
Generative Adaptation of Dynamics to Environmental Shifts via Weight-space Diffusion

Ruikun Li¹ Huandong Wang² Jingtao Ding² Yuan Yuan³ Qingmin Liao¹ Yong Li²

Abstract

Data-driven dynamics prediction often fails under environmental shifts, while traditional fine-tuning remains computationally prohibitive for hardware-constrained or data-scarce applications. We propose DynaDiff, a generative meta-learning framework that transitions the paradigm from gradient-based tuning or modulation to direct weight-space generation. Specifically, we first abstract expert weights as novel weight graphs, utilizing multi-head attention to explicitly capture topological coupling within weights. Subsequently, we design a functional loss to ensure that the generated models achieve consistency with expert models in physical behavior. Finally, we develop a dynamics-informed prompter that extracts cross-domain physical and spectral features from observation sequences to condition the diffusion model. Experiments demonstrate that DynaDiff boosts average prediction accuracy by 10.78% over competitive baselines. Furthermore, by pre-constructing a model zoo of expert predictors, we amortize the fine-tuning overhead into a one-time offline cost, significantly boosting deployment efficiency in new environments.

1. Introduction

Data-driven approaches have emerged as a powerful, equation-free paradigm for predicting physical dynamics (Wang et al., 2023; Ding et al., 2024), achieving considerable success across a diverse range of disciplines, including molecular dynamics (Mardt et al., 2018), fluid mechanics (Shu et al., 2023), and climate science (Bi et al., 2023).

¹Shenzhen International Graduate School, Tsinghua University ²Department of Electronic Engineering, Tsinghua University ³Courant Institute of Mathematical Sciences, New York University. Correspondence to: Huandong Wang <wanghuandong@tsinghua.edu.cn>.

In these systems, dynamical systems governed by the same underlying equations can exhibit vastly different evolutionary behaviors under varying environmental conditions e , which can be formally expressed as $\frac{dx}{dt} = f(x, t, e)$. For instance, fluid flows, described by the Navier-Stokes equations, can exhibit different vortex structures under various Reynolds number or external driving forces. Consequently, a predictive model f_{θ, e_a} , trained on observed trajectories of a specific environmental condition e_a struggles to generalize to unseen conditions e_b . Therefore, modeling the generalizable function f beyond the specific environment remains a critical problem for scientific machine learning (Subramanian et al., 2023; Goswami et al., 2022; Zhou et al., 2026).

Significant efforts have been undertaken to enable cross-environment prediction. Meta-learning approaches facilitate adaptation to unseen environments by simultaneously learning both environment-shared weights and environment-specific contexts (Kirchmeyer et al., 2022; Wang et al., 2022; Blanke & Lelarge, 2024). When applied to a new environment, the environment-specific contexts are tuned on new data to compose or modulate a tailored predictive model. Another strategy is to train environment-unified foundation models through well-designed architectures and large-scale parameterization (Herde et al., 2024; Hao et al., 2024; McCabe et al., 2024; Yang et al., 2023; Chen et al., 2024b). These models, pretrained on massive datasets, can be partially refined by finetuning on data specific to a target environment. However, from a model weight perspective, the essence of these methods only permit adaptation within a small, expert-specified subset of weights. This approach restricts the model’s ability to represent the true, complex manifold of expert weights across diverse environments. Moreover, the heavy reliance on gradient-based refinement or massive backbones renders existing paradigms less feasible for real-world scientific applications where data is scarce or deployment occurs on hardware-constrained platforms (Azzadenesheli et al., 2024). A more fundamental path is to directly generating the complete model weights θ via modeling the conditional distribution $p(\theta|e)$ (Figure 1).

Inspired by treating model weights as a data modality, this work focuses on generating environment-specific model weights (Figure 1c). By explicitly modeling the joint distri-

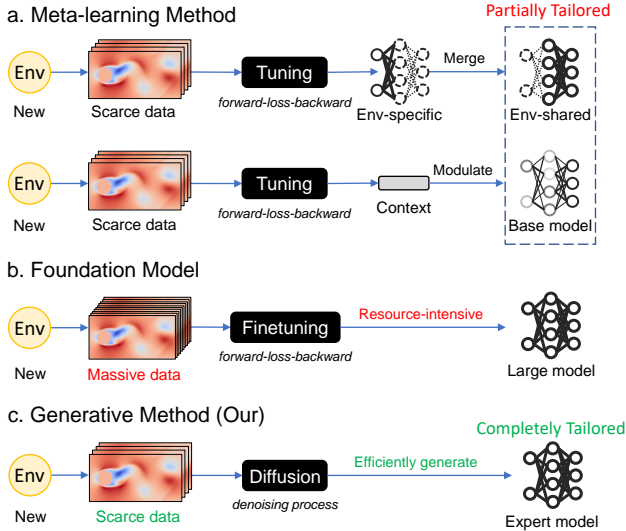


Figure 1. Paradigms for dynamics adaption.

