Meta-Learning Dynamics Forecasting Using Task Inference

Rui Wang*
University of California San Diego

Robin Walters*
Northeastern University

Rose Yu
University of California San Diego

Abstract

Current deep learning models for dynamics forecasting struggle with generalization. They can only forecast in a specific domain and fail when applied to systems with different parameters, external forces, or boundary conditions. We propose a model-based meta-learning method called `DyAd` which can generalize across heterogeneous domains by partitioning them into different tasks. `DyAd` has two parts: an encoder which infers the time-invariant hidden features of the task with weak supervision, and a forecaster which learns the shared dynamics of the entire domain. The encoder adapts and controls the forecaster during inference using adaptive instance normalization and adaptive padding. Theoretically, we prove that the generalization error of such procedure is related to the task relatedness in the source domain, as well as the domain differences between source and target. Experimentally, we demonstrate that our model outperforms state-of-the-art approaches on forecasting complex physical dynamics including turbulent flow, real-world sea surface temperature and ocean currents. We open-source our code at `https://github.com/Rose-STL-Lab/Dynamic-Adaptation-Network`.

1 Introduction

Modeling dynamical systems with deep learning has shown great success in a wide range of systems from climate science, Internet of Things (IoT) to infectious diseases [23, 64, 8, 33, 42, 22]. However, the main limitation of previous works is limited generalizability. Most approaches only focus on a specific system and train on past data in order to predict the future. Thus a new model must be trained to predict a system with different dynamics. Consider, for example, learning fluid dynamics; shown in Fig. 1 are two fluid flows with different degrees of turbulence. Even though the flows are governed by the same equations, the difference in buoyant forces would require two separate deep learning models to forecast. Therefore, it is imperative to develop *generalizable* deep learning models for dynamical systems that can learn and predict well over a large heterogeneous domain.

Meta-learning [63, 7, 13], or learning to learn, improves generalization by learning multiple tasks from the environment. Recent developments in meta-learning have been successfully applied to few-shot classification [46, 62], active learning [77], and reinforcement learning [18, 20]. However, meta-learning in the context of forecasting high-dimensional physical dynamics has not been really studied before. The challenges with meta-learning dynamical sys-

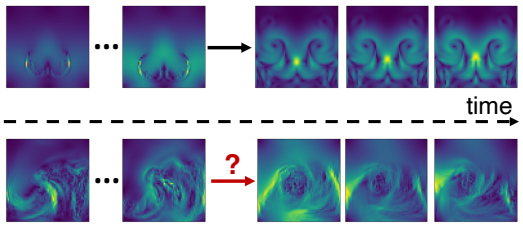

Figure 1: Meta-learning dynamic forecasting on turbulent flow. The model needs to generalize to a flow with a very different buoyant force.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

tems are unique in that (1) we need to efficiently infer the latent representation of the dynamical system given observed time series data, (2) we need to account for changes in unknown initial and boundary conditions, and (3) we need to model the temporal dynamics across heterogeneous domains.

Our approach is inspired by the fact that similar dynamical systems may share time-invariant hidden features. Even the slightest change in these features may lead to vastly different phenomena. For example, in climate science, fluids are governed by a set of differential equations called Navier-Stokes equations. Some features such as kinematic viscosity and external forces (e.g. gravity), are time-invariant and determine the flow characteristics. By inferring this latent representation, we can model diverse system behaviors from smoothly flowing water to atmospheric turbulence.

Inspired by neural style transfer [26], we propose a model-based meta-learning method, called `DyAd`, which can rapidly adapt to systems with varying dynamics. `DyAd` has two parts, an encoder g and a forecaster f. The encoder maps different dynamical systems to time-invariant hidden features representing constants of motion, boundary conditions, and external forces which characterize the system. The forecaster f then takes the learnt hidden features from the encoder and the past system states to forecast the future system state. Controlled by the time-invariant hidden features, the forecaster has the flexibility to adapt to a wide range of systems with heterogeneous dynamics.

Unlike gradient-based meta-learning techniques such as MAML [13], `DyAd` automatically adapts during inference using an encoder and does not require any retraining. Similar to model-based meta-learning methods such as MetaNets [46], we employ a two-part design with an adaptable learner which receives task-specific weights. However, for time series forecasting, since input and output come from the same domain, a support set of labeled data is unnecessary to define the task. The encoder can infer the task directly from query input.

Our contributions include:

- A novel model-based meta-learning method (`DyAd`) for dynamic forecasting in a large heterogeneous domain.
- An encoder capable of extracting the time-invariant hidden features of a dynamical system using time-shift invariant model structure and weak supervision.
- A new adaptive padding layer (AdaPad), designed for adapting to boundary conditions.
- Theoretical guarantees for `DyAd` on the generalization error of task inference in the source domain as well as domain adaptation to the target domain.
- Improved generalization performance on heterogeneous domains such as fluid flow and sea temperature forecasting, even to new tasks outside the training distribution.

2 Methods

2.1 Meta-learning in dynamics forecasting

Let $\mathbf{x} \in \mathbb{R}^d$ be a d-dimensional state of a dynamical system governed by parameters ψ. The problem of dynamics forecasting is that given a sequence of past states $(\mathbf{x}_1, \ldots, \mathbf{x}_t)$, we want to learn a map f such that: $f : (\mathbf{x}_1, \ldots, \mathbf{x}_t) \longmapsto (\mathbf{x}_{t+1}, \ldots \mathbf{x}_{t+k})$.

Here l is the length of the input series, and h is the forecasting horizon in the output. Existing approaches for dynamics forecasting only predict future data for a specific system as a single task. Here a task refers to forecasting for a specific system with a given set of parameters. The resulting models often generalize poorly to different system dynamics. Thus a new model must be trained to predict for each specific system.

To perform meta-learning, we identify each forecasting task by some parameters $c \subset \psi$, such as constants of motion, external forces, and boundary conditions. We learn multiple tasks simultaneously and infer the task from data. Here we use c for a subset of system parameters ψ, because we usually do not have the full knowledge of the system dynamics. In the turbulent flow example, the state $\mathbf{x}_t$ is the velocity field at time t. Parameters c can represent Reynolds number, mean vorticity, mean magnitude, or a vector of all three.

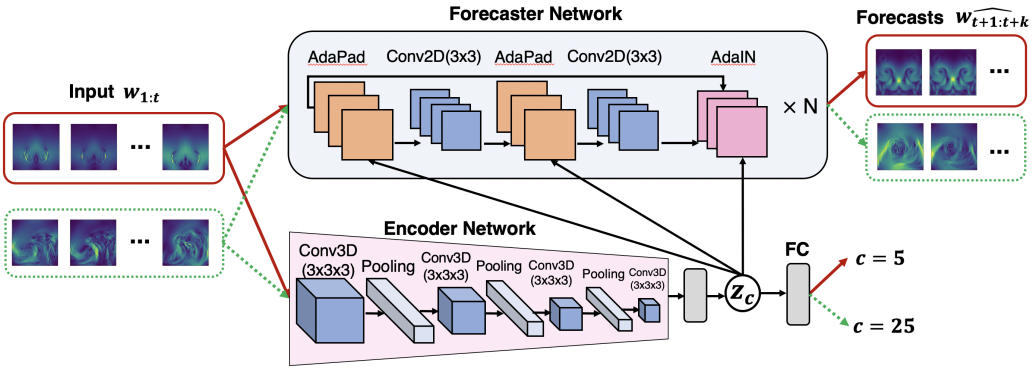

Figure 2: Overview of DyAd applied to two inputs of fluid turbulence, one with small external forces and one with larger external forces. The encoder infers the time-shift invariant characteristic variable z which is used to adapt the forecaster network.

Let μ be the data distribution over $\mathcal{X} \times \mathcal{Y}$ representing the function $f\colon \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} = \mathbb{R}^{d \times t}$ and $\mathcal{Y} = \mathbb{R}^{d \times k}$. Our main assumption is that the domain $\mathcal{X}$ can be partitioned into separate tasks $\mathcal{X} = \cup_{c \in \mathcal{C}} \mathcal{X}_c$, where $\mathcal{X}_c$ is the domain for task c and $\mathcal{C}$ is the set of all tasks. The data in the same task share the same set of parameters. Let μ_c be the conditional distribution over $\mathcal{X}_c \times \mathcal{Y}$ for task c.

During training, the model is presented with data drawn from a subset of tasks $\{(x, y) : (x, y) \sim \mu_c, c \sim C\}$. Our goal is to learn the function $f\colon \mathcal{X} \to \mathcal{Y}$ over the whole domain $\mathcal{X}$ which can thus generalize across all tasks $c \in \mathcal{C}$. To do so, we need to learn the map $g\colon \mathcal{X} \to \mathcal{C}$ taking $x \in \mathcal{X}_c$ to c in order to infer the task with minimal supervision.

