperparameter $\lambda_{kl}, \lambda_{itp}$ and ϵ

// Step 1: Instance Evolving Trajectory Mining

for sampled mini-batch $\{\{\mathbf{x}_s^i, y_s^i\}_{i=1}^B\}_{s=1}^T$ **do**

$\mathcal{L}_{pt} \leftarrow \sum_{s=1}^T \sum_{i=1}^B \ell_{task}(g_s(f(\mathbf{x}_s^i)), y_s^i)$

Update parameters of f and $\{g_s\}_{s=1}^T$

end for

Freeze the parameters of f and $\{g_s\}_{s=1}^T$

Calculate representation of each instance via $\mathbf{z}_s^i = f(\mathbf{x}_s^i)$

Calculate optimal transport plan γ_s via Eq. (8)

Construct instance evolving trajectories $\mathcal{D}_{tra} = \left\{\{\mathbf{z}_s^i, y_s^i, t_s\}_{s=1}^{s=T}\right\}_{i=1}^N$ via Eq. (10)

// Step 2: Continuous-time Modelling of Trajectories

for sampled mini-batch $\left\{\{\mathbf{z}_s^i, y_s^i, t_s\}_{s=1}^{s=T}\right\}_{i=1}^B \sim \mathcal{D}_{tra}$ **do**

$\mathcal{L} \leftarrow 0$

for each instance evolving trajectory $\{\mathbf{z}_s^i, y_s^i, t_s\}_{s=1}^{s=T}$ **do**

Calculate the posterior q_ν via Eq. (14)

Sample initial latent state $\mathbf{h}_0 \sim q_\nu$

Generate an interpolated trajectory $\left\{(\hat{\mathbf{z}}_1^i, \hat{y}_1^i), (\hat{\mathbf{z}}_{1+\epsilon}^i, \hat{y}_{1+\epsilon}^i), \cdots, (\hat{\mathbf{z}}_{T-1+\epsilon}^i, \hat{y}_{T-1+\epsilon}^i), (\hat{\mathbf{z}}_T^i, \hat{y}_T^i), (\hat{\mathbf{z}}_{T+1}^i, \hat{y}_{T+1}^i)\right\}$ via Eq. (12) and Eq. (13)

Calculate reconstruction loss $\mathcal{L}_{recon}$ and KL loss $\mathcal{L}_{kl}$ via Eq. (15)

Calculate interpolation loss $\mathcal{L}_{itp}$ via Eq. (17)

$\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{recon} + \lambda_{kl}\mathcal{L}_{kl} + \lambda_{itp}\mathcal{L}_{itp}$

end for

Update parameters θ using $\nabla_\theta \mathcal{L}$

Update parameters ν using $\nabla_\nu \mathcal{L}$

end for

and the conditional distribution of class k is:

$$p(\mathbf{z}|y=k) = \cdot \frac{1}{N_k} \sum_{i=1}^{N} K_i(\mathbf{z}) \cdot \mathbb{I}(\hat{y}^i = k) \tag{9}$$

where N_k is the number of instances in class k and $\mathbb{I}(\cdot)$ is the indicator function. The bandwidth (standard deviation) σ can be chosen by validation on the last source domain. Then, for a test instance $\mathbf{x}$, we firstly extract its representation $\mathbf{z} = f(\mathbf{x})$ and classify it by the Bayes' rule:

$$p(y=k|\mathbf{z}) = \frac{p(\mathbf{z}|y=k)p(y=k)}{\sum_{i=1}^K p(\mathbf{z}|y=i)p(y=i)}, \tag{10}$$

If there is no apparent class imbalance in the source domains, we consider the prior probability as a uniform distribution, i.e., $p(y=k) = \frac{1}{K}$. If there is a significant class imbalance in the source domains, and the class frequencies exhibit a temporal pattern, we use time series forecasting methods (e.g., ARIMA, RNNs) to predict $p(y=k)$ at t_{T+1}.

