Continuous Temporal Domain Generalization

Zekun Cai[1,4], **Guangji Bai**[2], **Renhe Jiang**[1*], **Xuan Song**[3,4], and **Liang Zhao**[2]

[1]The University of Tokyo, Tokyo, Japan
[2]Emory University, Atlanta, GA, USA
[3]Jilin University, Changchun, China
[4]Southern University of Science and Technology, Shenzhen, China
{caizekun,jiangrh,songxuan}@csis.u-tokyo.ac.jp
{guangji.bai,liang.zhao}@emory.edu

Abstract

Temporal Domain Generalization (TDG) addresses the challenge of training predictive models under temporally varying data distributions. Traditional TDG approaches typically focus on domain data collected at fixed, discrete time intervals, which limits their capability to capture the inherent dynamics within continuous-evolving and irregularly-observed temporal domains. To overcome this, this work formalizes the concept of Continuous Temporal Domain Generalization (CTDG), where domain data are derived from continuous times and are collected at arbitrary times. CTDG tackles critical challenges including: 1) Characterizing the continuous dynamics of both data and models, 2) Learning complex high-dimensional nonlinear dynamics, and 3) Optimizing and controlling the generalization across continuous temporal domains. To address them, we propose a Koopman operator-driven continuous temporal domain generalization (Koodos) framework. We formulate the problem within a continuous dynamic system and leverage the Koopman theory to learn the underlying dynamics; the framework is further enhanced with a comprehensive optimization strategy equipped with analysis and control driven by prior knowledge of the dynamics patterns. Extensive experiments demonstrate the effectiveness and efficiency of our approach. The code can be found at: https://github.com/Zekun-Cai/Koodos.

1 Introduction

In practice, the distribution of training data often differs from that of test data, leading to a failure in generalizing models outside their training environments. Domain Generalization (DG) [49; 46; 21; 14; 40] is a machine learning strategy designed to learn a generalized model that performs well on the unseen target domain. The task becomes particularly pronounced in dynamic environments where the statistical properties of the target domains change over time [16; 33; 4], prompting the development of Temporal Domain Generalization (TDG) [27; 39; 41; 2; 55; 57]. TDG recognizes that domain shifts are temporally correlated. It extends DG approaches by modeling domains as a sequence rather than as categorical entities, making it especially beneficial in fields where data is inherently time-varying.

Existing works in TDG typically concentrate on the discrete temporal domain, where domains are defined by distinct "points in time" with fixed time intervals, such as second-by-second data (Rot 2-Moons [48]) and annual data (Yearbook [54]). In this framework, data bounded in a time interval are considered as a detached domain, and TDG approaches primarily employ probabilistic models to predict domain evolutions. For example, LSSAE [41] employs a probabilistic generative model to analyze latent structures within domains; DRAIN [2] builds a Bayesian framework to predict future model parameters, and TKNets [57] constructs domain transition matrix derived from the data.

* Corresponding author

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

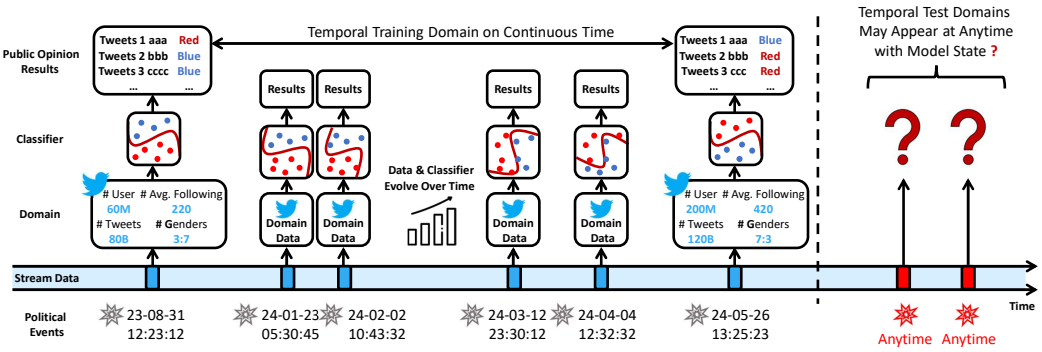


Figure 1: An example of continuous temporal domain generalization. Consider training classification models for public opinion prediction via tweets, where the training domains are only available at specific political events (e.g., presidential debates), we wish to generalize the model to any future based on the underlying data distribution drift within the time-irregularly distributed training domains.

However, in practice, data may not always occur or be observed at discrete, regularly spaced time points. Instead, events and observations unfold irregularly and unpredictably in the time dimension, leading to temporal domains distributed irregularly and sparsely over continuous time. Formally, this paper introduces a problem termed Continuous Temporal Domain Generalization (CTDG), where both seen and unseen tasks reside in different domains at continuous, irregular time points. Fig. 1 illustrates an example of public opinion prediction after political events via Twitter data. Unlike the assumption in traditional TDG that the temporal regularly domains, the data are collected for the times near political events that may occur in arbitrary times. In the meanwhile, the domain data evolve constantly and continuously over time, e.g., active users increase, new friendships are formed, and age and gender distribution changes. Correspondingly, the ideal classifier should gradually change with the domain at random moments to counter the data distribution change over time. Finally, we are concerned with the state of the predictive model at any moment in the future. CTDG is ubiquitous in other fields. For example, in disaster management, relevant data are collected during and following disasters, which may occur at any time throughout the year. In healthcare, critical information about diagnosis and treatment is typically only documented during episodes of care rather than evenly throughout the lifetime. Hence, the CTDG task necessitates the characterization of the continuous dynamics of irregular time-distributed domains, which cannot be handled by existing TDG methods designed for discrete-dynamics and fixed-interval times.

Despite the importance of CTDG, it is still a highly open research area that is not well explored because of several critical hurdles in: **1) Characterizing data dynamics and their impact on model dynamics.** The irregular times of the temporal domains require us to characterize the continuous dynamics of the data and, hence, the model dynamics ultimately. However, the continuous-time data dynamics are unknown and need to be learned across arbitrary time points. Furthermore, it is imperative yet challenging to know how the model evolves according to the data dynamics in continuous times. Therefore, we don't have a direct observation of the data dynamics and the model dynamics we want to learn, which prevents us from existing continuous time modeling techniques. **2) Learning the underlying dynamics of over-parametrized models.** Deep neural networks (e.g., Multi-Layer Perceptron and Convolutional Neural Network) are highly nonlinear and over-parametrized, and hence, the evolutionary dynamics of model states over continuous time are high-dimensional and nonlinear. Consequently, the principal dynamics reside in a great number of latent dimensions. Properly representing and mapping these dynamics into a learnable space remains a challenge. **3) Jointly optimizing the model and its dynamics under possible inductive bias.** The model learning for individual domains will be entangled with the learning of the continuous dynamics across these models. Furthermore, in many situations, we may have some high-level prior knowledge about the dynamics, such as whether there are convergent, divergent, or periodic patterns. It is an important yet open topic to embed them into the CTDG problem-solving.

To address all the challenges, we propose the Koopman operator-driven continuous temporal domain generalization framework (Koodos). Specifically, the Koodos framework articulates the evolutionary continuity of the predictive models in CTDG, then leverages a continuous dynamic approach to model its smooth evolution over time. Koodos further simplifies the nonlinear model system by projecting them into a linearized space via the Koopman Theory. Finally, Koodos provides an interface that

reveals the internal model dynamic characteristics, as well as incorporates prior knowledge and constraints directly into the joint learning process.

2 Related works

Domain Generalization (DG) and Domain Adaptation (DA). DG approaches attempt to learn a model from multiple source domains that generalize well to an unseen domain [38; 37; 28; 5; 13; 56]. Existing DG methods can be classified into three strategies [49]: (1) Data manipulation techniques, such as data augmentation [45; 46; 57] and data generation [32; 40]; (2) Representation learning focuses on extracting domain-invariant features [17; 18] and disentangling domain-shared from domain-specific features [30]; (3) Learning strategies encompass ensemble learning [34], meta-learning [26; 14; 9], and gradient-based approaches [21]. Unlike DG, DA methods require simultaneously accessing source and target domain data to facilitate alignment and adaptation [50; 51; 29]. The technique includes domain-invariant learning [17; 47; 35; 48; 53], domain mapping [6; 20; 15; 31], ensemble methods [43], and so on. *Both DG and DA are limited to considering generalization across categorical domains, which treats domains as individuals but ignores the smooth evolution of them over time.*

Temporal Domain Generalization (TDG). TDG is an emerging field that extends traditional DG techniques to address challenges associated with time-varying data distributions. TDG decouples time from the domain and constructs domain sequences to capture its evolutionary relationships. S-MLDG [27] pioneers a sequential domain DG framework based on meta-learning. Gradient Interpolation (GI) [39] proposes to extrapolate the generalized model by supervising the first-order Taylor expansion of the learned function. LSSAE [41] deploys a probabilistic framework to explore the underlying structure in the latent space of predictive models. DRAIN [2] constructs a recurrent neural network that dynamically generates model parameters to adapt to changing domains. TKNets [57] minimize the divergence between forecasted and actual domain data distributions to capture temporal patterns.