2.2 DyAd: Dynamic Adaptation Network

We propose a model-based meta-learning approach for dynamics forecasting. Given multiple forecasting tasks, we propose to learn the function f in two stages, that is, by first inferring the task c from the input x, and then adapting to a specialized forecaster $f_c\colon \mathcal{X}_c \to \mathcal{Y}$ for each task. An alternative is to use a single deep neural network to directly model f in one step over the whole domain. But this requires the training set to have good and uniform coverage of the different tasks. If the data distribution μ is highly heterogeneous or the training set is not sampled i.i.d. from the whole domain, then a single model may struggle with generalization.

We hypothesize that by partitioning the domain into different tasks, the model would learn to pick up task-specific features without requiring uniform coverage of the training data. Furthermore, by separating task inference and forecasting into two stages, we allow the forecaster to rapidly adapt to new tasks that never appeared in the training set.

As shown in Fig. 2, our model consists of two parts: an encoder g and a forecaster f. We introduce z_c as a time-invariant hidden feature for task c. We assume that c depends linearly on the hidden feature for simplicity and easy interpretation. We design the encoder to infer the hidden feature z_c given the input x. We then use z_c to adapt the forecaster f to the specific task, i.e., model $y = f_c(x)$ as $y = f(x, z_c)$. As the system dynamics are encoded in the input sequence x, we can feed the same input sequence x to a forecaster and generate predictions $\hat{y} = f_c(x)$.

2.3 Encoder Network

The encoder maps the input x to the hidden features z_c that are time-invariant. To enforce this inductive bias, we encode time-invariance both in the architecture and in the training objective.

Time-Invariant Encoder The encoder is implemented using 4 Conv 3D layers, each followed by `BatchNorm`, `LeakyReLU`, and max-pooling. Note that theoretically, max-pooling is not perfectly shift invariant since 2x2x2 max-pooling is equivariant to shifts of size 2 and only approximately invariant to shifts of size 1. But standard convolutional architectures often include max-pooling layers to boost performance. We convolve both across spatial and temporal dimensions.

After that, we use a global mean-pooling layer and a fully connected layer to estimate the hidden feature $\hat{z}_c$. The task parameter depends linearly on the hidden feature. We use a fully connected layer to compute the parameter estimate $\hat{c}$. Since convolutions are equivariant to shift (up to boundary frames) and mean pooling is invariant to shift, the encoder is shift-invariant. In practice, shifting the time sequence one frame forward will add one new frame at the beginning and drop one frame at the end. This creates some change in output value of the encoder. Thus, practically, the encoder is only approximately shift-invariant.

Encoder Training The encoder network g is trained first. To combat the loss of shift invariance from the change from the boundary frames, we train the encoder using a time-invariant loss. Given two training samples $(x^{(i)}, y^{(i)})$ and $(x^{(j)}, y^{(j)})$ and their task parameters c, we have loss

$$\mathcal{L}_{\texttt{enc}} = \sum_{c \sim \mathcal{C}} \|\hat{c} - c\|^2 + \alpha \sum_{i,j,c} \|\hat{z}_c^{(i)} - \hat{z}_c^{(j)}\|^2 + \beta \sum_{i,c} \|\|\hat{z}_c^{(i)}\|^2 - m\|^2 \tag{1}$$

where $\hat{z}^{(i)} = g(x^{(i)})$ and $\hat{z}^{(j)} = g(x^{(j)})$ and $\hat{c}^{(i)} = W\hat{z}_c^{(i)} + b$ is an affine transformation of z_c.

The first term $\|\hat{c} - c\|^2$ uses weak supervision of the task parameters whenever they are available. Such weak supervision helps guide the learning of hidden feature z_c for each task. While not all parameters of the dynamical system are known, we can compute approximate values in the datum $c^{(i)}$ based on our domain knowledge. For example, instead of the Reynolds number of the fluid flow, we can use the average vorticity as a surrogate for task parameters.

The second term $\|\hat{z}_c^{(i)} - \hat{z}_c^{(j)}\|^2$ is the time-shift invariance loss, which penalizes the changes in latent variables between samples from different time steps. Since the time-shift invariance of convolution is only approximate, this loss term drives the time-shift error even lower. The third term $|\|\hat{z}_c^{(i)}\| - m|^2$ (m is a positive value) prevents the encoder from generating small $\hat{z}_c^{(i)}$ due to time-shift invariance loss. It also helps the encoder to learn more interesting z, even in the absence of weak supervision.

Hidden Features The encoder learns time-invariant hidden features. These hidden features resemble the time-invariant dimensionless parameters [30] in physical modeling, such as Reynolds number in fluid mechanics. The hidden features may also be viewed as partial disentanglement of the system state. As suggested by [37, 47], our disentanglement method is guided by inductive bias and training objectives. Unlike complete disentanglement, as in e.g. [39], in which the latent representation is factored into time-invariant and time-varying components, we focus only on time-shift-invariance. Nonetheless, the hidden features can control the forecaster which is useful for generalization.

2.4 Forecaster Network

The forecaster incorporates the hidden feature z_c from the encoder and adapts to the specific forecasting task $f_c = f(\cdot, z_c)$. In what follows, we use z for z_c. We use two specialized layers, adaptive instance normalization (AdaIN) and adaptive padding (AdaPad). AdaIN has been used in neural style transfer [26, 21] to control generative networks. Here, AdaIN may adapt for specific coefficients and external forces. We also introduce a new layer, $\mathrm{AdaPad}(x, z)$, which is designed for encoding the boundary conditions of dynamical systems. The backbone of the forecaster network can be any sequence prediction model. We use a design that is similar to `ResNet` for spatiotemporal sequences.

AdaIN We employ AdaIN to adapt the forecaster network. Denote the channels of input x by x_i and let $\mu(x_i)$ and $\sigma(x_i)$ be the mean and standard deviation of channel i. For each AdaIN layer, a particular style is computed $s = (\mu_i, \sigma_i)_i = Az + b$, where the linear map A and bias b are learned weights. Adaptive instance normalization is then defined as $y_i = \sigma_i \frac{x_i - \mu(x_i)}{\sigma(x_i)} + \mu_i$. In essence, the channels are renormalized to the style s.

For dynamics forecasting, the hidden feature z encodes data analogous to the coefficients of a differential equation and external forces on the system. In numerical simulation of a differential equation these coefficients enter as scalings of different terms in the equation and the external forces are added to the combined force equation [24]. Thus in our context AdaIN, which scales channels and adds a global vector, is well-suited to injecting this information.

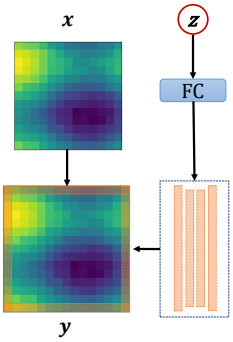

Figure 3: Illustration of the AdaPad operation.

AdaPad To complement AdaIN, we introduce AdaPad, which encodes the boundary conditions of each specific dynamical system. Generally when predicting dynamical systems, error is introduced along the boundaries since it is unknown how the dynamics interact with the boundary of the domain, and there may be unknown inflows or outflows. In our method, the inferred hidden feature z may contain the boundary information. AdaPad uses the hidden features to compute the boundary conditions via a linear layer. Then it applies the boundary conditions as padding immediately outside the spatial domain in each layer, as shown in Fig. 3.

Forecaster Training The forecaster is trained separately after the encoder. The kernels of the convolutions and the mappings of the AdaIN and AdaPad layers are all trained simultaneously as the forecaster network is trained. Denote the true state as y and the predicted state as $\hat{y}$, we compute the loss per time step $\|\hat{y} - y\|^2$ for each example. We accumulate the loss over different time steps and generate multi-step forecasts in an autoregressive fashion.

2.5 Theoretical Analysis

The high-level idea of our method is to learn a good representation of the dynamics that generalizes well across a heterogeneous domain, and then adapt this representation to make predictions on new tasks. Our model achieves this by learning on multiple tasks simultaneously and then adapting to new tasks with domain transfer. We prove that learning the tasks simultaneously as opposed to independently results in better generalization (Proposition B.3 in Appendix). We also provide a theoretical decomposition for the generalization error.Specifically, our hypothesis space has the form $\{x \mapsto f_\theta(x, g_\phi(x))\}$ where ϕ and θ are the weights of the encoder and forecaster respectively. Let $\epsilon_{\mathcal{X}}$ be the error over the entire domain $\mathcal{X}$, that is, for all c. Let $\epsilon_{\texttt{enc}}(g_\phi) = \mathbb{E}_{x\sim\mathcal{X}}(\mathcal{L}_1(g(x), g_\phi(x))$ be the encoder error where $g\colon \mathcal{X} \to \mathcal{C}$ is the ground truth. Let $\mathcal{G} = \{g_\phi\colon \mathcal{X} \to \mathcal{C}\}$ be the encoder hypothesis space. Denote the empirical risk of g_ϕ by $\hat{\epsilon}_{\texttt{enc}}(g_\phi)$. W_1 denotes the Wasserstein distance between tasks. $R(\mathcal{G})$ and $R(\mathcal{F})$ represent the Rademacher complexities of encoder and forecaster. The following Proposition decomposes the generalization error in terms of forecaster error, encoder error, and distance between tasks.