Regression. Let $\hat{\mathbf{Z}} = [\hat{\mathbf{z}}^1, \cdots, \hat{\mathbf{z}}^N]^T$ and $\hat{\mathbf{Y}} = [\hat{y}^1, \cdots, \hat{y}^N]^T$. Since during pretraining on all source domains, each task head g_s used is a single linear layer without activation, essentially applying a linear model on representations. Therefore, it is assumed that the linear relationship between the representations and labels also approximately holds for the test domain. Training g is equivalent to performing linear regression on the representations. For the sake of simplicity, we directly consider the optimization objective of **Ridge Regression** [Hoerl and Kennard, 1970], which is to minimize:

$$J(\mathbf{W}, b) = \|\hat{\mathbf{Y}} - \hat{\mathbf{Z}}\mathbf{W} - b\|^2 + \lambda_{ridge}\|\mathbf{W}\|^2, \tag{11}$$

with respect to weight $\mathbf{W}$ and bias b. This regularization term $\|\mathbf{W}\|^2$ is chosen for its ability to handle multicollinearity and improve the stability of the regression coefficients. Ridge regression has an explicit solution as follows:

$$\mathbf{W} = (\hat{\mathbf{Z}}^T\hat{\mathbf{Z}} + \lambda_{ridge}\mathbf{I})^{-1}\hat{\mathbf{Z}}^T\hat{\mathbf{Y}}, \tag{12}$$

$$b = \frac{1}{N}\mathbf{1}^T(\hat{\mathbf{Y}} - \hat{\mathbf{Z}}\mathbf{W}). \tag{13}$$

Then, $\mathbf{W}$ and b are used to make prediction for a test instance $\mathbf{x}$:

$$y_{pred} = \mathbf{z}^T\mathbf{W} + b, \text{ where } \mathbf{z} = f(\mathbf{x}). \tag{14}$$

B Additional Experimental Details

B.1 Dataset Details

In this section, we provide a detailed description of the datasets used in our experiments.

2-Moons: This is a variant of the 2-entangled moons dataset, with a lower moon and an upper moon labeled 0 and 1 respectively. Each moon consists of 100 instances, and 10 domains are obtained by sampling 200 data points from the 2-Moons distribution, and rotating them counterclockwise in units of 18°. Domains 1 to 9 (both inclusive) are our training domains ($\boldsymbol{T} = \mathbf{9}$), and domain 10 is for testing. Concept drift means the rotation of the moon-shape clusters.

Rot-MNIST: This is an adaptation of the popular MNIST digit dataset [Deng, 2012], where the task is to classify a digit from 0 to 9 given an image of the digit. We generate 5 domains by rotating the images in steps of 15°. To generate the i-th domain, we sample 1,000 images from the MNIST dataset and rotate them counter-clockwise by $15 \times (i-1)$ degrees. We take the first four domains as train domains ($\boldsymbol{T} = \mathbf{4}$) and the fifth domain as test.

ONP: This dataset [Fernandes *et al.*, 2015] summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. The goal is to predict the number of shares in social networks (popularity). We split the dataset by time into 6 domains and use the first 5 for training ($\boldsymbol{T} = \mathbf{5}$). The concept drift is reflected in the change of time, but previous works [Nasery *et al.*, 2021; Bai *et al.*, 2023] have proven the concept drift is not strong.

Shuttle: This dataset[1] provides about 58,000 data points for space shuttles in flight. The task is multiclass classification with a heavy class imbalance. The dataset was divided into 8 domains based on the time points associated with points, with times between 30-70 being the train domains ($\boldsymbol{T} = \mathbf{7}$) and 70 -80 being the test domain.

Elec2: This contains information about the demand of electricity in a particular province. The task is binary classification, to predict if the demand of electricity in each period (of 30 mins) was higher or lower than the average demand over the last day. We consider two weeks to be one time domain, and train on 40 domains ($\boldsymbol{T} = \mathbf{40}$) while testing on the 41st domain.

House: This dataset has housing price data from 2013-2019. This is a regression task to predict the price of a house given the features. We treat each year as a separate domain, but also give information about the exact date of purchase to the models. We take data from the year 2019 to be test data and prior data as training ($\boldsymbol{T} = \mathbf{6}$). Similar to Elec2, the concept drift in this dataset is how the housing price changed from 2013-2019 for a certain region.