Despite these studies, traditional TDG methods are limited to requiring domains presented in discrete time, which disrupts the inherent continuity of changes in data distribution, and the generalization can only be carried forward by very limited steps. *No work treats time as a continuous variable, thereby failing to capture the full dynamics of evolving domains and generalize to any moment in the future.*

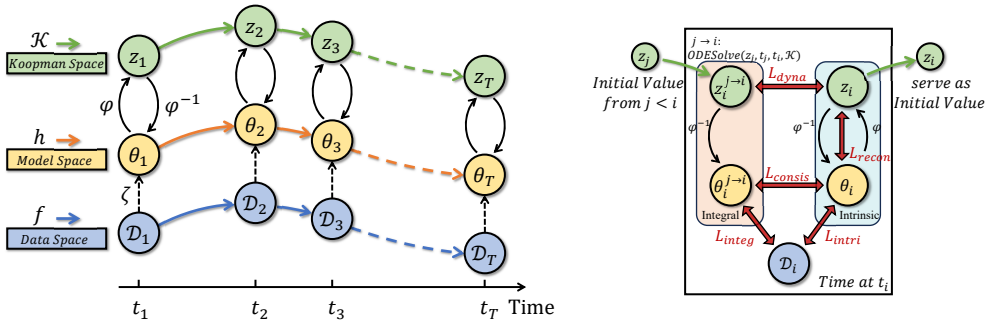
Continuous Dynamical Systems (CDS). CDS are fundamental in understanding how systems evolve without the constraints of discrete intervals. They are the study of the dynamics for systems defined by differential equations. The linear multistep method or the Runge-Kutta method can solve the Order Differential Equations (ODEs) [19]. Distinguishing from traditional methods, Neural ODEs [11] represent a significant advancement in the field of CDS. It defines a hidden state as a solution to the ODEs initial-value problem, and parameterizes the derivatives of the hidden state using a neural network. The hidden state can then be evaluated at any desired time using a numerical ODEs solver. Many recent studies have proposed on which to learn differential equations from data [42; 22; 23; 36].

3 Problem definition

Continuous Temporal Domain Generalization (CTDG): We address prediction tasks where the data distribution evolves over time. In predictive modeling, a domain $\mathcal{D}(t)$ is defined as a dataset collected at time t consisting of instances $\{(x_i^{(t)}, y_i^{(t)}) \in \mathcal{X}(t) \times \mathcal{Y}(t)\}_{i=1}^{N(t)}$, where $x_i^{(t)}$, $y_i^{(t)}$ and $N(t)$ represent the feature, target and the number of instances at time t, and $\mathcal{X}(t)$, $\mathcal{Y}(t)$ denote the input feature space and label space at time t, respectively. We focus on the existence of gradual concept drift across continuous time, indicating that domain conditional probability distributions $P(Y(t)|X(t))$, with $X(t)$ and $Y(t)$ representing the random variables for features and targets at time t, change smoothly and seamlessly over continuous time like streams without abrupt jumps.

During training, we are provided with a sequence of observed domains $\{\mathcal{D}(t_1), \mathcal{D}(t_2), \ldots, \mathcal{D}(t_T)\}$ collected at arbitrary times $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$, where $t_i \in \mathbb{R}^+$ and $t_1 < t_2 < \ldots < t_T$. For each domain $\mathcal{D}(t_i)$ at time $t_i \in \mathcal{T}$, we learn the predictive model $g(\cdot; \theta(t_i)) : \mathcal{X}(t_i) \to \mathcal{Y}(t_i)$, where $\theta(t_i)$ denotes the parameters of function g at timestamp t_i. We model the dynamics across the parameters $\{\theta(t_1), \theta(t_2), \ldots, \theta(t_T)\}$, and finally predict the parameters $\theta(s)$ for the predictive model $g(\cdot; \theta(s)) : \mathcal{X}(s) \to \mathcal{Y}(s)$ at any given time $s \notin \mathcal{T}$. For simplicity, in subsequent, we will use $\mathcal{D}_i$, X_i, Y_i, θ_i to represent $\mathcal{D}(t_i)$, $X(t_i)$, $Y(t_i)$, $\theta(t_i)$ at time t_i.

Unlike traditional temporal domain generalization approaches [39; 2; 52; 57] that divide time into discrete intervals and require domain data be collected at fixed time steps, the CTDG problem



(a) Three Spaces and Parallel Flows in Continuous Temporal Domains (b) State Constraints at a Given Moment

Figure 2: Macro-flows and micro-constraints in the proposed model framework.

treats time as continuous variable, allowing for the time points in training and test set to be any arbitrary positive real numbers, and requiring models to deal with the temporal domains as continuous streams. CTDG problem poses several unprecedented, substantial challenges in: 1) Characterizing the continuous dynamics of data and how that decides model dynamics; 2) Modeling the low-dimensional continuous dynamics of the predictive model, which are embedded in high-dimensional space due to over-parametrization; and 3) Optimizing and analyzing the continuous predictive model system, including the application of inductive biases to control its behavior.

4 Methodology

In this section, we outline our strategies to tackle the problem of CTDG by overcoming its substantial challenges. First, to learn the continuous drifts of data and models, we synchronize between the evolving domain and the model by establishing the continuity of the model parameters over time, which is ensured through the formulation of differential equations as elaborated in Section 4.1. To fill the gap between high-dimensional model parameters and low-dimensional model continuous dynamics, we simplify the representation of complex dynamics into a principal, well-characterized Koopman space as is detailed in Section 4.2. To jointly learn the model and its dynamics under additional inductive bias, in Section 4.3, we design a series of organized loss functions to form an efficient end-to-end optimization strategy. Overall, as shown in Fig. 2(a), there are three dynamic flows in our system, which are the Data Flow, the Model Flow, and the Koopman Representation Flow. Through the proposed framework, we aim to ensure not only that the model responds to the statistical variations inherent in the data, but also that the characteristics of the three flows are consistent.

4.1 Characterizing the continuous dynamics of the data and the model

In this section, we explore the relationship between the evolving continuous domains and the corresponding dynamics of the predictive models. We demonstrate that the dynamics of temporal domains lead to the internal update of the predictive model continuously over time in Theorem 1. Following this, we develop a learnable dynamic system for the continuous state of the model by synchronizing the behaviors of the domain and model.

Assumption 1. *Consider the gradual concept drift within the continuous temporal domains. It is assumed that the conditional probability distribution $P_t(Y|X)$ changes continuously over time, and its dynamics are characterized by a function f, which models the variations in the distribution.*

Theorem 1. *(Continuous Evolution of Model Parameters) Given Assumption 1, it follows that the parameters θ_t of the predictive model $g(\cdot;\theta_t)$ also evolve continuously over time, and its dynamics are jointly determined by the current state of the model and the function f.*

Proof. The temporal derivative of the functional space $g(\cdot;\theta_t)$ represents its evolution in direct response to the changes in the conditional probability distribution. Without loss of generality, the ground-truth functional space can be modeled by an ordinary differential equation:

$$dg(\cdot;\theta_t)/dt = f(g(\cdot;\theta_t), t). \tag{1}$$

Applying the chain rule to decompose the temporal derivative of g:

$$\frac{d}{dt}g(\cdot;\theta_t) = \sum_{i=1}^{n} \frac{\partial g}{\partial \theta_{t,i}} \frac{d\theta_{t,i}}{dt} = J_g(\theta_t)\frac{d\theta_t}{dt}, \tag{2}$$

where $J_g(\theta_t)$ is the Jacobian matrix of g with respect to θ_t.

By equating the Eq. 2 to the expression involving f:

$$J_g(\theta_t)\frac{d\theta_t}{dt} = f(g(\cdot;\theta_t), t). \tag{3}$$

Assuming that $J_g(\theta_t)$ is invertible, the ground-truth derivative of θ_t with respect to time is given by:

$$\frac{d\theta_t}{dt} = J_g(\theta_t)^{-1} f(g(\cdot;\theta_t), t). \tag{4}$$

□

It is known from the Proof that the evolution continuity of θ_t follows from a differential equation. Eq. 4 identifies a dynamic framework for the predictive model parameters, which provides a strong motivation to develop dynamic systems for them.

The principal challenge arises from the unknown dynamics enclosed in f, which prevents getting θ_t by direct mathematical computation. Recognizing this, we propose a learning-based approach to model the parameter dynamics from the observed domains. We construct a learnable parameter dynamics function $h(\theta_t, t; \phi)$, parameterized by ϕ, optimized to topological conjugation between the model dynamics and the data dynamics. Topological conjugation guarantees that the parameter's orbits faithfully characterize the underlying data dynamics, thereby ensuring the model's generalization capability across all time. Specifically, as illustrated in Fig. 2(a) of the Data Flow and the Model Flow, assuming a conceptual perfect mapping ξ from the data space to the parameter space, topological conjugation suggests that the composition of the learned dynamic h after ξ should behave identically to applying ξ after the dynamics f, i.e., $h \circ \xi = \xi \circ f$ holds, where $\circ$ represents function composition. In the CTDG problem, the objective of conjugation is to optimize the model dynamics h and the predictive parameters θ simultaneously, to minimize the discrepancy between the dynamically derived parameters from h and those obtained through direct training, formulated as follows:

$$\phi = \arg\min_{\phi} \sum\nolimits_{i=1}^{T} \sum\nolimits_{j=1}^{i} \|\theta_i, \theta_i^{j\to i}\|_2, \tag{5}$$

where $\|\cdot\|_2$ denotes the Euclidean norm, and each θ_i is determined by:

$$\theta_i = \arg\min_{\theta_i} \mathcal{L}(Y_i, g(X_i; \theta_i)), \tag{6}$$

and $\theta_i^{j\to i}$ is defined as the integral parameters at t_i obtained from $t_j < t_i$:

$$\theta_i^{j\to i} = \theta_j + \int_{t_j}^{t_i} h(\theta_\tau, \tau; \phi)\, d\tau. \tag{7}$$

Here, $\mathcal{L}$ symbolizes the loss function tailored to the prediction task. By employing Eq. 7, the model dynamics are defined for all observation time, and ϕ can be obtained by optimizing Eq. 5 and Eq. 6 via gradient descent. After that, the parameters of the predictive model can be calculated at any specific time using ODE solvers.

4.2 Modeling nonlinear model dynamics by Koopman operators

The aforementioned learning approaches are, however, limited in efficient modeling and prediction due to the entangled nonlinear dynamics in the high-dimensional parameters space. Identifying strongly nonlinear dynamics through such space is particularly susceptible to overfitting [8], leading to future predictive models eventually diverging from bifurcation to chaos. Despite the complexity of h, it is governed by the data dynamics f, which are typically low-dimensional and exhibit simple, predictable patterns. This suggests that the governing equations for h are sparse within the over-parameterized space, and thus allow for a more simplified and manageable representation in a properly transformed space. Motivated by this, we propose leveraging the well-established Koopman Theory [24; 7] to represent these complex parameter dynamics. Koopman Theory provides a method for the global linearization of any nonlinear dynamics. It expresses the complex dynamic system as an infinite-dimensional Koopman operator acting on the Hilbert space of the system state measurement functions, in which the nonlinear dynamics will become linearized.