Proposition 2.1. *Assume the forecaster $c \mapsto f_\theta(\cdot, c)$ is Lipschitz continuous with Lipschitz constant γ uniformly in θ and $l \leq 1/2$. Let $\lambda_c = \min_{f\in\mathcal{F}}(\epsilon_c(f) + 1/K\sum_{k=1}^{K}\epsilon_{c_k}(f))$. For large n and probability $\geq 1-\delta$,*

$$\epsilon_{\mathcal{X}}(f_\theta(\cdot,g_\phi(\cdot))) \leq \gamma\hat{\epsilon}_{\texttt{enc}}(g_\phi) + \frac{1}{K}\sum_{k=1}^{K}\hat{\epsilon}_{c_k}(f_\theta(\cdot, c_k)) + \mathbb{E}_{c\sim\mathcal{C}}\left[W_1\left(\hat{\mu}_c, \frac{1}{K}\sum_{k=1}^{K}\hat{\mu}_{c_k}\right) + \lambda_c\right]$$
$$+2\gamma R(\mathcal{G}) + 2R(\mathcal{F}) + (\gamma+1)\sqrt{\log(1/\delta)/(2nK)} + \sqrt{2\log(1/\delta)}\left(\sqrt{1/n} + \sqrt{1/(nK)}\right).$$

This result helps to quantify the trade-offs with respect to different terms in the error bound in our two-part architecture and the error of the model can be controlled by minimizing the empirical errors of the encoder and forecaster. See the Appendix Proposition B.6 for full proofs.

3 Related Work

Learning Dynamical Systems. Deep learning models are gaining popularity for learning dynamical systems [60, 8, 29, 5, 73, 64, 51]. An emerging topic is physics-informed deep learning [53, 6, 11, 69, 4, 3, 68, 12, 10] which integrates inductive biases from physical systems to improve learning. For example, [45, 6] incorporated Koopman theory into the architecture. [43] used deep neural networks to solve PDEs with physical laws enforced in the loss functions. [17] and [9] build models upon Hamiltonian and Lagrangian mechanics that respect conservation laws. [68] proposed a hybrid approach by marrying two well-established turbulent flow simulation techniques with deep learning to produce better prediction of turbulence. [10] propose a physics-informed GAN architecture that uses physics knowledge to inform the learning of both the generator and discriminator models. However, these approaches deal with a *specific* system dynamics instead of meta-learning.

Multi-task learning and Meta-learning. Multi-task learning [65] focuses on learning shared representations from multiple related tasks. Architecture-based MTL methods can be categorized into

encoder-focused [35] and decoder-focused [74]. There are also optimization-based MTL methods, such as task balancing methods [27]. But MTL assumes tasks are known a priori instead of inferring the task from data. On the other hand, the aim of meta-learning [63] is to leverage the shared representation to fast adapt to unseen tasks. Based on how the meta-level knowledge is extracted and used, meta-learning methods are classified into model-based [46, 1, 50, 59, 78], metric-based [67, 61] and gradient-based [13, 57, 16, 76]. Most meta-learning approaches are not designed for forecasting with a few exceptions. [72] proposed to train a domain classifier with weak supervision to help domain adaptation but focused on low-dimensioal time series classification problems. [50] designed a residual architecture for time series forecasting with a meta-learning parallel. [1] proposed a modular meta-learning approach for continuous control. But forecasting physical dynamics poses unique challenges to meta-learning as we seek ways to encode physical knowledge into our model.

Style Transfer. Our approach is inspired by neural style transfer techniques. In style-transfer, a generative network is controlled by an external style vector though adaptive instance normalization between convolutional layers. Our hidden representation bears affinity with the "style" vector in style transfer techniques. Rather than aesthetic style in images, our hidden representation encodes time-invariant features. Style transfer initially appear in non-photorealistic rendering [31]. Recently, neural style transfer [25] has been applied to image synthesis [15, 36], videos generation [56], and language translation [52]. For dynamical systems, [58] adapts texture synthesis to transfer the style of turbulence for animation. [28] studies unsupervised generative modeling of turbulent flows but for super-resolution reconstruction rather than forecasting.

Video Prediction. Our work is also related to video prediction. Conditioning on the historic observed frames, video prediction models are trained to predict future frames, e.g., [41, 14, 75, 66, 49, 14, 70, 71, 32, 40, 34]. There is also conditional video prediction [48] which achieves controlled synthesis. Many of these models are trained on natural videos from unknown physical processes. Our work is substantially different because we do not attempt to predict object or camera motions. However, our method can be potentially combined with video prediction models to improve generalization.

4 Experiments

Table 1: Prediction RMSE on the turbulent flow and sea surface temperature datasets. Prediction RMSE and ESE (energy spectrum errors) on the future and domain test sets of ocean currents dataset.

Model	Turbulent Flows		Sea Temperature		Ocean Currents	
	future	domain	future	domain	future	domain
`ResNet`	0.94±0.10	0.65±0.02	0.73±0.14	0.71±0.16	9.44±1.55 | 0.99±0.15	9.65±0.16 | 0.90±0.16
`ResNet-c`	0.88±0.03	0.64±0.01	0.70±0.08	0.71±0.06	9.71±0.01 | 0.81±0.03	9.15±0.01 | 0.73±0.03
`U-Net`	0.92±0.02	0.68±0.02	0.57±0.05	0.63±0.05	7.64±0.05 | 0.83±0.02	7.61±0.14 | 0.86±0.03
`Unet-c`	0.86±0.07	0.68±0.03	0.47±0.02	0.45±0.06	**7.26±0.01** | 0.94±0.02	7.51±0.03 | 0.87±0.04
`PredRNN`	0.75±0.02	0.75±0.01	0.67±0.12	0.99±0.07	8.49±0.01 | 1.27±0.02	8.99±0.03 | 1.69±0.01
`VarSepNet`	0.67±0.05	0.63±0.06	0.63±0.14	0.49±0.09	9.36±0.02 | 0.63±0.04	7.10±0.01 | 0.58±0.02
`Mod-attn`	0.63±0.12	0.92±0.03	0.89±0.22	0.98±0.17	8.08±0.07 | 0.76±0.11	8.31±0.19 | 0.88±0.14
`Mod-wt`	0.58±0.03	0.60±0.07	0.65±0.08	0.64±0.09	10.1±0.12 | 1.19±0.72	8.11±0.19 | 0.82±0.19
`MetaNet`	0.76±0.13	0.76±0.08	0.84±0.16	0.82±0.09	10.9±0.52 | 1.15±0.18	11.2±0.16 | 1.08±0.21
`MAML`	0.63±0.01	0.68±0.02	0.90±0.17	0.67±0.04	10.1±0.21 | 0.85±0.06	10.9±0.79 | 0.99±0.14
`DyAd+ResNet`	**0.42±0.01**	**0.51±0.02**	0.42±0.03	0.44±0.04	7.28±0.09 | **0.58±0.02**	**7.04±0.04** | **0.54±0.03**
`DyAd+Unet`	0.58±0.01	0.59±0.01	**0.35±0.03**	**0.42±0.05**	7.38±0.01 | 0.70±0.04	7.46±0.02 | 0.70±0.07
`DyAd+Unet-Big`	0.57±0.02	0.59±0.01	0.36±0.01	0.43±0.02	7.72±0.02 | 0.70±0.04	7.84±0.10 | 0.79±0.02

4.1 Datasets

We experiment on three datasets: synthetic turbulent flows, real-world sea surface temperature and ocean currents data. These are difficult to forecast using numerical methods due to unknown external forces and complex dynamics not fully captured by simplified mathematical models.

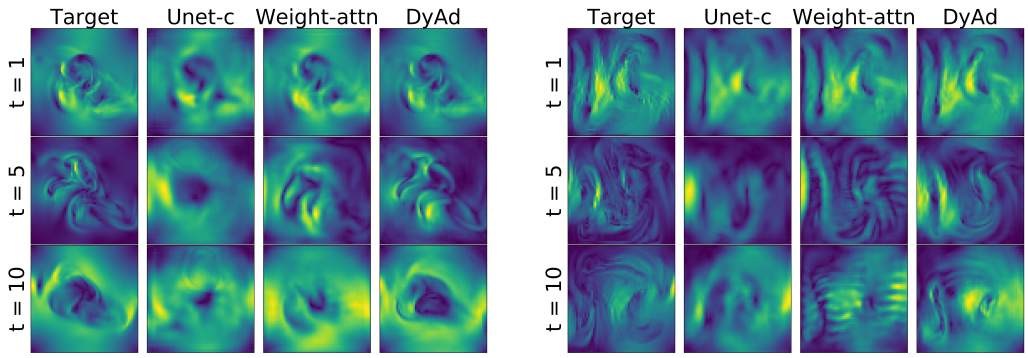

Figure 4: Target and predictions by `Unet-c`, `Modular-wt` and `DyAd` at time 1, 5, 10 for turbulent flows with buoyancy factors 9 (left) and 21 (right) respectively. `DyAd` can easily generate predictions for various flows while baselines have trouble understanding and disentangling buoyancy factors.