Appliance: This dataset [Candanedo *et al.*, 2017] is used to create regression models of appliances energy use in a low energy building. The data set is at 10 min for about 4.5 months in 2016, and we treat each half month as a single domain, resulting in 9 domains in total. The first 8 domains are used for training ($\boldsymbol{T} = \mathbf{8}$) and the last one is for testing. Similar to Elec2, the drift for this dataset corresponds to how the appliances energy usage changes in a low energy building over about 4.5 months in 2016.

B.2 Baseline Details

In this section, we provide a detailed description of the compared baseline methods.

Time-Agnostic Baselines: (1) Offline: This method trains a prediction model on all the source domains using Empirical Risk Minimization (ERM). (2) LastDomain: This method trains a model only on the last source domain using ERM. (3) IncFinetune: This method biases the training towards more recent data. After being trained on the prior domains, the model is finetuned on the subsequent domain using a reduced learning rate in a sequential manner.

Continuous Domain Adaptation: (1) CDOT [Ortiz-Jimenez *et al.*, 2019]: This method transports the most recent labeled examples $\mathcal{D}_T$ to the future using a learned coupling from past data, and trains a classifier on them. (2) CIDA [Wang *et al.*, 2020a]: This method combines traditional adversarial adaptation with a discriminator that models the encoding-conditioned domain index distribution.

Temporal Domain Generalization: (1) GI [Nasery *et al.*, 2021]: This method proposes a time-sensitive model architecture and encourages the model to learn functions which can extrapolate well to the near future by supervising the first order Taylor expansion of the learnt function. (2) LSSAE [Qin *et al.*, 2022]: This method uses two latent variables to represent the covariate shift and label shift, and propose a domain-related module and a category-related module to infer their dynamic transition functions based on different timestamps, using variational inference. (3) DDA [Zeng *et al.*, 2023]: This method designs a domain transformer module which generates future instances based on instances of previously seen domains, and the module is trained using a bi-level meta-learning framework. (4) DRAIN [Bai *et al.*, 2023]: This method uses recurrent neural networks to autoregressively predict the optimal model parameters of the next domain. (5) EvoS [Xie *et al.*, 2023]: This method uses a multi-scale attention module to learn the evolving pattern of the first-order and second-order statistics of feature distribution of each domain, and transforms each domain distribution into a common normal distribution via feature standardization.

[1] https://archive.ics.uci.edu/dataset/148/statlog+shuttle

B.3 Network Architectures and Implementation Details

Feature Extractor and Task Head. To ensure a fair comparison, we keep a consistent model architecture for the feature extractor and task head across out method and all the compared baselines. Concretely, we use the same backbone model architecture specified by [Bai *et al.*, 2023]:

- 2-Moons: The feature extractor f is a Multi-layer Perceptron (MLP) with two hidden layers, each with a dimension of 50.
- Rot-MNIST: The feature extractor f consists of two convolution layers with kernel shape 3×3, and each convolution layer is followed by a max pooling layer with kernel size 2 and stride = 2.
- ONP: The feature extractor f is an MLP with two hidden layers, each with a dimension of 200.
- Shuttle: The feature extractor f is an MLP with three hidden layers, each with a dimension of 128.
- Elec2: The feature extractor f is an MLP with two hidden layers, each with a dimension of 128.
- House: The feature extractor f is an MLP with two hidden layers, each with a dimension of 128.
- Appliance: The feature extractor f is an MLP with two hidden layers, each with a dimension of 128.

For Rot-MNIST, the task head g is a two-layer MLP with the hidden dimension being 256 and output dimension being 10. For the other datasets, the task head is a single linear layer.

Optimal Transport. We use the POT[2] package to compute the optimal transportation plan.