To facilitate this approach, the target is to identify a set of intrinsic coordinates φ that spans a Koopman-invariant subspace. As illustrated in Fig. 2(a) of the Model Flow and Koopman Flow, this transformation maps the parameters θ into a latent space $z = \varphi(\theta)$, with z resides in a low n-dimensional space, $\mathcal{Z} = \mathbb{R}^n$, where the dynamics become more linearized. We also aim to find a finite-dimensional approximation of the Koopman operator, denoted as $\mathcal{K} \in \mathbb{R}^{n\times n}$, that acts on the latent space and advances the observation of the state to the future $\mathcal{K} : \mathcal{Z} \to \mathcal{Z}$. After that we have:

$$d\varphi(\theta)/dt = \mathcal{K}\varphi(\theta). \tag{8}$$

The z dynamics are easier to track because they principleize the dynamical system while maintaining its characteristics. It gives us a tighter representation of the parameters within a linear space, which allows us to learn the simple $\mathcal{K}$ operator instead of the complex coupled dynamics $h(\cdot;\phi)$. Following the transformation, the dynamics of the parameters can be expressed by:

$$z_i^{j\to i} = z_j + \int_{t_j}^{t_i} \mathcal{K} z_\tau \, d\tau, \quad \text{where } z_j = \varphi(\theta_j). \tag{9}$$

Finally, an inverse transformation provided by $\theta_i^{j\to i} = \varphi^{-1}(z_i^{j\to i})$ that maps Koopman space back to the original parameter space. The relational among θ, $\mathcal{K}$, φ, φ^{-1} and the Koopman invariant subspace are bounded by a series of loss functions detailed in the next Section.

4.3 Joint optimization of model and its dynamics with prior knowledge

We introduce a comprehensive optimization approach designed to ensure that the system accurately captures the dynamics of the data. This process requires the joint optimization of several interconnected components under the optional constraint of additional prior knowledge about the underlying dynamics: the predictive model parameters $\theta_{1:T}$, the transformation functions φ and φ^{-1}, and the Koopman operator $\mathcal{K}$. Our primary objectives are threefold: *1) Ensuring high prediction accuracy, 2) Maintaining consistency of parameters across different representations and transformations, and 3) Learning the Koopman invariant subspaces effectively*. Fig. 2(b) illustrates the role of each constraint within our system: we manage two sets of states, intrinsic and integral, aligning across three spaces.

Predictive Model Parameters (θ): Each observation time corresponds to a predictive model, which is tasked with making predictions and serving as the initial values for the dynamical system. They are primarily optimized to minimize the prediction error L_{intri} of different domains:

$$L_{intri} = \sum\nolimits_{i=1}^{T} \mathcal{L}(Y_i, g(X_i;\theta_i)). \tag{10}$$

Koopman Network Parameters (φ, φ^{-1}, $\mathcal{K}$): We estimate a function that transforms the parameters of the predictive model into Koopman space. Meanwhile, these predictive model parameters must be stable and consistent when converted between representations. This is realized by three constraints:

1. **Reconstruction Loss:** An autoencoder is leveraged to map parameter space to Koopman space by encoder φ and back to the original space by decoder φ^{-1}. L_{recon} ensures consistency between θ and its reconstructed form via transformations:

$$L_{recon} = \sum\nolimits_{i=1}^{T} \|\theta_i, \varphi^{-1}(\varphi(\theta_i))\|_2. \tag{11}$$

2. **Dynamic Fidelity Loss:** This term ensures that the transformation produces a Koopman invariant subspaces in which the dynamics can forward correctly. It measures the fit of the Koopman operator $\mathcal{K}$ against the transferred parameter:

$$L_{dyna} = \sum\nolimits_{i=1}^{T} \sum\nolimits_{j=1}^{i} \|z_i, z_i^{j\to i}\|_2, \tag{12}$$

   where $z_i = \varphi(\theta_i)$ and $z_i^{j\to i} = z_j + \int_{t_j}^{t_i} \mathcal{K} z_\tau \, d\tau$.

3. **Consistency Loss:** It measures the consistency between the original and the dynamic parameter in the model space:

$$L_{consis} = \sum\nolimits_{i=1}^{T} \sum\nolimits_{j=1}^{i} \|\theta_i, \theta_i^{j\to i}\|_2, \quad \text{where } \theta_i^{j\to i} = \varphi^{-1}(z_i^{j\to i}) \tag{13}$$

Additionally, we load the dynamically integral parameters $\theta_i^{j\to i}$ back into the predictive model to evaluate its predictive capability, quantified by $L_{integ} = \sum_{i=1}^{T}\sum_{j=1}^{i}\mathcal{L}(Y_i, g(X_i;\theta_i^{j\to i}))$. Finally, the system optimizes the following combined loss to refine all components simultaneously:

$$\{\theta_{1:T}, \varphi, \varphi^{-1}, \mathcal{K}\} = \underset{\theta_{1:T},\varphi,\varphi^{-1},\mathcal{K}}{\arg min}\ (\alpha L_{intri} + \alpha L_{integ} + \beta L_{recon} + \gamma L_{dyna} + \beta L_{consis}), \quad (14)$$

with α, β, and γ as adjustable weights to balance the magnitude of each term, ensuring that no single term dominates during training.

During inference, given the moment t_s, the model uses the state from the nearest observation moment t_{obs} as an initial value, integrating over the time interval $[t_{obs}, t_s]$ in Koopman space to give the generalized model state $\theta_s^{obs\to s}$ at the desired test moment.

Analysis and Control of the Temporal Domain Generalization. Integrating Koopman theory into continuous temporal domain modeling facilitates the application of optimization, estimation, and control techniques, particularly through the spectral properties of the Koopman operator. We remark that the Koopman operator serves as a pivotal interface for analyzing and controlling the generalization process. Constraints imposed on the Koopman space will be equivalently mapped to the Model space. For instance, the eigenvalues of $\mathcal{K}$ are crucial as they provide insights into the system's stability and dynamics, as illustrated below:

$$z_i^{j\to i} = z_j + \int_{t_j}^{t_i} \mathcal{K} z_\tau \, d\tau, \quad \text{where } \lambda_i \text{ is an eigenvalue of } \mathcal{K} \quad (15)$$

1. **System Assessment.** The generalization process is considered stable if all λ_i satisfy $\mathrm{Re}(\lambda_i) < 0$. Conversely, the presence of any λ_i such that $\mathrm{Re}(\lambda_i) > 0$ indicates instability in the system. When $\mathrm{Re}(\lambda_i) = 0$, the system may exhibit oscillatory behavior. By analyzing the locations of these poles in the complex plane, we can assess the system's long-term dynamics, helping us identify whether the generalized model is likely to collapse in the future.
2. **Behavioral Constraints.** Adding explicit constraints to $\mathcal{K}$ can guide the generalization toward desired behaviors. This process not only facilitates the incorporation of prior knowledge about domains but also tailors the system to specific characteristics. To name just a few, if the data dynamics are known to be periodic, configuring $\mathcal{K}$ as $\mathcal{K} = B - B^T$, with B as learnable parameters, ensures that the model system exhibits consistent periodic behavior since the eigenvalues of $B - B^T$ are purely imaginary values. Additionally, employing a low-rank approximation such as $\mathcal{K} = UV^T$, with $U, V \in \mathbb{R}^{n\times k}$ and $k < n$, allows the model to concentrate on the most significant dynamical features and explicitly control the degrees of freedom of the generalization process.

Theoretical Analysis. In this work, we theoretically proved that our proposed continuous-time TDG method has a smaller or equal error compared to the discrete-time method for approximating temporal distribution drift. This demonstrates that *the ODE method provides a more accurate approximation due to its consideration of the integral of changes over time, reducing the accumulation of errors compared to the step-wise updates of the discrete-time methods.*

Theorem 2 (Superiority of continuous-time methods over discrete-time methods (informal))**.** *Continuous-time methods have smaller or equal errors compared to discrete-time methods in approximating temporal distribution drift, due to its consideration of the integral of changes over time.*

The formal version and proof of Theorem 2 are given in Appendix C. We also provide a detailed model complexity analysis in Appendix A.1.4.

5 Experiment

In this section, we present the performance of the Koodos in comparison to other approaches through both quantitative and qualitative analyses. Our evaluation aims at *1) assessing the generalizability of Koodos in continuous temporal domains; 2) assessing whether Koodos captures the correct underlying data dynamics*; and *3) assessing whether Koodos can use inductive bias to guide the behavior of the generalization*. Detailed experiment settings (i.e., dataset details, baseline details, hyperparameter settings, ablation study, scalability analysis, and sensitivity analysis) are demonstrated in Appendix A.1. Besides, since the TDG is a special case of the CTDG, we also conducted experiments on the traditional discrete temporal domain generalization task. Results are shown in Appendix B.

Table 1: Performance comparison on continuous temporal domain datasets. The classification tasks report Error rates (%) except for the AUC for the Twitter dataset. The regression tasks report MAE. 'N/A' implies that the method does not support the task.