Turbulent Flow with Varying Buoyancy. We generate a synthetic dataset of turbulent flows with a numerical simulator, PhiFlow[1]. It contains 64×64 velocity fields of turbulent flows in which we vary the buoyant force acting on the fluid from 1 to 25. Each buoyant force corresponds to a forecasting task and there are 25 tasks in total. We use the mean vorticity of each task as partial supervision c as we can directly calculate it from the data. Vorticity can characterize formation and circular motion of turbulent flows.

Sea Surface Temperature. We evaluate on a real-world sea surface temperature data generated by the NEMO ocean engine [38][2]. We select an area from Pacific ocean range from 01/01/2018 to 12/31/2020. The corresponding latitude and longitude are (-150∼-120, -20∼-50). This area is then divided into 25 64×64 subregions, each is a task since the mean temperature varies a lot along longitude and latitude. For the encoder training, we use season as an additional supervision signal besides the mean temperature of each subregion. In other words, the encoder should be able to infer the mean temperature of the subregion as well as to classify four seasons given the temperature series.

Ocean Currents. We also experiment with the velocity fields of ocean currents from the same region and use the same task division as the sea surface temperature data set. Similar to the turbulent flow data set, we use the mean vorticity of each subregion as the weak-supervision signal.

4.2 Baselines

We include several SoTA baselines from meta-learning, as well as common methods for dynamics forecasting.

- `ResNet` [19]: A widely adopted video prediction model [69].
- `U-net` [55]: Originally developed for biomedical image segmentation, adapted for dynamics forecasting [11].
- `ResNet/Unet-c`: Above `ResNet` and `Unet` with an additional final layer that generates task parameter c and trained with weak-supervision and forecasting loss altogether.
- `PredRNN` [70]: A state-of-art RNNs-based spatiotemporal forecasting model.
- `VarSepNet` [12]: A convolutional dynamics forecasting model based on spatiotemporal disentanglement.
- `Mod-attn`[1]: A modular meta-learning method which combines the outputs of modules to generalize to new tasks using attention.
- `Mod-wt`: A modular meta-learning variant which uses attention weights to combine the parameters of the convolutional kernels in different modules for new tasks.
- `MetaNet` [46]: A model-based meta-learning method which requires a few samples from test tasks as a support set to adapt.

[1] `https://github.com/tum-pbs/PhiFlow`

[2] The data are available at `https://resources.marine.copernicus.eu/?option=com_csw&view=details&product_id=GLOBAL_ANALYSIS_FORECAST_PHY_001_024`

Model	future	domain
DyAd(ours)	**0.42**$\pm$**0.01**	**0.51**$\pm$**0.02**
No_enc	0.63$\pm$0.03	0.60$\pm$0.02
No_AdaPad	0.47$\pm$0.01	0.54$\pm$0.02
Wrong_enc	0.66$\pm$0.02	0.62$\pm$0.03
End2End	0.45$\pm$0.01	0.54$\pm$0.01

Table 2: Ablation study: prediction RMSE of DyAd and its variations with different components removed from DyAd.

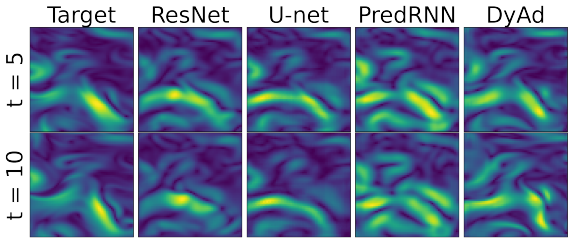

Figure 5: DyAd, ResNet, U-net, PredRNN velocity norm ($\sqrt{u^2+v^2}$) predictions on an ocean current sample in the future test set.

- MAML [13]: A optimization-based meta-learning approach. We replaced the original classifier with a ResNet for regression.

Note that both ResNet-c and Unet-c have access to task parameters c. Modular-attn has a convolutional encoder f that takes the same input x as each module M to generate attention weights, $\sum_{l=1}^{m} \frac{\exp[f(x)(l)]}{\sum_{k=1}^{m}\exp[f(x)(k)]} M_l(x)$. Modular-wt also has the same encoder but to generate weights for combining the convolution parameters of all modules. MetaNet requires samples from test tasks as a support set and MAML needs adaptation retraining on test tasks, while other models do not need any information from the test domains. Thus, we use additional samples of up to 20% of the test set from test tasks. MetaNet uses these as a support set. MAML is retrained on these samples for 10 epoch for adaptation. To demonstrate the generalizability of DyAd, we experimented with ResNet, U-net and a big U-net as our forecaster.

4.3 Experiments Setup

For all datasets, we use a sliding window approach to generate samples of sequences. For test-future, we train and test on the same tasks but different time steps. For test-domain, we train and test on different tasks with an 80-20 split. All models are trained to make next step prediction given the previous steps as input. We forecast in an autoregressive manner to generate multi-step ahead predictions. All results are averaged over 3 runs with random initialization.

Apart from the root mean square error (RMSE), we also report the energy spectrum error (ESE) for ocean current prediction which quantifies the physical consistency. ESE indicates whether the predictions preserve the correct statistical distribution and obey the energy conservation law, which is a critical metric for physical consistency. See details about energy spectrum and complete experiments details in Appendix A.1.

5 Results

5.1 Prediction Performance.

Table 1 shows the RMSE of multi-step predictions on Turbulent Flows (20 steps), Sea Surface Temperature (10 steps), and Ocean Currents (10 step) in two testing scenarios. We observe that DyAd makes the most accurate predictions in both scenarios across all datasets. Comparing ResNet/Unet-c with DyAd, we observe the clear advantage of task inference with separate training. VarSepNet achieves competitive performances on Ocean Currents (second best) through spatiotemporal disentanglement but cannot adapt to future data. Table 1 also reports ESEs on real-world Ocean Currents. DyAd not only has small RMSE but also obtains the smallest ESE, suggesting it captures the statistical distribution of ocean currents well.

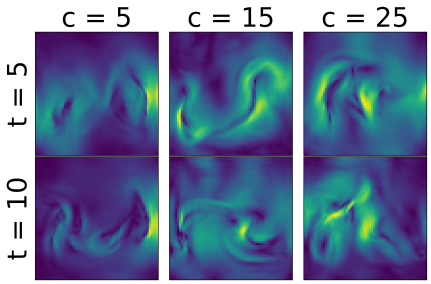

Figure 6: Outputs from DyAd while we vary encoder input but keep the forecaster input fixed. From left to right, the encoder is fed with flow with different buoyancy factor $c = 5, 15, 25$. the forecaster network input has fixed buoyancy $c = 15$.

Figure 4 shows the target and the predicted velocity norm fields ($\sqrt{u^2 + v^2}$) by `Unet-c`, `Modular-wt` and `DyAd` at time step 1, 5, 10 for Turbulent Flows with buoyancy factors 9 and 21 respectively. We can see that `DyAd` can generate realistic flows with the corresponding characteristics while the baselines have trouble understanding and disentangling the buoyancy factor. Figure 5 shows `DyAd`, `ResNet`, `U-net`, `PredRNN` predictions on an ocean current sample in the future test set, and we see the shape of predictions by `DyAd` is closest to the target. These results demonstrate that `DyAd` not only forecasts well but also accurately captures the physical characteristics of the system.

5.2 Controllable Forecast.

`DyAd` infers the hidden features from data, which allows direct control of the latent space in the forecaster. We vary the encoder input while keeping the forecaster input fixed. Figure 6 right shows the forecasts from `DyAd` when the encoder is fed with flows having different buoyancy factors $c = 5, 15, 25$. As expected, with higher buoyancy factors, the predictions from the forecaster become more turbulent. This demonstrates that the encoder can successfully disentangle the latent representation of difference tasks, and control the predictions of the forecaster.

5.3 Ablation Study.

We performed an ablation study of `DyAd` on the turbulence dataset to understand the contribution of each component.

Ablation of the Model Architecture We first performed ablation study of the model architecture, shown in Table 2. We first remove the encoder from `DyAd` while keeping the same forecaster network (`No_enc`). The resulting model degrades but still outperforms `ResNet`. This demonstrates the effectiveness of `AdaIN` and `AdaPad` for forecasting. We also tested `DyAd` with AdaIN only (`No_AdaPad`), and the performance without AdaPad was slightly worse.

Another notable feature of our model is the ability to infer tasks with weakly supervised signals c. It is important to have a c that is related to the task domain. As an ablative study, we fed the encoder in `DyAd` with a random c, leading to `Wrong_enc`. We can see that having the totally wrong supervision may slightly hurt the forecasting performance. We also trained the encoder and the forecaster in `DyAd` altogether (`End2End`) but observed worse performance. This validates our hypothesis about the significance of domain partitioning and separate training strategy.