Neural Differential Equations. The architecture of drift network ϕ_θ is a 4-layer MLP which takes the concatenation of representation $\mathbf{z}$ and time t as input, consisting of two hidden layers and an output layer whose dimension is the same as the dimension of latent variable $\mathbf{h}$, i.e., d_h. The noise type of SDE is diagonal, and the architecture of diffusion network σ_θ is a 3-layer MLP which takes the concatenation of representation $\mathbf{z}$ and time t as input, consisting of one hidden layer and an output layer whose dimension is also d_h. The hidden dimension is set equal to the dimension of representation. The Brownian motion is d_h-dimensional. The prior of $\mathbf{h}$ is a learnable Gaussian. The decoder ψ_θ is a single linear layer. For the approximate posterior q_ν, we use a Gated Recurrent Unit (GRU) to encode a series $\{\mathbf{z}_s, y_s, t_s\}_{s=1}^{s=T}$ into a context vector, which is then transformed by a linear layer to output the Gaussian posterior $\mathcal{N}(\mathbf{m}_{\mathbf{h}_0}, \mathbf{v}_{\mathbf{h}_0})$. We implement the NDEs using torchdiffeq[3] and torchsde[4] packages.

B.4 Hyperparameters

For tuning hyperparameters, we consider data from the most recent domain ($\mathcal{D}_T$) as the validation set. Following the hyperparameter search protocol in DomainBed [Gulrajani and Lopez-Paz, 2021], we set a search space for each hyperparameter, as listed in Table 3. The regularization strength of ridge regression is set as 1. The network is trained using the Adam optimizer with weight decay. Codes for reproducing our experimental results are provided.

Hyper-Parameter	Search Space
Batch size	$\{32, 64, 128\}$
Learning rate	{2e-5, 1e-4, 2e-4, 5e-4, 2e-3}
d_h	$\{32, 64, 128\}$
λ_{kl}	{1e-4, 1e-3, 1e-2}
λ_{itp}	$\{0.1, 0.5, 1.0, 2.0\}$
α	{1e-4, 1e-2, 1.0}

Table 3: Hyperparameter search space.

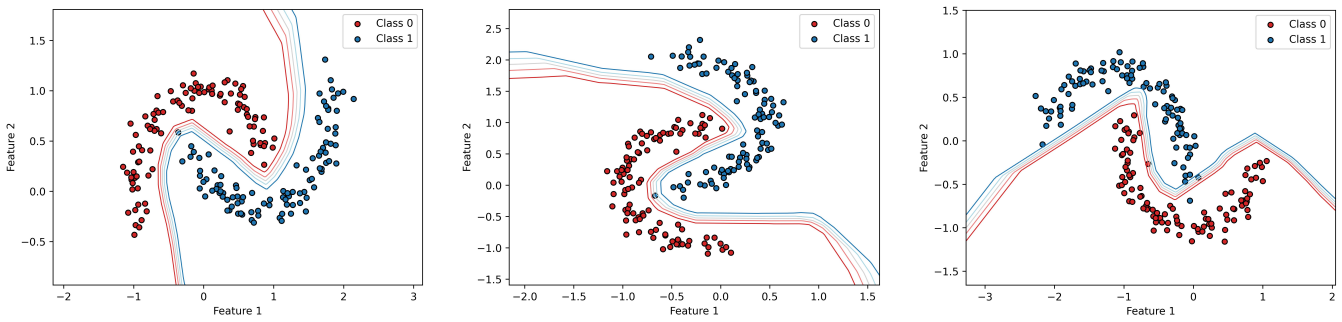
B.5 Qualitative Analysis

In this section, we provide a qualitative analysis by visualizing the decision boundary on the 2-Moons dataset. As shown in Figure 3a - 3c, we visualize the decision boundary learned by CTOT on source domains $\mathcal{D}_2, \mathcal{D}_6$, and the final decision boundary predicted by CTOT on the target domain $\mathcal{D}_{10}$. As observed, CTOT successfully captures the temporal evolving pattern of the decision boundary, ensuring its accurate rotation over time.

[2]https://github.com/PythonOT/POT

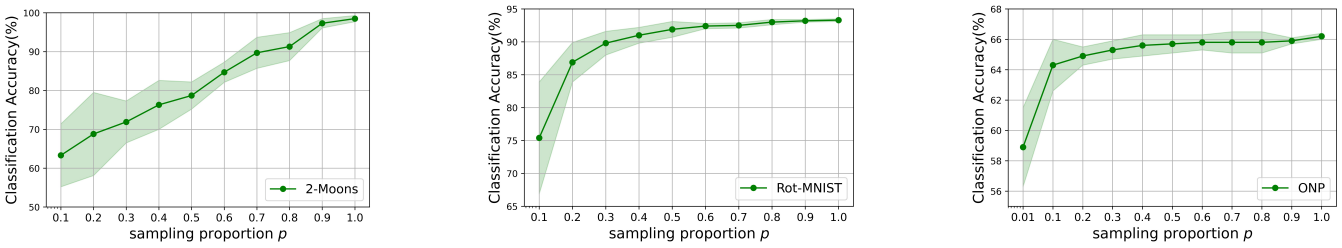
[3]https://github.com/rtqichen/torchdiffeq