Model	Classification				Regression	
	2-Moons	Rot-MNIST	Twitter	Yearbook	Cyclone	House
Offline	13.5 ± 0.3	$\underline{6.6 \pm 0.2}$	0.54 ± 0.09	8.6 ± 1.0	18.7 ± 1.4	19.9 ± 0.1
LastDomain	55.7 ± 0.5	74.2 ± 0.9	0.54 ± 0.12	11.3 ± 1.3	22.3 ± 0.7	20.6 ± 0.7
IncFinetune	51.9 ± 0.7	57.1 ± 1.4	0.52 ± 0.01	11.0 ± 0.8	19.9 ± 0.7	20.6 ± 0.2
IRM	15.6 ± 0.2	8.6 ± 0.4	0.53 ± 0.11	$\underline{8.3 \pm 0.5}$	18.0 ± 0.8	19.8 ± 0.2
V-REx	$\underline{12.8 \pm 0.2}$	8.6 ± 0.3	0.58 ± 0.05	8.9 ± 0.5	17.7 ± 0.5	20.2 ± 0.1
CIDA	18.7 ± 2.0	8.3 ± 0.7	0.63 ± 0.03	8.4 ± 0.8	$\underline{17.0 \pm 0.4}$	10.2 ± 1.0
TKNets	39.6 ± 1.2	37.7 ± 2.0	0.57 ± 0.04	8.4 ± 0.3	N/A	N/A
DRAIN	53.2 ± 0.9	59.1 ± 2.3	0.57 ± 0.04	10.5 ± 1.0	23.6 ± 0.5	$\underline{9.8 \pm 0.1}$
DRAIN-Δt	46.2 ± 0.8	57.2 ± 1.8	0.59 ± 0.02	11.0 ± 1.2	26.2 ± 4.6	9.9 ± 0.1
DeepODE	17.8 ± 5.6	48.6 ± 3.2	$\underline{0.64 \pm 0.02}$	13.0 ± 2.1	18.5 ± 3.3	10.7 ± 0.4
Koodos (Ours)	$\mathbf{2.8 \pm 0.7}$	$\mathbf{4.6 \pm 0.1}$	$\mathbf{0.71 \pm 0.02}$	$\mathbf{6.6 \pm 1.3}$	$\mathbf{16.4 \pm 0.3}$	$\mathbf{9.0 \pm 0.2}$

Datasets. We compare with classification datasets: Rotated Moons (2-Moons), Rotated MNIST (Rot-MNIST), Twitter Influenza Risk (Twitter), and Yearbook; and the regression datasets: Tropical Cyclone Intensity (Cyclone), House Prices (House). More details can be found in Appendix A.1.1.

Baselines. We employ three categories of baselines: **Practical baselines**, including 1) Offline; 2) LastDomain; 3) IncFinetune; 4) IRM [1]; 5) V-REx [25]. **Discrete temporal domain generalization methods**, including 1) CIDA [48]; 2) TKNets [57]; 3) DRAIN [2]; 4) DRAIN-Δt. **Continuous temporal domain generalization methods**, including 1) DeepODE [11]. Comparison method details can be found in Appendix A.1.2.

Metrics. Error rate (%) is used for classification tasks. As the Twitter dataset has imbalanced labels, the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve is used. Mean Absolute Error (MAE) is used for regression tasks. All models were trained on training domains and then deployed on all unseen test domains. Each method's experiments were repeated five times, with mean results and standard deviations reported. Detailed parameter settings for each dataset are provided in Appendix A.1.3.

5.1 Quantitative analysis: generalization across continuous temporal domains

We first present the performance of our proposed method against baseline methods, highlighting results from Table 1. Koodos exhibits outstanding generalizability across continuous temporal domains. A key observation is that all baseline models struggle to handle synthetic datasets, particularly challenged by the continuous and substantial concept drift (i.e., 18 degrees of rotation per second). In real-world datasets, methods like CIDA, DRAIN, and DeepODE demonstrate effectiveness in certain cases. However, the performance gap between them and Koodos highlights the importance of explicitly considering continuous temporal modeling. For instance, while DRAIN attempts to address domain dynamics through a probabilistic sequential approach via LSTM, this introduces considerable errors due to the inherent discrete temporal. Moreover, while DRAIN-Δt and DeepODE adjust for temporal irregularities accordingly, they fail to adequately synchronize the data and model dynamics, leading to unsatisfactory results. In contrast, Koodos establishes a promising approach and a benchmark in CTDG tasks, with quantitative analysis firmly confirming the approach's superiority.

5.2 Qualitative analysis: data dynamics and the learned model dynamics

We conducted a qualitative comparison of different models by visualizing their decision boundaries on the 2-Moons dataset, as depicted in Fig. 3. Each row represents a different method: DRAIN-Δt, DeepODE, and Koodos, with the timeline at the bottom tracking progress through test domains. DRAIN-Δt displays the highest error rate, showing substantial deviation from the anticipated trajectories, especially after the third domain. We also observe that DRAIN-Δt seems to freeze when predicting multiple steps, likely due to its underlying model, DRAIN, uses a recursive training strategy within a single domain and is explicitly designed for one-step prediction. DeepODE shows a relatively better performance. It benefits from leveraging ODEs to maintain continuity in the model dynamics. However, the nonlinear variation of the predictive model parameters complicates its ability to abstract and simplify the real dynamics. Its predictions start close to the desired paths but diverge over time. Finally, Koodos exhibits the highest performance with clear and concise boundaries,

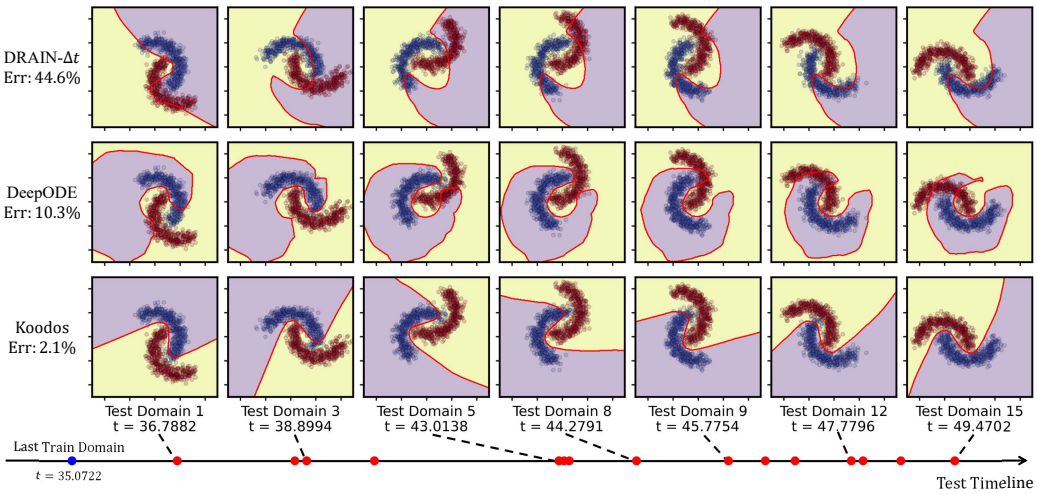
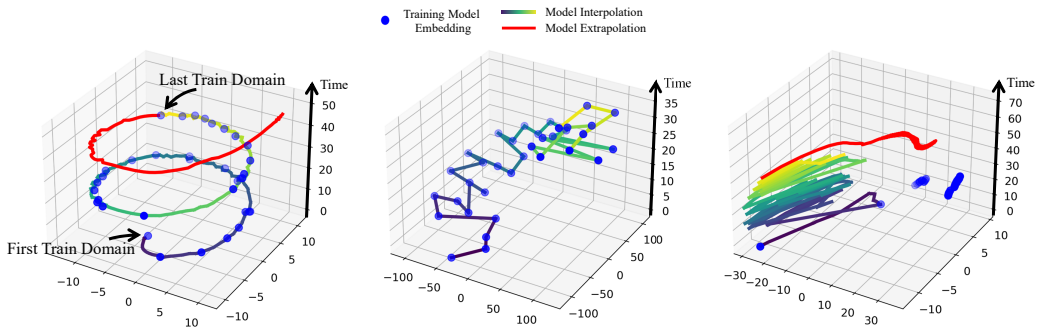


Figure 3: Visualization of decision boundary of the 2-Moons dataset (purple and yellow show data regions, red line shows the decision boundary). Top to bottom compares two baseline methods with ours; left to right shows partial test domains (all test domains are marked with red points on the timeline). All models are learned using data before the last train domain.



(a) Koodos (b) DRAIN (c) DeepODE

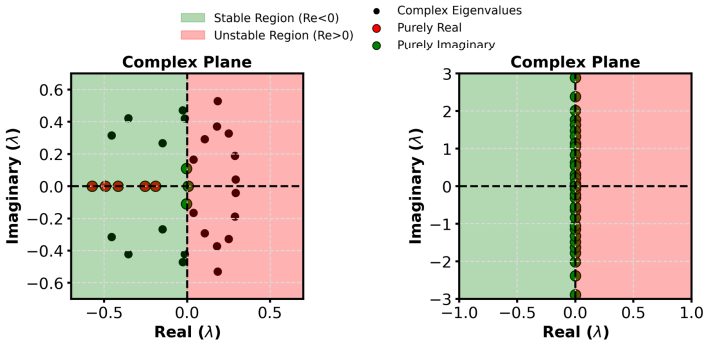
Figure 4: Interpolated and extrapolated predictive model trajectories. **Left**: Koodos captures the essence of generalization through the harmonious synchronization of model and data dynamics; **Middle**: DRAIN, as a probabilistic model, fails to capture continuous dynamics, which is presented as jumps from one random state to another. **Right**: DeepODE demonstrates a certain degree of continuity, but the dynamics are incorrect.

consistently aligning with the actual dynamics and maintaining high fidelity across all tested domains, showcasing its robustness in continuous temporal domain modeling and generalization.

Fig. 4(a) demonstrates the space-time evolution of the generalization process of Koodos. By applying t-SNE, the predictive model parameters are reduced to a 2-dimensional representation space, plotted against the time on the Z-axis. We used Koodos to interpolate the model states (35 seconds displayed in blue-yellow line) among the training domain states (marked by blue docs) in steps of 0.2 seconds, and similarly extrapolated 75 steps (15 seconds displayed in red line). The visualization clearly shows that Koodos synchronizes the model dynamics with the data dynamics: the interpolation creates a cohesive, upward-spiraling trajectory transitioning from the first to the last training domain, while the extrapolation correctly extends this trajectory outward into a new area, demonstrating the effective of Koodos from another intuitive way. We also show the space-time evolution of baseline models in Fig. 4(b,c), in which we do not find meaningful patterns of the generalization process.