Ablation of the Encoder Training Losses. We performed an ablation study of three training loss terms for the encoder to show the necessity of each term, shown in Table 5 in appendix A. We tried training the encoder only with the supervision of task parameters (`Sup_c`). We also tested training with the first two terms but without the magnitude loss (`Sup_c+time_inv`). We can see that missing any of three loss terms would make the encoder fail to learn the correct task-specific and time-invariant features.

Alternatives to AdaIN. We also tested 5 different alternatives to AdaIN for injecting the hidden feature z_c into the forecaster, and reported the results in Table 4 in Appendix A. We tested concatenating z_c to the input of the forecaster, using z_c as the kernels of the forecaster, concatenating z_c to the hidden states, adding z_c to the hidden states, using z_c as the biases in the convolutional layers in the forecaster. AdaIN worked better than any alternative we tested.

The implementation code of `DyAd` and baselines are included in the supplementary material. The readme file includes the instructions of downloading PhiFlow and generating the turbulent flow dataset. The ocean currents and sea temperature can be downloaded manually from the link provided in the Experiments section.

6 Conclusion

We propose a model-based meta-learning method, `DyAd` to forecast physical dynamics. `DyAd` uses an encoder to infer the parameters of the task and a prediction network to adapt and forecast giving the inferred task. Our model can also leverage any weak supervision signals that can help distinguish different tasks, allowing the incorporation of additional domain knowledge. On challenging turbulent flow prediction and real-world ocean temperature and currents forecasting tasks, we observe superior

performance of our model across heterogeneous dynamics. Future work would consider non-grid data such as flows on a graph or a sphere.

Acknowledgments and Disclosure of Funding

We are grateful to several anonymous reviewers for their comments which have helped us to improve this work. This work was supported in part by U.S. Department Of Energy, Office of Science grant DE-SC0022255, U. S. Army Research Office grant W911NF-20-1-0334, and NSF grants #2134274 and #2146343. R. Walters was supported by the Roux Institute and the Harold Alfond Foundation and NSF grants #2107256 and #2134178. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award ßASCR-ERCAP0022715.

References

[1] Ferran Alet, Tomas Lozano-Perez, and L. Kaelbling. Modular meta-learning. *ArXiv*, abs/1806.10166, 2018.

[2] Rie Kubota Ando, Tong Zhang, and Peter Bartlett. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(11), 2005.

[3] I. Ayed, Emmanuel de Bézenac, A. Pajot, J. Brajard, and P. Gallinari. Learning dynamical systems from partial observations. *ArXiv*, abs/1902.11136, 2019.

[4] Ibrahim Ayed, Emmanuel De Bézenac, Arthur Pajot, and Patrick Gallinari. Learning partially observed PDE dynamics with neural networks, 2019.

[5] Omri Azencot, N. Erichson, V. Lin, and Michael W. Mahoney. Forecasting sequential data using consistent koopman autoencoders. In *International Conference on Machine Learning*, 2020.

[6] Omri Azencot, N Benjamin Erichson, Vanessa Lin, and Michael Mahoney. Forecasting sequential data using consistent koopman autoencoders. In *International Conference on Machine Learning*, pages 475–485. PMLR, 2020.

[7] Jonathan Baxter. Theoretical models of learning to learn. In *Learning to learn*, pages 71–94. Springer, 1998.

[8] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6572–6583, 2018.

[9] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *ArXiv*, abs/2003.04630, 2020.

[10] Arka Daw, M Maruf, and Anuj Karpatne. Pid-gan: A gan framework based on a physics-informed discriminator for uncertainty quantification with physics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 237–247, 2021.

[11] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *International Conference on Learning Representations*, 2018.

[12] Jérémie Donà, Jean-Yves Franceschi, sylvain lamprier, and patrick gallinari. {PDE}-driven spatiotemporal disentanglement. In *International Conference on Learning Representations*, 2021.

[13] Chelsea Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference of Machine Learning*, 2017.

[14] Chelsea Finn, Ian Goodfellow, and Sergey Leine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[15] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[16] E. Grant, Chelsea Finn, S. Levine, Trevor Darrell, and T. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *ArXiv Preprint*, abs/1801.08930, 2018.

[17] S. Greydanus, Misko Dzamba, and J. Yosinski. Hamiltonian neural networks. *ArXiv*, abs/1906.01563, 2019.

[18] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 5307–5316, 2018.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Yun Hua, Xiangfeng Wang, Bo Jin, Wenhao Li, Junchi Yan, Xiaofeng He, and Hongyuan Zha. Hmrl: Hyper-meta learning for sparse reward reinforcement learning problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 637–645, 2021.

[21] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[22] Zijie Huang, Yizhou Sun, and Wei Wang. Coupled graph ode for learning interacting system dynamics. In *27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2021*, pages 705–715, 2021.

[23] Jessica Hwang, Paulo Orenstein, Judah Cohen, Karl Pfeiffer, and Lester Mackey. Improving subseasonal forecasting in the western us with machine learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2325–2335, 2019.

[24] J.C.Butcher. *Applied Numerical Mathematics*, volume 20. Elsevier B.V., 1996.

[25] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *IEEE transactions on visualization and computer graphics*, 26(11):3365–3385, 2019.

[26] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[27] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[28] Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. *arXiv:1908.10515*, 2019.

[29] J. Zico Kolter and Gaurav Manek. Learning stable deep dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 11128–11136, 2019.

[30] Josef Kunes. *Dimensionless physical quantities in science and engineering*. Elsevier, 2012.

[31] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the" art": A taxonomy of artistic stylization techniques for images and video. *IEEE transactions on visualization and computer graphics*, 19(5):866–885, 2012.

[32] Vincent Le Guen and Nicolas Thome. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *Computer Vision and Pattern Recognition (CVPR)*. 2020.

[33] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations*, 2021.

[34] Haoxing Lin, Rufan Bai, Weijia Jia, Xinyu Yang, and Yongjian You. Preserving dynamic attention for long-term spatial-temporal prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 36–46, 2020.

[35] Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-end multi-task learning with attention. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1871–1880, 2019.

[36] Xiao Liu, Spyridon Thermos, Alison Q. O'Neil, and Sotirios A. Tsaftaris. Semi-supervised meta-learning with disentanglement for domain-generalised medical image segmentation. In *MICCAI*, 2021.

[37] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR, 2019.

[38] Gurvan Madec et al. NEMO ocean engine, 2015. Technical Note. Institut Pierre-Simon Laplace (IPSL), France. `https://epic.awi.de/id/eprint/39698/1/NEMO_book_v6039.pdf`.

[39] Armand Comas Massague, Chi Zhang, Zlatan Feric, Octavia Camps, and Rose Yu. Learning disentangled representations of video with missing data. *arXiv preprint arXiv:2006.13391*, 2020.

[40] Armand Comas Massague, Chi Zhang, Zlatan Feric, Octavia I. Camps, and Rose Yu. Learning disentangled representations of video with missing data. *arXiv preprint arXiv: 2006.13391*.

[41] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[42] Yasuko Matsubara and Yasushi Sakurai. Dynamic modeling and forecasting of time-evolving data streams. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 458–468, 2019.

[43] George E Karniadakis Maziar Raissi, Paris Perdikaris. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[44] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[45] Jeremy Morton, Antony Jameson, Mykel J. Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[46] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *Proceedings of machine learning research*, 70:2554–2563, 2017.

[47] Weili Nie, Tero Karras, Animesh Garg, Shoubhik Debhath, A. Patney, Ankit B. Patel, and Anima Anandkumar. Semi-supervised stylegan for disentanglement learning. In *International Conference on Machine Learning*, 2020.

[48] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2863–2871, 2015.

[49] Sergiu Oprea, P. Martinez-Gonzalez, A. Garcia-Garcia, John Alejandro Castro-Vargas, S. Orts-Escolano, J. Garcia-Rodriguez, and Antonis A. Argyros. A review on deep learning techniques for video prediction. *ArXiv*, abs/2004.05214, 2020.

[50] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2019.

[51] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[52] Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. Style transfer through back-translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 866–876, 2018.

[53] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[54] Ievgen Redko, Amaury Habrard, and Marc Sebban. Theoretical analysis of domain adaptation with optimal transport. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 737–753. Springer, 2017.

[55] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[56] Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos. In *German conference on pattern recognition*, pages 26–36. Springer, 2016.

[57] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, 2019.

[58] Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. Example-based turbulence style transfer. *ACM Transactions on Graphics (TOG)*, 37(4):1–9, 2018.

[59] Sungyong Seo, Chuizheng Meng, Sirisha Rambhatla, and Y. Liu. Physics-aware spatiotemporal modules with auxiliary tasks for meta-learning. *arXiv preprint arXiv: 2006.08831*, 2020.

[60] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, D. Yeung, W. Wong, and Wang chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model. In *Advances in neural information processing systems*, 2017.

[61] J. Snell, Kevin Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, 2017.