[4]https://github.com/google-research/torchsde



(a) Source Domain $\mathcal{D}_2$ (b) Source Domain $\mathcal{D}_6$ (c) Target Domain $\mathcal{D}_{10}$

Figure 3: **Visualization of the decision boundary of CTOT** on 2-Moons dataset.



(a) 2-Moons dataset (b) Rot-MNIST dataset (c) ONP dataset

Figure 4: **Sensitivity analysis of sampling.** Classification accuracy (%) with standard deviation are shown.

B.6 Training Time Analysis

We further conduct the model scalability analysis by comparing the running time of our proposed method with the Offline baseline on three datasets (2-Moons, Elec2 and House). As shown in Table 4, the time cost of the pretraining step of CTOT is slightly larger than that of Offline, since we train multiple domain-specific task heads. We observe that the overall time overhead of our approach is 1 to 5 times that of Offline, depending on the specific dataset and hyperparameter settings. **This indicates that our approach has good scalability and does not introduce excessive time overhead.**

Method	2-Moons	Elec2	House
Offline	27.5	910.2	169.3
CTOT (pretrain)	28.2	922.5	189.2
CTOT (total)	65.4	1204.3	860.1

Table 4: Training time in seconds of various algorithms on some datasets.

B.7 Sensitivity Analysis of Sampling

In practical applications, especially when the sample size is very large, the computational cost of determining the optimal transport plan will be significant (For example, time complexity $\mathcal{O}(n^3 \log n)$ for exact solver and nearly $\mathcal{O}(n^2)$ for the Sinkhorn solver). Therefore, we additionally conduct an observational experiment to assess the sensitivity of CTOT's performance to data sampling. Specifically, we randomly sample a proportion p of instances for each source domain, and then use these sampled instances to construct instance evolving trajectories. The results are shown in Figure 4. From Figure 4a, we observe that the performance on the 2-Moons dataset decays as p decreases. From Figure 4b, we observe that the performance on Rot-MNIST is relatively stable when $0.5 < p < 1$, but a noticeable decline occurs when p decreases below 0.5. As shown in Figure 4c, the performance on ONP is very stable when $0.1 < p < 1$, and begins to decay when $p < 0.1$. Note that without sampling, the total number of trajectories in datasets 2-Moons, Rot-MNIST, and ONP is 200, 1000, and 7049, respectively. We conclude that: **when the dataset is large, the performance of CTOT is stable across a wide range of sampling proportion, as the sampled data has the capability to approximate temporal drift correctly and provides an ample number of trajectories for model learning.** When the dataset is small, no sampling is needed.

B.8 Remarks and Future Work

We mark that the proposed CTOT provides a very generic framework for TDG, which can have many variants.

- For instance evolving trajectory mining, variants of optimal transport can be adapted. The structure of the optimal transport plan can be constrained by regularization techniques such as entropic regularization, quadratic regularization or group lasso regularization [Flamary *et al.*, 2016]. The structural relationships across instances within each domain can also be taken into account, which leads to the Fused-Gromov-Wasserstein (FGW) transport [Vayer *et al.*, 2020].
- For continuous-time modelling, the choice of continuous-time models is also versatile, as three advantages of such models are highly suitable for TDG tasks: (1) They offer the flexibility to handle irregularly-sampled time series, (2) They are suitable for modelling continuous dynamics, (3) They offer stronger extrapolation capabilities than discrete-time models, especially when observations are sparse. Apart from NDEs, there are other continuous-time modelling methods which can be used, such as continuous-time flow process (CTFP) [Deng *et al.*, 2020]. The interpolation loss we proposed, along with the method for extrapolating to generate data, are compatible with them.

We leave these discussions for future work.