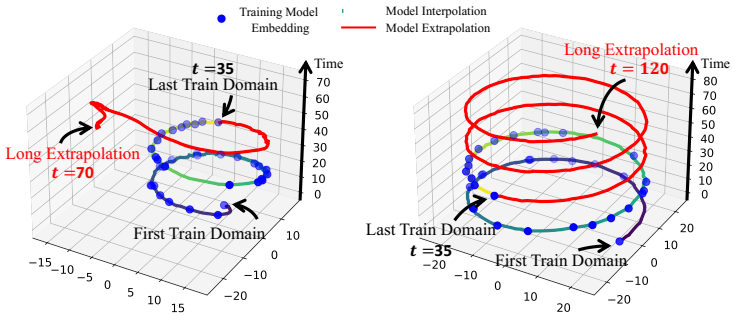
5.3 Analysis and control of the generalization process

The learned Koopman operator provides valuable insights into the dynamics of generalized predictive models. We analyze the behavior of the learned Koodos model on the 2-Moons dataset, focusing on the eigenvalues of the Koopman operator. As illustrated in Fig. 5(a), the eigenvalues are distributed



(a) Without Inductive Bias (b) With Inductive Bias

Figure 5: Eigenvalue distribution of the Koopman operator. **Left**: $\mathcal{K}$ as learnable; **Right**: $\mathcal{K} = B - B^T$ with B as learnable.



(a) Without Inductive Bias (b) With Inductive Bias

Figure 6: Extremely long-term extrapolated predictive model trajectories in uncontrolled and controlled settings. **Left**: $\mathcal{K}$ as learnable; **Right**: $\mathcal{K} = B - B^T$ with B as learnable.

across both stable (Re<0) and unstable (Re>0) regions on the complex plane. The spectral distribution suggests that while Koodos performs effectively across all tested domains, it will eventually diverge towards instability and finally collapse. To validate, we extended the extrapolation of the Koodos to an extremely long-term (i.e., 35 seconds future). Results depicted in Fig. 6(a) demonstrate that the generalized model's trajectory significantly deviates from the anticipated spiral path, suggesting that extremely long-term generalization will end up with the accumulation of errors.

Fortunately, Koodos's innovative design allows it to incorporate knowledge that extends beyond the observable data. By configuring the Koopman operator as $\mathcal{K} = B - B^T$, we ensure that all eigenvalues of the final learned $\mathcal{K}$ are purely imaginary (termed Koodos$^+$), promoting stable and periodic behavior. This adjustment is reflected in Fig. 5(b), where the eigenvalues are tightly clustered around the imaginary axis. As shown in Fig. 6(b), the embeddings and trajectories of Koodos$^+$ are cohesive and maintain stability over extended periods. Remarkably, even with 85 seconds of extrapolation, Koodos$^+$ shows no signs of performance degradation, highlighting the significance of human inductive bias in improving the robustness and reliability of the generalization process.

6 Conclusion

We tackle the problem of continuous temporal domain generalization by proposing a continuous dynamic system network, Koodos. We characterize the dynamics of the data and determine its impacts on model dynamics. The Koopman operator is further learned to represent the underlying dynamics. We also design a comprehensive optimization framework equipped with analysis and control tools. Extensive experiments show the efficiency of our design.

Acknowledgments and Disclosure of Funding

This work was supported by Japan Science and Technology Agency (JST) SICORP, Grant Number JPMJSC2104; JST SPRING, Grant Number JPMJSP2108; the SPRING GX Overseas Dispatch Program and the Shibasaki Scholarship Foundation.

References

[1] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[2] Guangji Bai, Chen Ling, and Liang Zhao. Temporal domain generalization with drift-aware dynamic neural networks. In *The Eleventh International Conference on Learning Representations*, 2023.

[3] Guangji Bai, Johnny Torres, Junxiang Wang, Liang Zhao, Cristina Abad, and Carmen Vaca. Sign-regularized multi-task learning. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pages 793–801. SIAM, 2023.

[4] Guangji Bai, Qilong Zhao, Xiaoyang Jiang, Yifei Zhang, and Liang Zhao. Saliency-guided hidden associative replay for continual learning. *arXiv preprint arXiv:2310.04334*, 2023.

[5] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. Metareg: Towards domain generalization using meta-regularization. *Advances in neural information processing systems*, 31, 2018.

[6] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[7] Steven L Brunton, Marko Budišić, Eurika Kaiser, and J Nathan Kutz. Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*, 2021.

[8] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[9] Zekun Cai, Renhe Jiang, Xinyu Yang, Zhaonan Wang, Diansheng Guo, Hill Hiroki Kobayashi, Xuan Song, and Ryosuke Shibasaki. Memda: Forecasting urban time series with memory-based drift adaptation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 193–202, 2023.

[10] Boyo Chen, Buo-Fu Chen, and Hsuan-Tien Lin. Rotation-blended cnns on a new open dataset for tropical cyclone image-to-intensity regression. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 90–99, 2018.

[11] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[12] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[13] Qi Dou, Daniel Coelho de Castro, Konstantinos Kamnitsas, and Ben Glocker. Domain generalization via model-agnostic learning of semantic features. *Advances in neural information processing systems*, 32, 2019.

[14] Yingjun Du, Jun Xu, Huan Xiong, Qiang Qiu, Xiantong Zhen, Cees GM Snoek, and Ling Shao. Learning to learn with variational information bottleneck for domain generalization. In *European Conference on Computer Vision*, pages 200–216, 2020.

[15] Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, Renjun Xu, and Chongjun Wang. Adarnn: Adaptive learning and forecasting for time series. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

[16] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[17] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

[18] Rui Gong, Wen Li, Yuhua Chen, and Luc Van Gool. Dlow: Domain flow for adaptation and generalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2477–2486, 2019.

[19] David Francis Griffiths and Desmond J Higham. *Numerical methods for ordinary differential equations: initial value problems*, volume 5. Springer, 2010.

[20] Huan He, Owen Queen, Teddy Koker, Consuelo Cuevas, Theodoros Tsiligkaridis, and Marinka Zitnik. Domain adaptation for time series under feature and label shifts. In *International Conference on Machine Learning*, pages 12746–12774. PMLR, 2023.

[21] Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *16th European Conference on Computer Vision, ECCV 2020*, pages 124–140. Springer, 2020.

[22] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[24] Bernard O Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[25] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). In *International conference on machine learning*, pages 5815–5826. PMLR, 2021.

[26] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, 2018.

[27] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. Sequential learning for domain generalization. In *European Conference on Computer Vision*, pages 603–619. Springer, 2020.

[28] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[29] Jingjing Li, Zhiqi Yu, Zhekai Du, Lei Zhu, and Heng Tao Shen. A comprehensive survey on source-free domain adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[30] Wen Li, Zheng Xu, Dong Xu, Dengxin Dai, and Luc Van Gool. Domain generalization and adaptation using low rank exemplar svms. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1114–1127, 2017.

[31] Wendi Li, Xiao Yang, Weiqing Liu, Yingce Xia, and Jiang Bian. Ddg-da: Data distribution generation for predictable concept drift adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4092–4100, 2022.

[32] Alexander H Liu, Yen-Cheng Liu, Yu-Ying Yeh, and Yu-Chiang Frank Wang. A unified feature disentangler for multi-domain image translation and manipulation. *Advances in neural information processing systems*, 31, 2018.

[33] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363, 2018.

[34] Massimiliano Mancini, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. Best sources forward: domain generalization through source-specific nets. In *2018 25th IEEE international conference on image processing (ICIP)*, pages 1353–1357. IEEE, 2018.

[35] Massimiliano Mancini, Samuel Rota Bulo, Barbara Caputo, and Elisa Ricci. Adagraph: Unifying predictive and continuous domain adaptation through graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6568–6577, 2019.

[36] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

[37] Saeid Motiian, Marco Piccirilli, Donald A Adjeroh, and Gianfranco Doretto. Unified deep supervised domain adaptation and generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5715–5725, 2017.

[38] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. Domain generalization via invariant feature representation. In *International conference on machine learning*, pages 10–18. PMLR, 2013.

[39] Anshul Nasery, Soumyadeep Thakur, Vihari Piratla, Abir De, and Sunita Sarawagi. Training for the future: A simple gradient interpolation loss to generalize along time. *Advances in Neural Information Processing Systems*, 34:19198–19209, 2021.

[40] Fengchun Qiao, Long Zhao, and Xi Peng. Learning to learn single domain generalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12556–12565, 2020.

[41] Tiexin Qin, Shiqi Wang, and Haoliang Li. Generalizing to evolving domains with latent structure-aware sequential autoencoder. In *International Conference on Machine Learning*, pages 18062–18082. PMLR, 2022.

[42] Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

[43] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. In *International conference on machine learning*, pages 2988–2997. PMLR, 2017.

[44] Mona Schirmer, Mazin Eltayeb, Stefan Lessmann, and Maja Rudolph. Modeling irregular time series with continuous recurrent units. In *International Conference on Machine Learning*, pages 19388–19405. PMLR, 2022.

[45] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[46] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[47] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017.

[48] Hao Wang, Hao He, and Dina Katabi. Continuously indexed domain adaptation. In *International Conference on Machine Learning*, pages 9898–9907. PMLR, 2020.

[49] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and S Yu Philip. Generalizing to unseen domains: A survey on domain generalization. *IEEE transactions on knowledge and data engineering*, 35(8):8052–8072, 2022.

[50] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[51] G Wilson and DJ Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology*, 11(5):1–46, 2020.

[52] Mixue Xie, Shuang Li, Longhui Yuan, Chi Liu, and Zehui Dai. Evolving standardization for continual domain generalization over temporal drift. *Advances in Neural Information Processing Systems*, 36, 2024.

[53] Zihao Xu, Guang-Yuan Hao, Hao He, and Hao Wang. Domain-indexing variational bayes: Interpretable domain index for domain adaptation. In *Eleventh International Conference on Learning Representations (ICLR)*, 2023.

[54] Huaxiu Yao, Caroline Choi, Bochuan Cao, Yoonho Lee, Pang Wei W Koh, and Chelsea Finn. Wild-time: A benchmark of in-the-wild distribution shift over time. *Advances in Neural Information Processing Systems*, 35:10309–10324, 2022.

[55] LIN Yong, Fan Zhou, Lu Tan, Lintao Ma, Jianmeng Liu, HE Yansu, Yuan Yuan, Yu Liu, James Y Zhang, Yujiu Yang, et al. Continuous invariance learning. In *The Twelfth International Conference on Learning Representations*, 2023.

[56] Dazhou Yu, Guangji Bai, Yun Li, and Liang Zhao. Deep spatial domain generalization. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 1293–1298. IEEE, 2022.