[62] Qiuling Suo, Jingyuan Chou, Weida Zhong, and Aidong Zhang. Tadanet: Task-adaptive network for graph-enriched meta-learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1789–1799, 2020.

[63] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[64] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian fluid simulation with convolutional networks. In *ICML'17 Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3424–3433, 2017.

[65] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2021.

[66] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *International Conference on Learning Representations (ICLR)*, 2017.

[67] Oriol Vinyals, Charles Blundell, T. Lillicrap, K. Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.

[68] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1457–1466, 2020.

[69] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. *arXiv preprint arXiv:2002.03061*, 2020.

[70] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S Yu. PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs. In *Advances in Neural Information Processing Systems*, pages 879–888, 2017.

[71] Yunbo Wang, Haixu Wu, Jianjin Zhang, Zhifeng Gao, Jianmin Wang, Philip S Yu, and Mingsheng Long. PredRNN: A recurrent neural network for spatiotemporal predictive learning, 2021.

[72] Garrett Wilson, Janardhan Rao Doppa, and Diane J Cook. Multi-source deep domain adaptation with weak supervision for time-series sensor data. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1768–1778, 2020.

[73] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):95, 2018.

[74] Dan Xu, Wanli Ouyang, Xiaogang Wang, and N. Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 675–684, 2018.

[75] Tianfan Xue, Jiajun Wu, Katherine Bouman, and Bill Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in neural information processing systems (NeurIPS)*, pages 91–99, 2016.

[76] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *International Conference on Machine Learning*, pages 7045–7054. PMLR, 2019.

[77] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7343–7353, 2018.

[78] Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization. In *International Conference on Learning Representations*, 2021.

A Additional Results

A.1 Experiments Details

We use a sliding window approach to generate samples of sequences. For test-future, we train and test on the same tasks but different time steps. For test-domain, we train and test on different tasks with an 80%-20% split. All models are trained to make next step prediction given the previous steps as input. We forecast in an autoregressive manner to generate multi-step ahead predictions. All results are averaged over 3 runs with random initialization.

We tuned learning rate (1e-2$\sim$1e-5), batch size (16$\sim$64), the number of unrolled prediction steps for backpropogation (2$\sim$6), and hidden size (64$\sim$512). We fixed the number of historic input frames as 20. The models that use ResNet, including `MAML`, `MetaNets` and `DyAd+ResNet`, have same number of residual blocks for fair comparison. When we trained the encoder on turbulence and sea surface temperature, we used $\alpha = 1$ and $\beta = 1$ in the Equ. 1. For ocean currents, we used $\alpha = 0.2$ and $\beta = 0.2$. We performed all experiments on 4 V100 GPUs. Table 3 in Appendix A displays the number of parameters of each fine-tuned model on the turbulence dataset.

We compare our model with a series of baselines on the multi-step forecasting with different dynamics. We consider two testing scenarios: (1) dynamics with different initial conditions (test-future) and (2) dynamics with different parameters such as external force (test-domain). The first scenario evaluates the models' ability to extrapolate into the future for the same task. The second scenario is to estimate the capability of the models to generalize across different tasks.

Apart from the root mean square error (RMSE), we also report the energy spectrum error (ESE) for ocean current prediction which quantifies the physical consistency. ESE indicates whether the predictions preserve the correct statistical distribution and obey the energy conservation law, which is a critical metric for physical consistency. The turbulence kinetic energy spectrum $E(k)$ is related to the mean turbulence kinetic energy as

$$\int_0^\infty E(k)dk = (\overline{(u')^2} + \overline{(v')^2})/2, \quad \overline{(u')^2} = \frac{1}{T}\sum_{t=0}^{T}(u(t) - \bar{u})^2.$$

where the k is the wavenumber and t is the time step. The spectrum can describe the transfer of energy from large scales of motion to the small scales and provides a representation of the dependence of energy on frequency. Thus, the ESE can indicate whether the predictions preserve the correct statistical distribution and obey the energy conservation law. A trivial example that can illustrate why we need ESE is that if a model simply outputs moving averages of input frames, the accumulated RMSE of predictions might not be high but the ESE would be really big because all the small or even medium eddies are smoothed out.

A.2 Addtional Experiments Results

Table 3: The number of parameters of the fine-tuned model of each architecture on the turbulent flow dataset

`ResNet`	`U-net`	`PredRNN`	`VarSep`	`NetMod-attn`	`Mod-wt`	`MetaNets`	`MAML`
20.32	9.69	27.83	9.85	13.19	13.19	9.63	20.32
`DyAd+ResNet`	`DyAd+Unet`	`DyAd+Unet-Big`					
15.60	12.20	23.64					

B Theoretical Analysis

The high-level idea of our method is to learn a good representation of the dynamics that generalizes well across a heterogeneous domain, and then adapt this representation to make predictions on new tasks. Our model achieves this by learning on multiple tasks simultaneously and then adapting to new tasks with domain transfer. We prove that learning the tasks simultaneously as opposed

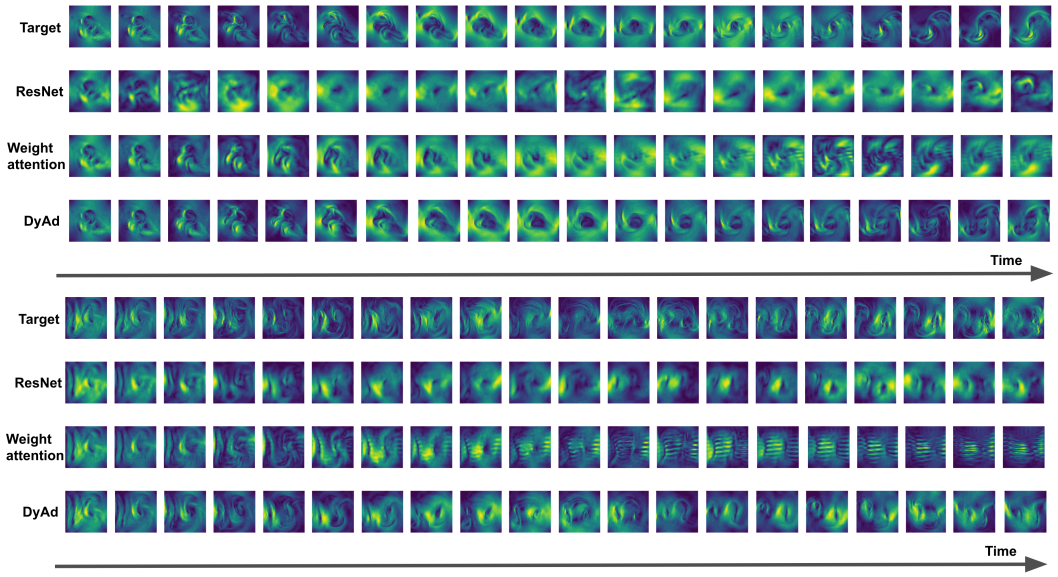

Figure 7: Target and full 20-step predictions by `Unet-c`, `Modular-wt` and `DyAd` turbulent flows with buoyancy factors 9 (top) and 21 (bottom) respectively.

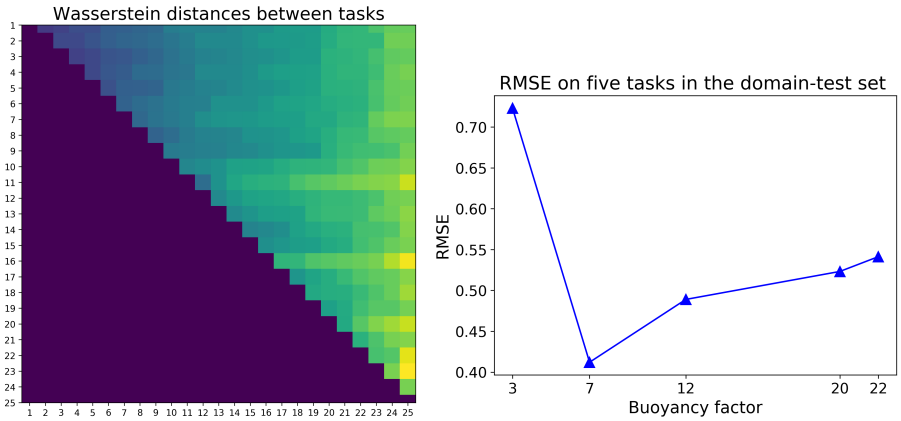

Figure 8: Left: Pairwise RMSEs between the averaged samples of different tasks in the turbulent flow dataset. RMSE between the averaged samples is a lower bound of Wasserstein distance between tasks. Right: DyAd+ResNet prediction RMSE breakdown on five tasks in the domain test set.

Table 4: We tested concatenating z_c to the input of the forecaster (ConcatI), using z_c as the kernels of the forecaster (KGen), concatenating z_c to the hidden states (ConcatH), adding z_c to the hidden states(Sum), using z_c as the biases in the convolutional layers (Bias). AdaIN worked better than any alternative we tested.