[57] Qiuhao Zeng, Wei Wang, Fan Zhou, Gezheng Xu, Ruizhi Pu, Changjian Shui, Christian Gagné, Shichun Yang, Charles X Ling, and Boyu Wang. Generalizing across temporal domains with koopman operators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16651–16659, 2024.

[58] Liang Zhao, Qian Sun, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan. Feature constrained multi-task learning models for spatiotemporal event forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 29(5):1059–1072, 2017.

A Appendix

A.1 Experimental details

A.1.1 Dataset details

In this paper, we explore a variety of datasets to analyze the performance of machine learning models under conditions of continuous temporal domain. We employ the following datasets:

1. **Rotated 2-Moons** This dataset is a variant of the classic 2-entangled moons dataset, where the lower and upper moon-shaped clusters are labeled 0 and 1, and each contains 500 instances. We randomly generated 50 distinct timestamps from the continuous real number time interval $[0, 50]$. Each timestamp corresponds to a specific domain, with each domain created by rotating the moons 18° counterclockwise per unit of time. *The continuous concept drift is represented by the progressive positional changes of the moon-shaped clusters.*
2. **Rotated MNIST** This dataset is a variant of the classic MNIST dataset [12], comprising images of handwritten digits from 0 to 9. We randomly generated 50 distinct timestamps from the continuous real number time interval $[0, 50]$. Each timestamp corresponds to a specific domain, within which we randomly selected 1,000 images from the MNIST dataset and rotated them 18° counterclockwise per unit of time. *Similar to 2-Moons, the continuous concept drift means the rotation of the images.*
3. **Twitter** The Twitter dataset [58; 3] utilizes tweet data to predict flu intensity. We randomly collected streaming tweets starting at 50 arbitrary timestamps lasting 7 days during the 2010-2014 flu seasons and then examined the volume of disease-related terms to predict current flu trends. The result is validated against the Influenza-Like Illness (ILI) activity levels reported by the Centers for Disease Control and Prevention (CDC). *The continuous concept drift in this dataset is characterized by the fluctuations in tweet volumes over the years and the variation in flu activity throughout the season.*
4. **Yearbook** The Yearbook dataset [54] consists of frontal-facing yearbook portraits collected between 1930 and 2013 from 128 high schools. The task is to classify gender from images. We randomly sampled 40 years of data from the 84-year dataset, with each year representing a domain to represent the incomplete temporal domain collection process, which resulted in variable time intervals between consecutive domains. *The Yearbook dataset provides a visual record of evolving fashion styles and social norms across the decades.*
5. **Cyclone** The Cyclone dataset [10] is collected by satellite remote sensing and is dedicated to the task of tropical cyclone imagery to wind intensity. When each cyclone occurs, the satellite collects a series of images for its entire life cycle as a domain, with the date of its occurrence representing a temporal domain time. We focused on cyclone data from the West Pacific region covering 2014 to 2016 and formed 72 continuous domains. *The dataset is event-triggered, and the variation in wind strength associated with the seasonal dates results in a continuous concept drift.*
6. **House** This dataset comprises housing price data from 2013 to 2019 and is utilized for a regression task to predict the price of a house given the features. We extracted sales data for one-month durations across 40 arbitrary time periods from 2013 through 2019. *The concept drift in this dataset is how the housing price changes over time for a certain region.*

To more clearly illustrate the temporal arbitrariness of the continuous temporal domain, we plot the time distribution of all domains for each dataset in Fig. 7, where the blue line represents the timestamp of one specific domain. We use the last 30% domain of each dataset as test domains, which are marked with gray shading.

A.1.2 Comparison methods

1. **Practical Baseline** *(1) Offline Model*: This model operates without temporal variations and is trained using Empirical Risk Minimization (ERM) across all source domains. *(2) LastDomain*: This model is also trained without temporal variations but focuses solely on the last source domain using ERM. *(3) IncFinetune*: This approach begins by applying the baseline method at the initial time point and subsequently fine-tunes the model at each following time point using a lower learning rate. *(4) IRM*: IRM is to train a predictive model

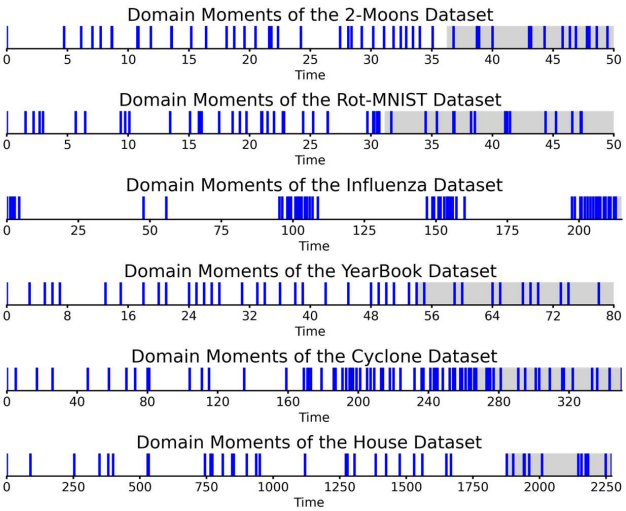


Figure 7: The distribution of continuous temporal domain datasets along the time dimension. The test domains are marked with gray shading.

over all distributions that do not rely on spurious correlations as much as possible. *(5) V-REx*: Like IRM, V-REx performs causal identification, while also providing some robustness to changes in the covariate shift by risk extrapolation.

2. **Discrete Temporal Domain Generalization Baseline** Continuous temporal domain tasks require consideration of domains for the long future rather than a limited number of steps. Existing methods are mainly unable to deal with this problem. Among them, although DRAIN [2] is designed only for predicting one-step future prediction, its internal LSTM structure has the potential of long-term future prediction, so we implemented DRAIN as a discrete baseline. We implemented two versions of it: *(1) DRAIN*: This approach tackles the temporal domain generalization problem by proposing a dynamic neural network, which uses an LSTM to capture the evolving pattern of a neural network, during the test, DRAIN will consider all future test domains as one domain, and utilizes the LSTM to predict the future parameters of this target domain. *(2) DRAIN-*Δt: A variant of DRAIN, in which the internal LSTM is changed to a continues recurrent units [44] to model irregular distributed domains. Under this setting, DRAIN-Δt treats the test domains as an irregular time series and predicts the model state at that given moment. We also added CIDA [48] and TKNets [57] as comparisons. *(3) CIDA* treats the domain index as a continuous variable to help the discriminator using a distance-based loss. *(4) TKNets* trains a prediction model for the target domain by leveraging the evolving pattern among domains.
3. **Continuous Temporal Domain Generalization Baseline** There is currently no method to generalize the model over the continuous time domain. However, we constructed a method that can infer continuous states using a neural dynamical system. *(1) DeepODE*: We train a deep neural Ordinary Differential Equations (NeuralODEs) [11] to model the continuous dynamics of the predictive model parameters. DeepODE consists of three structures. It first uses an encoder to embed the model parameters as dense vectors, then uses NeuralODEs to infer their state in the desired future, and later uses a decoder to transfer the state back to the parameters. DeepODE treats the first domain as an initial value and infers the state of future training domains at once, using the inferred model's performance on training domains to optimize its autoencoder and NeuralODEs. In the testing phase, when a specific moment is given, we use DeepODE to infer the prediction model parameters at that moment, and then reload the estimated parameters into the prediction model for prediction.

A.1.3 Model configuration

We use autoencoder to achieve the φ and φ^{-1}, which we call Encoder and Decoder in the following. A linear transformation layer without bias (named Dynamic Model) is served as the Koopman operator. In certain complex data dynamics, the Koopman operator needs to expand its dimensions (potentially up to infinity) for a better approximation of the dynamics. Since it's not feasible to increase the

dimensions endlessly, we approximate the infinitely wide Koopman operator using multiple linear layers with ReLU. In practice, we found that two linear layers work well enough. Additionally, Predictive Models for each domain are trained to provide initial parameter values for Generalized Models. We use NeuralODEs [11] as the ODEs solvers. For large models with massive parameters, we treat the head network as a feature extractor shared by all domains, inferring only the state of the remaining layers. All experiments are conducted on a 64-bit machine with two 20-core Intel Xeon Silver 4210R CPU @ 2.40GHz, 378GB memory, and four NVIDIA GeForce RTX 3090. We use Adam Optimizer for all experiments, and we specify the architecture as well as other details for each dataset as follows:

1. **Rotated 2-Moons** The Predictive Model consists of 3 hidden layers, with a dimension of 50 each. We use a ReLU layer after each layer and a Sigmoid layer after the output layer. The Encoder and Decoder are both a 4-layer Multi-Layer Perceptron (MLP), with dimensions of $[1024, 512, 128, 32]$ for each layer. A 32-dimensional linear layer Dynamic Model serves as the Koopman operator. The learning rate for the Predictive Model is set at 1×10^{-2}, while for the others is set at 1×10^{-3}.
2. **Rotated MNIST** The Prediction Model features a convolutional neural network architecture comprising three convolutional layers with channel $[32, 32, 64]$ and a kernel size of 3. Each convolutional layer is followed by a ReLU layer and a max pooling layer with a kernel size of 2. Following the flattening, the network includes two linear layers with dimensions $[128, 10]$. A dropout layer is added between the linear layers to prevent overfitting. We treat the convolutional layer as the feature extractor. The Encoder and Decoder are both a 4-layer MLP with dimensions of $[1024, 512, 128, 32]$. A 32-dimensional two linear layers Dynamic Model serves as the Koopman operator. The learning rate for all parts is set at 1×10^{-3}.
3. **Twitter** The Prediction Model consists of 3 hidden layers, with a dimension of $[128, 32, 1]$. We use the ReLU layer after each layer and a Sigmoid layer after the output layer. The Encoder and Decoder are both a 4-layer MLP, with dimensions of $[1024, 512, 128, 32]$ for each layer. A 32-dimensional two linear layers Dynamic Model serves as the Koopman operator. The learning rate for all parts is set at 1×10^{-3}.
4. **Yearbook** The Prediction Model features a convolutional neural network architecture comprising three convolutional layers with channel $[32, 32, 64]$ and a kernel size of 3. Each convolutional layer is followed by a ReLU layer and a max pooling layer with a kernel size of 2. Following the flattening, the network includes three hidden linear layers with dimensions $[128, 32, 1]$. A dropout layer is added between the linear layers to prevent overfitting. The convolutional layer is used as the feature extractor. The Encoder and Decoder are both a 4-layer MLP with dimensions of $[1024, 512, 128, 32]$. A 32-dimensional two linear layers Dynamic Model serves as the Koopman operator. The learning rate is set at 1×10^{-3}.
5. **Cyclone** The Prediction Model features a convolutional neural network architecture comprising four convolutional layers with channel $[32, 32, 64, 64]$ and a kernel size of 3. Each convolutional layer is followed by a ReLU layer and a max pooling layer with a kernel size of 2. Following the flattening, the network includes three hidden linear layers with dimensions $[128, 32, 1]$. We treat the convolutional layer as the feature extractor. The Encoder and Decoder are both a 4-layer MLP, with dimensions of $[1024, 512, 128, 32]$ for each layer. A 32-dimensional 2 linear layers Dynamic Model serves as the Koopman operator. The learning rate for all parts is set at 1×10^{-3}.
6. **House** The Prediction Model consists of 3 hidden layers with a dimension of 400. We use the ReLU layer after each layer. The Encoder and Decoder are both a 4-layer MLP with dimensions of $[1024, 512, 128, 32]$. A 32-dimensional two linear layers Dynamic Model serves as the Koopman operator. The learning rate for all parts is set at 1×10^{-3}.

A.1.4 Complexity analysis

In our framework, coordinate transformations between spaces are implemented using an autoencoder achieved by MLP. The complexity of these transformations is described by a linear relationship in terms of N, expressed as $\mathcal{O}(2(Nd + E) + F)$, where N denotes the number of parameters θ in the predictive models, d denotes the number of neurons in the first MLP layer, E denotes for the number of parameters remaining in the autoencoder, and F denotes the number of parameters in the Koopman operator. In many deep learning tasks, significant portions of the model's layers function as representation learning and can be shared, resulting in a small N and controlled model complexity.

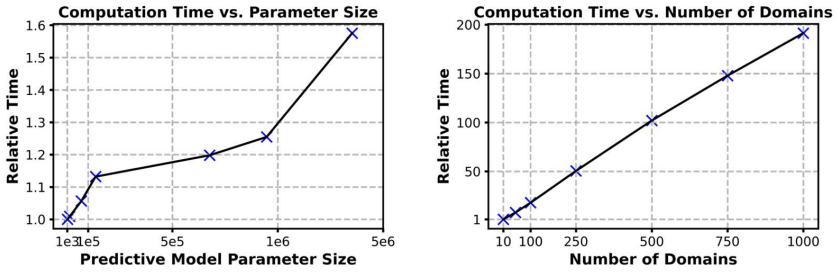
Table 2: Ablation test results for different datasets.

Ablation	2-Moons	Rot-MNIST	Cyclone
✗L_{integ}	27.4 ± 20.3	42.7 ± 1.0	55.0 ± 1.0
✗L_{recon}	3.2 ± 0.1	6.3 ± 0.3	17.2 ± 0.4
✗L_{dyna}	5.9 ± 5.1	31.7 ± 28.9	23.3 ± 4.9
✗L_{consis}	30.5 ± 18.4	5.1 ± 0.4	17.4 ± 0.5
✗$K.S$	37.2 ± 7	OOM	OOM
Koodos	**2.8** ± **0.7**	**4.6** ± **0.1**	**16.4** ± **0.3**

A.1.5 Ablation study

We conducted an ablation study to systematically assess the contribution of various constraints and components within Koodos on two classification datasets: 2-Moons, Rot-MNIST, and a regression dataset Cyclone. The results are summarized in the Table 2. For each test, specific loss constraints were removed, indicated by ✗, and we also tested the effect of bypassing the Koopman Space but learning dynamics directly in the parameter space, designated as ✗K.S.

As shown in the table, each component of constraints contributes to robust performance across all tested datasets. The removal of particular elements has marked effects on model stability and accuracy. Specifically, the absence of L_{integ} and L_{dyna} constraints leads to significant performance degradation, highlighting their essential roles in learning the correct dynamics of the model through continuous temporal domain data. Removing other constraints leads to unpredictable results on certain datasets or a decline in effectiveness, indicating their importance in ensuring system effectiveness and stability under various input conditions. The considerable increase in error observed with the ✗K.S on the 2-Moons dataset demonstrates the challenges associated with modeling nonlinear dynamics directly in the parameter space, and the out-of-memory (OOM) problems with Rot-MNIST and Cyclone upon the removal of the Koopman Space suggest that this component is not only pivotal but also computationally demanding.



(a) Scalability for Parameter Size (b) Scalability for Domain Number

Figure 8: Scalability analysis w.r.t the number of parameters and the number of domains.

Table 3: Cost of training and testing time.

Model	Train Time (s)	Test Time (s)
DRAIN	67	<1
DeepODE	289	1
CIDA	610	3.05
TKNets	948	2
Koodos	566	2.83

A.1.6 Scalability analysis

We conducted a comprehensive analysis to evaluate the scalability of our system concerning both the number of parameters of the predictive model and the number of domains, as shown in Fig. 8. Computational times are normalized relative to the shortest time to provide a consistent basis.

To explore how the size of predictive model parameters affects runtime, we experimented with various configurations by varying the depth and number of neurons in the model. These configurations were tested on the Cyclone dataset with parameter counts ranging from 2,000 to over one million. As depicted in Fig. 8(a), there is a relatively linear and low growth rate in computation time as the parameter size increases. This gradual increase suggests that the autoencoder in Koodos to encode the parameter space into a Koopman space effectively mitigates the impact of increased parameter scale, maintaining computational efficiency even as predictive model complexity grows.

To assess the effect of domain count on runtime, we generated synthetic 2-Moons datasets with varying domain numbers: 10, 50, 100, 250, 500, 750, and 1000 domains, each with 200 training instances categorized into two labels. The runtime is plotted against the number of domains in Fig. 8(b), demonstrating a linear increase in computational time. This linear progression aligns with expectations for a system processing a sequence of input domains through ODEs, indicating predictable and manageable scalability as domain count increases.

We further conduct the model scalability analysis by comparing the running time of Koodos with other state-of-the-art baselines: DRAIN, DeepODE, CIDA, and TKNets in the 2-Moons dataset. As shown in Table 3, our model strikes a good balance between training time and effectiveness.

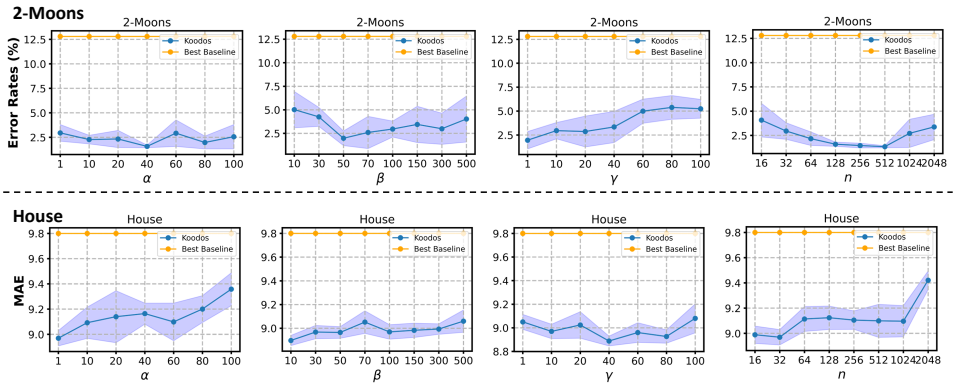


Figure 9: Sensitivity analysis.

A.1.7 Sensitivity analysis

We conducted the sensitivity analysis on 2-Moons and House datasets to understand the hyperparameters in Koodos: loss weights (α, β, γ) and the dimensions n of the Koopman operator $\mathcal{K}$. The loss weights are set based on the magnitude of each loss term. For instance, in the 2-Moons dataset, the cross-entropy loss L_{intri} and L_{integ} are approximately 1 after convergence, the L_{recon} and L_{consis} in the Model space are around 0.01, and L_{dyna} in the Koopman space is around 0.1. Accordingly, we set the initial values of α, β, γ to 1, 100, and 10, respectively. We then adjust each term independently within its respective range: α and γ are varied from 1 to 100, and β from 10 to 1000. The dimension n of the Koopman operator varies from 16 to 2048. Fig. 9 shows the results.

It can be seen that: (1) Koodos exhibits stable behavior with controlled variance across a wide range of hyperparameter values, evidencing its insensitivity to hyperparameter variations. (2) Setting the loss weights of (α, β, γ) by the magnitude of each loss term is sufficient for achieving good performance, and fine-tuning the weights may give better results. (3) A hundreds-dimension approximation of the Koopman operator is sufficient to achieve good results. Too high may lead to overfitting.

A.1.8 Limitations

The limitation arises from the assumptions inherent in the research area of the temporal domain generalization framework. TDG assumes that domain pattern drifts follow certain predictable, smooth patterns, allowing for modeling future changes as a sequence. CTDG extends TDG to continuous time, further generalizing the problem to handle domains collected at arbitrary times. Thus, their assumptions are aligned in focusing on smooth, predictable concept drift. To address other kinds of drift, alternative frameworks may be required. However, different frameworks can be combined to address the full spectrum of domain generalization challenges. Exploring such a comprehensive approach may represent a promising research direction.

Table 4: Performance comparison on discrete temporal domain datasets. The classification tasks report Error rates (%), and the regression tasks report MAE.