RMSE	AdaIN	ConcatI	KGen	ConcatH	Sum	Bias
`future`	0.42	1.09	0.78	0.75	0.84	0.84
`domain`	0.51	0.85	0.78	0.74	0.80	0.78

to independently results in tighter generalization bounds. As these are upper bounds, they do not necessarily imply better generalization, although empirically, this is what we observe. Although we are not able to provide a proof of our empirical finding that an encoder-forecaster architecture outperforms a monolithic architecture, we do provide an analysis of the error, decomposing it into error due to the forecaster, encoder, and domain adaptation. This decomposition helps to quantify the trade-offs involved in our two-part architecture.

Table 5: Ablation study of the encoder training loss terms

Model	future	domain
DyAd (ours)	**0.42±0.01**	**0.51±0.02**
Sup_c	0.53±0.01	0.56±0.01
Sup_c+time_inv	0.51±0.02	0.54±0.02

Suppose we have K tasks $\{c_k\}_{k=1}^K \sim \mathcal{C}$, each of which is sampled from a continuous parameter space $\mathcal{C}$. Here c_k are the parameters of the task, which can be inferred by the encoder. For each task c_k, we have a collection of data $\hat{\mu}_{c_k}$ of size n, sampled from μ_k, a shorthand for μ_{c_k}.

B.1 Multi-Task Learning Error

We want to bound the true loss ϵ using the empirical loss $\hat{\epsilon}$ and Rademacher complexity of the hypothesis class $\mathcal{F}$. We can use the classic results from [2]. Define empirical Rademacher complexity for all tasks as

$$\hat{R}_{\mathbf{X}}(\mathcal{F}) = \mathbb{E}_\sigma\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\sigma_k^{(i)} l(f(\mathbf{x}_k^{(i)}))\right)\right] \tag{2}$$

where $\{\sigma_k^{(i)}\}$ are independent binary variables $\sigma_k^{(i)} \in \{-1, 1\}$. The true Rademacher complexity is then defined $R(\mathcal{F}) = \mathbb{E}_{\mathbf{X}}(\hat{R}_{\mathbf{X}}(\mathcal{F}))$.

The following theorem restates the main result from [2]. We simplify the statement by using Rademacher complexity rather than the set cover number argument used in the original proof. The assumption of bounded loss is reasonable given a prescribed data domain and data preprocessing.

Theorem B.1. *([2]) Assume the loss is bounded $l \leq 1/2$. Given n samples each from K different forecasting tasks $\mu_1, \cdots, \mu_k$, then with probability at least $1-\delta$, the following inequality holds for each $f \in \mathcal{F}$ in the hypothesis class:*

$$\epsilon(f) \leq \hat{\epsilon}(f) + 2R(\mathcal{F}) + \sqrt{\log(1/\delta)/(2nK)}$$

Proof. Consider $\{\mathbf{x}_k^{(i)}\}$ as independent random variables. For a function ϕ that satisfies

$|\phi(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(i)}, \cdots \mathbf{x}^{(n)}) - \phi(\mathbf{x}^{(1)}, \cdots, \tilde{\mathbf{x}}^{(i)}, \cdots \mathbf{x}^{(n)})| \leq c_i$

by McDiarmid's inequality, we have

$$p\Big(\phi(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}) - \mathbb{E}[\phi] \geq t\Big) \leq \exp\left(-\frac{2t^2}{\sum_i c_i^2}\right).$$

Applying this inequality to the max difference $Q(\mathbf{X}) = \sup_{f\in\mathcal{F}}[\epsilon(f) - \hat{\epsilon}(f, \mathbf{X})]$, then with probability at least $1-\delta$, we have

$$Q(\mathbf{X}) - \mathbb{E}_{\mathbf{X}}[Q(\mathbf{X})] \leq C\sqrt{\frac{\log 1/\delta}{nK}}$$

where C is a constant depending on the bounds c_i. If the loss $l \leq 1/2$, then $|Q| \leq 1/2$ and so we can take $c_i = 1$ leading to $C = 1/\sqrt{2}$. A standard computation (see [44] Theorem 3.3) using the law of total expectation shows $\mathbb{E}_{\mathbf{X}}[Q(\mathbf{X})] \leq 2R(\mathcal{F})$, which finishes the proof. □

The following inequality compares the performance for multi-task learning to learning individual tasks. Let $R_k(\mathcal{F})$ be the Rademacher complexity for $\mathcal{F}$ over μ_k.

Lemma B.2. *The Rademacher complexity for multi-task learning is bounded $R(\mathcal{F}) \leq (1/K)\sum_{k=1}^K R_k(\mathcal{F})$.*

Proof. We compute the empirical Rademacher complexity,

$$\begin{aligned}
\hat{R}_{\mathbf{X}}(\mathcal{F}) &= \mathbb{E}_\sigma\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{nK}\sum_{k=1}^{K}\sum_{i=1}^{n}\sigma_k^{(i)} l\left(f\left(\mathbf{x}_k^{(i)}\right)\right)\right)\right] \\
&\leq \mathbb{E}_\sigma\left[\sum_{k=1}^{K}\sup_{f\in\mathcal{F}}\left(\frac{1}{nK}\sum_{i=1}^{n}\sigma_k^{(i)} l\left(f\left(\mathbf{x}_k^{(i)}\right)\right)\right)\right] \\
&= \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{\sigma_k}\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_k^{(i)} l\left(f\left(\mathbf{x}_k^{(i)}\right)\right)\right)\right] \\
&= \frac{1}{K}\sum_{k=1}^{K}\hat{R}_{\mathbf{X}_k}(\mathcal{F})
\end{aligned}$$

□

Proposition B.3. *Assume $n = n_k$ for all tasks k and the loss l is bounded $l \leq 1/2$, then the generalization bound given by considering each task individually is*

$$\epsilon(f) \leq \hat{\epsilon}(f) + 2\left(\frac{1}{K}\sum_{k=1}^{K}R_k(\mathcal{F})\right) + \sqrt{\frac{\log 1/\delta}{2n}}, \tag{3}$$

which is weaker than the bound of Theorem B.1.

Proof. Applying the classical analog of Theorem B.1 to a single task, we find with probability greater than $1-\delta$,

$$\epsilon_k(f) \leq \hat{\epsilon}_k(f) + 2R_k(\mathcal{F}) + C_k\sqrt{\frac{\log 1/\delta}{n}}.$$

Averaging over all tasks yields

$$\frac{1}{K}\sum_{k=1}^{K}\epsilon_k(f) \leq \frac{1}{K}\sum_{k=1}^{K}\hat{\epsilon}_k(f) + 2\frac{1}{K}\sum_{k=1}^{K}R_k(\mathcal{F}) + \frac{1}{K}\sum_{k=1}^{K}C_k\sqrt{\frac{\log 1/\delta}{n}}.$$

Since the loss l is bounded $l \leq 1/2$, we can take $C = C_k = 1/\sqrt{2}$, giving the generalization upper bound for the joint error of Equation 3. By Lemma B.2 and the fact $1/\sqrt{2nK} \leq 1/\sqrt{2n}$, the bound in Theorem B.1 is strictly tighter. □

The upper bound in Theorem B.1 is strictly tighter than that of Proposition B.3 as the first terms $\hat{\epsilon}(f)$ are equal, the second term is smaller by Lemma B.2 and the third is smaller since $1/\sqrt{2nK} \leq 1/\sqrt{2n}$. This helps explain why our multitask learning forecaster has better generalization than learning each task independently. The shared data tightens the generalization bound. Ultimately, though a tighter upper bound suggests lower error, it does not strictly imply it. We further verify this theory in our experiments by comparison to baselines which learn each task independently.

B.2 Domain Adaptation Error

Since we test on $c \sim \mathcal{C}$ outside the training set $\{c_k\}$, we incur error due to domain adaptation from the source domains $\mu_{c_1}, \ldots, \mu_{c_K}$ to target domain μ_c with μ being the true distribution. Denote the corresponding empirical distributions of n samples per task by $\hat{\mu}_c = \frac{1}{n_c}\sum_{i=1}^{n_c}\delta_{\mathbf{x}_c^{(i)}}$.

For different c and c', the domains μ_c and $\mu_{c'}$ may have largely disjoint support, leading to very high KL divergence. However, if c and c' are close, samples $\mathbf{x}_c \sim \mu_c$ and $\mathbf{x}_{c'} \sim \mu_{c'}$ may be close in the domain $\mathcal{X}$ with respect to the metric $\|\cdot\|_{\mathcal{X}}$. For example, if the external forces c and c' are close, given $\mathbf{x}_c \sim \mu_c$ there is likely $\mathbf{x}_{c'} \sim \mu_{c'}$ such that the distance between the velocity fields $\|\mathbf{x}_c - \mathbf{x}_{c'}\|$ is small. This implies the distributions μ_c and $\mu_{c'}$ may be be close in Wasserstein distance $W_1(\mu_c, \mu_{c'})$. The bound from [54] applies well to our setting as such:

Theorem B.4. *Let* $\lambda_c = \min_{f \in \mathcal{F}} \left(\epsilon_c(f) + 1/K \sum_{k=1}^{K} \epsilon_{c_k}(f) \right)$. *There is* $N = N(\dim(\mathcal{X}))$ *such that for* $n > N$, *for any hypothesis* f, *with probability at least* $1 - \delta$,

$$\epsilon_c(f) \leq \frac{1}{K} \sum_{k=1}^{K} \epsilon_{c_k}(f) + W_1 \left(\hat{\mu}_c, \frac{1}{K} \sum_{k=1}^{K} \hat{\mu}_{c_k} \right) + \sqrt{2 \log(1/\delta)} \left(\sqrt{1/n} + \sqrt{1/(nK)} \right) + \lambda_c.$$

Proof. We apply [54] Theorem 2 to target domain $\mu_T = \mu_c$ and joint source domain $\mu_S = 1/K \sum_{k=1}^{K} \mu_{c_k}$ with empirical samples $\hat{\mu}_T = \hat{\mu}_c$ and $\hat{\mu}_S = 1/K \sum_{k=1}^{K} \hat{\mu}_{c_k}$. □

B.3 Encoder versus Prediction Network Error

Our goal is to learn a joint hypothesis h over the entire domain $\mathcal{X}$ in two steps, first inferring the task c and then inferring x_{t+1} conditioned on c. Error from DyAd may result from either the encoder g_ϕ or the prediction network f_θ. Our hypothesis space has the form $\{x \mapsto f_\theta(x, g_\phi(x))\}$ where ϕ and θ are the weights of the encoder and prediction network respectively. Let $\epsilon_{\mathcal{X}}$ be the error over the entire domain $\mathcal{X}$, that is, for all c. Let $\epsilon_{\texttt{enc}}(g_\phi) = \mathbb{E}_{x \sim \mathcal{X}}(\mathcal{L}_1(g(x), g_\phi(x))$ be the encoder error where $g \colon \mathcal{X} \to \mathcal{C}$ is the ground truth. We state a result that decomposes the final error into that attributable to the encoder and that to the forecaster.

Proposition B.5. *Assume* $c \mapsto f_\theta(\cdot, c)$ *is Lipschitz continuous with Lipschitz constant* γ *uniformly in* θ. *Then we bound*

$$\epsilon_{\mathcal{X}}(f_\theta(\cdot, g_\phi(\cdot))) \leq \gamma \epsilon_{\texttt{enc}}(g_\phi) + \mathbb{E}_{c \sim \mathcal{C}} \left[\epsilon_c(f_\theta(x, c)) \right] \tag{4}$$

where the first term is the error due to the encoder incorrectly identifying the task and the second term is the error due the prediction network alone.

The hypothesis in the second term consists of the prediction network combined with the ground truth task label $x \mapsto f_\theta(x, g(x))$. In practice, the Lipschitz constant γ of the encoder may be indirectly minimized through weight decay.

Proof. By the triangle inequality and linearity of expectation,

$$\begin{aligned} \epsilon_{\mathcal{X}}(f_\theta(\cdot, g_\phi(\cdot))) &= \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| f_\theta(x, g_\phi(x)) - f(x) \|_{\mathcal{Y}} \right] \right] \\ &\leq \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| f_\theta(x, g_\phi(x)) - f_\theta(x, c) \|_{\mathcal{Y}} \right] \right] \\ &+ \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| f_\theta(x, c) \|_{\mathcal{Y}} - \| f(x) \|_{\mathcal{Y}} \right] \right]. \end{aligned}$$

By Lipschitz continuity,
$\leq \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\gamma \| g_\phi(x) - c \|_{\mathcal{C}} \right] \right] + \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| f_\theta(x, c) \|_{\mathcal{Y}} - \| f(x) \|_{\mathcal{Y}} \right] \right],$
which, since $g(x) = c$ and by linearity of expectation,
$= \gamma \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| g_\phi(x) - g(x) \|_{\mathcal{C}} \right] \right] + \mathbb{E}_{c \sim \mathcal{C}} \left[\mathbb{E}_{x \sim \mu_c} \left[\| f_\theta(x, c) \|_{\mathcal{Y}} - \| f(x) \|_{\mathcal{Y}} \right] \right]$
by definition of ϵ_{enc} and ϵ_c,$= \gamma \epsilon_{\texttt{enc}}(g_\phi) + \mathbb{E}_{c \sim \mathcal{C}} \left[\epsilon_c(f_\theta(x, c)) \right]$. □

By combining Theorem B.1, Proposition B.5, and Theorem B.4, we can bound the generalization error in terms of the empirical error of the prediction network on the source domains, the Wasserstein distance between the source and target domains, and the empirical error of the encoder. Let $\mathcal{G} = \{g_\phi \colon \mathcal{X} \to \mathcal{C}\}$ be the task encoder hypothesis space. Denote the empirical risk of the encoder g_ϕ with respect to $\mathbf{X}$ by $\hat{\epsilon}_{\texttt{enc}}(g_\phi)$.

Proposition B.6. *Assuming the hypotheses of Theorem B.1, Proposition B.5, and Theorem B.4,*

$$\begin{aligned} \epsilon_{\mathcal{X}}(f_\theta(\cdot, g_\phi(\cdot))) \leq \gamma \hat{\epsilon}_{\texttt{enc}}(g_\phi) &+ \frac{1}{K} \sum_{k=1}^{K} \hat{\epsilon}_{c_k}(f_\theta(\cdot, c_k)) + 2\gamma R(\mathcal{G}) + 2R(\mathcal{F}) \\ &+ (\gamma + 1) \sqrt{\frac{\log(1/\delta)}{2nK}} + \sqrt{2 \log(1/\delta)} \left(\sqrt{1/n} + \sqrt{1/(nK)} \right) \\ &+ \mathbb{E}_{c \sim \mathcal{C}} \left[W_1 \left(\hat{\mu}_c, \frac{1}{K} \sum_{k=1}^{K} \hat{\mu}_{c_k} \right) + \lambda_c \right]. \end{aligned}$$

Proof. We provide an upper bound for the right-hand side of (4). By Theorem B.1 or [44], Theorem 3.3, we can bound

$$\epsilon_{\texttt{enc}}(g_\phi) \leq \hat{\epsilon}_{\texttt{enc}}(g_\phi) + 2R(\mathcal{G}) + \sqrt{\frac{\log(1/\delta)}{2nK}}. \tag{5}$$

In order to apply Theorem B.1 to the risk ϵ_c and relate it to the empirical risk, we first relate the error on the target domain back to the source domain of our empirical samples. By Theorem B.4,

$$\begin{aligned}\epsilon_c(f_\theta(\cdot, c)) \leq & \frac{1}{K}\sum_{k=1}^{K} \epsilon_{c_k}(f_\theta(\cdot, c_k)) + W_1\left(\hat{\mu}_c, \frac{1}{K}\sum_{k=1}^{K}\hat{\mu}_{c_k}\right) \\ & + \sqrt{2\log(1/\delta)}\left(\sqrt{1/n} + \sqrt{1/(nK)}\right) + \lambda_c. \end{aligned} \tag{6}$$

Applying Theorem B.1, this is

$$\begin{aligned} \leq & \frac{1}{K}\sum_{k=1}^{K} \hat{\epsilon}_{c_k}(f_\theta(\cdot, c_k)) + 2R(\mathcal{F}) + \sqrt{\frac{\log 1/\delta}{2nK}} + W_1\left(\hat{\mu}_c, \frac{1}{K}\sum_{k=1}^{K}\hat{\mu}_{c_k}\right) \\ & + \sqrt{2\log(1/\delta)}\left(\sqrt{1/n} + \sqrt{1/(nK)}\right) + \lambda_c. \end{aligned} \tag{7}$$

Substituting (5) and (7) into (4) gives

$$\begin{aligned} \epsilon_{\mathcal{X}}(f_\theta(\cdot, g_\phi(\cdot))) \leq \gamma & \left(\hat{\epsilon}_{\texttt{enc}}(g_\phi) + 2R(\mathcal{G}) + \sqrt{\frac{\log(1/\delta)}{2nK}}\right) \\ & + \mathbb{E}_{c\sim\mathcal{C}}\left[\frac{1}{K}\sum_{k=1}^{K} \hat{\epsilon}_{c_k}(f_\theta(\cdot, c_k)) + 2R(\mathcal{F}) + \sqrt{\frac{\log 1/\delta}{2nK}}\right. \\ & \left. + W_1\left(\hat{\mu}_c, \frac{1}{K}\sum_{k=1}^{K}\hat{\mu}_{c_k}\right) + \sqrt{2\log(1/\delta)}\left(\sqrt{1/n} + \sqrt{1/(nK)}\right) + \lambda_c\right]. \end{aligned}$$

Using linearity of the expectation over c, removing it where there is no dependence on c, and rearranging terms. □

Although this result does not settle either the question of end-to-end versus pre-training or encoder-forecaster versus monolithic model, it quantifies the trade-offs these choices depend on. We empirically consider both questions in the experiments section.