Model	Classification				Regression	
	2-Moons	Rot-MNIST	ONP	Shuttle	House	Appliance
Offline	22.4 ± 4.6	18.6 ± 4.0	$\underline{33.8 \pm 0.6}$	0.77 ± 0.1	11.0 ± 0.36	10.2 ± 1.1
LastDomain	14.9 ± 0.9	17.2 ± 3.1	36.0 ± 0.2	0.91 ± 0.18	10.3 ± 0.16	9.1 ± 0.7
IncFinetune	16.7 ± 3.4	10.1 ± 0.8	34.0 ± 0.3	0.83 ± 0.07	9.7 ± 0.01	8.9 ± 0.5
CDOT	9.3 ± 1.0	14.2 ± 1.0	34.1 ± 0.0	0.94 ± 0.17	-	-
CIDA	10.8 ± 1.6	9.3 ± 0.7	34.7 ± 0.6	-	9.7 ± 0.06	8.7 ± 0.2
GI	3.5 ± 1.4	7.7 ± 1.3	36.4 ± 0.8	0.29 ± 0.05	9.6 ± 0.02	8.2 ± 0.6
DRAIN	$\underline{3.2 \pm 1.2}$	$\underline{7.5 \pm 1.1}$	38.3 ± 1.2	$\underline{0.26 \pm 0.05}$	$\underline{9.3 \pm 0.14}$	$\underline{6.4 \pm 0.4}$
Koodos (Ours)	$\mathbf{1.3 \pm 0.4}$	$\mathbf{7.0 \pm 0.3}$	$\mathbf{33.5 \pm 0.4}$	$\mathbf{0.24 \pm 0.04}$	$\mathbf{8.8 \pm 0.19}$	$\mathbf{4.8 \pm 0.3}$

B Experiments on discrete temporal domain generalization

While our primary focus is on Continuous Temporal Domain Generalization (CTDG), we have also conducted experiments in the more conventional context of Temporal Domain Generalization (TDG). In essence, TDG can be seen as a specialized case of CTDG, where making the continuous system with a step size fixed at 1. Such a setting does not affect the proper ability of the CTDG model to capture the data dynamics and the corresponding model dynamics.

In TDG experiments, we follow the problem definition in DRAIN [2]. Formally, a sequence of data domains is collected over discrete time intervals. Each domain, denoted as $\mathcal{D}_t$, corresponds to a dataset collected at a specific time t where $t = 1, 2, ..., T$. We have a sequence of domains $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_T\}$, where each domain $\mathcal{D}_t = \{(x_i^t, y_i^t)\}_{i=1}^{N_t}$ consists of N_t instances with features $x_i^t \in X_t$ and labels $y_i^t \in Y_t$, and X_t and Y_t represent the random variables for features and targets. The goal in TDG is to train a predictive model $g(X_t; \theta_t)$ on these historical domains, and modeling the dynamics of $\{\theta_1, \theta_2, ..., \theta_T\}$, then predict the θ_{T+1} for the unseen future domain $\mathcal{D}_{T+1}$ at time $t = T + 1$, finally do the prediction task $Y_{T+1}^* = g(X_{T+1}; \theta_{T+1})$.

We have replicated the experimental setups typically used in TDG research. Specifically, we used datasets, settings, and hyperparameters from the DRAIN [2]. We compare with the following classification datasets: Rotated Moons (2-Moons), Rotated MNIST (Rot-MNIST), Online News Popularity (ONP), and Shuttle; and the following regression datasets: House prices dataset (House), Appliances energy prediction dataset (Appliance). Note that the first two datasets differ from datasets used in our CTDG main experiments in Exp. 5, as here they are time-regularly distributed and only have 1-step future domain to predict. We keep the architecture of the predictive model for each dataset the same as DRAIN [2].

Performance comparison of all methods in terms of misclassification Error (in %) for classification tasks and mean absolute error (MAE) for regression tasks. All experiments are repeated 5 times for each method, and we report the average results and the standard deviation in the quantitative analysis.

The results are shown in Table 4. The Koodos model consistently outperforms all baselines across various datasets, demonstrating its robustness and superior adaptability also made to traditional temporal domain generalization tasks. Notably, Koodos significantly reduces error margins, especially in real-world datasets such as House and Appliance. The exceptional performance of Koodos can be attributed to its fundamental continuous dynamic system design, which effectively captures and synchronizes dynamic changes in data and models.

C Theorem proofs

Assumptions

1. **Continuous Data Distribution Drift**: - Let $p(t)$ be the data distribution at time t. - Assume $p(t)$ is continuously differentiable, and the drift $\frac{dp(t)}{dt}$ is Lipschitz continuous with constant L.

2. **Training and Target Domains**: - The model is trained on a sequence of training domains $\{p(t_i)\}_{i=1}^{T}$ where $t_1 < t_2 < \ldots < t_T$. - The target domain is at a future time s, with the data distribution $p(s)$. - The time intervals $\Delta t_i = t_{i+1} - t_i$ can be arbitrary.

3. **Model Training**: - For the ODE model, let $x(t)$ be the state at time t, governed by the ODE:

$$\frac{dx(t)}{dt} = f(x(t), t),$$

where f is a smooth function. - For the RNN model, let h_{t_i} be the state at discrete time t_i, updated by:

$$h_{t_{i+1}} = h_{t_i} + \Delta t_i \cdot \phi(h_{t_i}, x_{t_i}),$$

where ϕ is a smooth function and x_{t_i} is the input at time t_i.

4. **Generalization Error**: - The generalization error on the target domain $p(s)$ depends on the accumulated training errors and the error propagation from the last training domain $p(t_T)$ to the target domain $p(s)$.

Theorem 3 (Formal version of Theorem 2)**.** *Given the assumptions, a continuous-time method using an Ordinary Differential Equation (ODE) provides a smaller or equal generalization error on an unseen target domain compared to a discrete-time method using a Recurrent Neural Network (RNN), due to its more accurate approximation of the data distribution drift over time.*

Proof. Below is the formal proof:

Accumulated Training Error

1. **ODE Model**: - The state $x(t + \Delta t_i)$ can be approximated by the ODE integral:

$$x(t_i + \Delta t_i) = x(t_i) + \int_{t_i}^{t_i + \Delta t_i} f(x(\tau), \tau)\, d\tau. \tag{16}$$

- Using the Taylor series expansion for $f(x(t), t)$:

$$x(t_i + \Delta t_i) = x(t_i) + f(x(t_i), t_i)\Delta t_i + \frac{(\Delta t_i)^2}{2}\frac{\partial f}{\partial t} + \mathcal{O}((\Delta t_i)^3). \tag{17}$$

- The error for one step is:

$$\text{Error}_{\text{ODE, step}} = \mathcal{O}((\Delta t_i)^2). \tag{18}$$

- The accumulated error over T steps is:

$$\text{Error}_{\text{ODE, train}} = \sum_{i=1}^{T-1} \mathcal{O}((\Delta t_i)^2). \tag{19}$$

- Given that the intervals Δt_i can be large, the accumulated error becomes:

$$\text{Error}_{\text{ODE, train}} = \mathcal{O}\left(\sum_{i=1}^{T-1} (\Delta t_i)^2\right). \tag{20}$$

2. **RNN Model**: - The state $h_{t_{i+1}}$ is updated as:

$$h_{t_{i+1}} = h_{t_i} + \Delta t_i \cdot \phi(h_{t_i}, x_{t_i}). \tag{21}$$

- Assuming $h_{t_i} \approx x(t_i)$ and $\phi(h_{t_i}, x_{t_i}) \approx f(x(t_i), t_i)$, the error for one step is:

$$\text{Error}_{\text{RNN, step}} = \mathcal{O}(\Delta t_i). \tag{22}$$

- The accumulated error over T steps is:

$$\text{Error}_{\text{RNN, train}} = \sum_{i=1}^{T-1} \mathcal{O}(\Delta t_i). \tag{23}$$

- Given that the intervals Δt_i can be large, the accumulated error becomes:

$$\text{Error}_{\text{RNN, train}} = \mathcal{O}\left(\sum_{i=1}^{T-1} \Delta t_i\right). \tag{24}$$

Error Propagation to Target Domain

1. ODE Model: - To handle an arbitrary future time s, we integrate the error over many small steps from t_T to s:

$$\text{Error}_{\text{ODE, target}} \approx \text{Error}_{\text{ODE, train}} + \int_{t_T}^{s} \mathcal{O}((\tau - t_T)^2)\, d\tau. \tag{25}$$

- Since τ ranges from t_T to s:

$$\text{Error}_{\text{ODE, target}} \approx \text{Error}_{\text{ODE, train}} + \mathcal{O}((s - t_T)^3). \tag{26}$$

2. RNN Model: - Similarly, the error propagation to the target domain s involves many small steps:

$$\text{Error}_{\text{RNN, target}} \approx \text{Error}_{\text{RNN, train}} + \sum_{k=0}^{K-1} \mathcal{O}(\Delta t_k), \tag{27}$$

- where K is the number of small steps from t_T to s and Δt_k are the small intervals:

$$\text{Error}_{\text{RNN, target}} \approx \text{Error}_{\text{RNN, train}} + \mathcal{O}(s - t_T). \tag{28}$$

Generalization Error Comparison

- For the ODE model:

$$\text{Error}_{\text{ODE, target}} \approx \mathcal{O}\left(\sum_{i=1}^{T-1} (\Delta t_i)^2\right) + \mathcal{O}((s - t_T)^3). \tag{29}$$

- For the RNN model:

$$\text{Error}_{\text{RNN, target}} \approx \mathcal{O}\left(\sum_{i=1}^{T-1} \Delta t_i\right) + \mathcal{O}(s - t_T). \tag{30}$$

- Since $(\Delta t_i)^2$ is much smaller than Δt_i for any Δt_i, and $(s - t_T)^3$ is much smaller than $s - t_T$, we have:

$$\text{Error}_{\text{ODE, target}} \leq \text{Error}_{\text{RNN, target}}. \tag{31}$$

□

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect our paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have included a section to discuss our paper's limitation in Appendix A.1.8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provided all assumptions and proofs for theories in our paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provided comprehensive information as well as the source code of our method.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provided open-access data and code to reproduce our experiments.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provided the necessary details to run our experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We are closely following existing literature in reporting the experimental results and error bars.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided the necessary details on the computing resources we used in our experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no potential negative societal impact found in this work. Our research is foundational research and not tied to particular applications.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly credited and cited licenses of existing assets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provided proper documents for our released code and data.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our research does not involve these.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.