bution of environments and weights, this generative adaptation is fundamentally suited for data-scarce scenarios where finetuning is impractical. However, the challenge of generating model weights for physical dynamics tailored to specific environments lies in three points. First, model weights exhibit functionally significant structures that are naturally dictated by the underlying network architecture. Thus, naive flattening weights into sequences would lead to the loss of crucial structural relationships (Kofinas et al., 2024). Second, the high dimensionality of weights results in an exceptionally vast parameter space. Minor variations in the weights of even a single layer can be amplified into significant difference in predictive performance (Plattner et al., 2025; Meynent et al., 2025). Therefore, traditional metrics like MSE are inadequate for assessing weight similarity. Finally, practical applications typically lack explicit physical knowledge of the environment, leaving only short trajectory snippets as available data. It is necessary to extract discriminative features of the underlying dynamics from such limited observations.

To address these challenges, we propose a novel generative meta learning framework, Dynamics-informed weight Diffusion (DynaDiff). DynaDiff employs weight graphs to represent predictive models, aggregating weights into node features to preserve their inherent connectivity and accommodate arbitrary model architectures (challenge 1). It utilizes a node-attention Variational Autoencoder (VAE) to learn latent representations for the diffusion model, and incorporates a functional loss that measures similarity based on model output consistency rather than raw parameter values (challenge 2). For unseen environments where data is scarce, we design a dynamics-informed prompter, which distills both physical features and temporal dynamics from limited observations, thereby providing a highly informative prompt for the diffusion model (challenge 3). Our contributions can be summarized as follows:

- We propose modeling the joint distribution of model weights on environments for cross-environment prediction, thereby rapidly generating expert weights for new environments without tuning.
- We construct weight graphs based on model architecture to preserve connectivity and design a functional loss for weight similarity perception. This significantly enhances the generative model’s ability to learn effective representations of model weights.
- Extensive experiments¹ on simulated and real-world systems demonstrate that DynaDiff improves cross-environment generalization, boosting average prediction accuracy by 10.78% over competitive baselines.

2. Preliminary

2.1. Problem Definition

Given environmental conditions $e \in \mathcal{E}$, the time-dependent system dynamics function is instantiated as $\frac{dx}{dt} = f(x, t, e) = f_e(x, t) \in \mathcal{F}$. The environment space \mathcal{E} and the function space \mathcal{F} are linked by the governing equations f , forming a joint set $\{e, f_e\}$. We employ a data-driven model $f_{\theta, e}$, parameterized by θ , to learn f_e , thereby formalizing the function space \mathcal{F} as the model’s weight space Θ . The environment space is divided into an observed environment set \mathcal{E}_{tr} and an unseen environment set \mathcal{E}_{te} , and consequently, the weight space is also partitioned into corresponding subspaces Θ_{tr} and Θ_{te} . Treating model weights as the modeling object, we learn the inherent joint distribution of environments and weights from the joint observation space $\{\mathcal{E}_{tr}, \Theta_{tr}\}$. For a new environment $e \in \mathcal{E}_{te}$, we generate a corresponding predictive function $f_{\theta, e}$ once learning is complete.

Notably, we posit that even when sharing the same governing equations, each environment determines a unique dynamical function. At test time, given a short observation sequence $X_L = \{x_0, \dots, x_{L-1}\}$ from a new, unseen environment $e \in \mathcal{E}_{te}$, our goal is to generate the complete model weights θ_{new} by modeling the conditional distribution $P(\theta|X_L)$. This approach, which generates a full expert model from scratch without requiring gradient-based finetuning, significantly differs from existing practices in dynamics prediction.

2.2. Conditional Diffusion

Diffusion models (Rombach et al., 2022; Dhariwal & Nichol, 2021) learn a probabilistic transformation from a prior Gaussian $p_{prior} \in \mathcal{N}(\mathbf{0}, \mathbf{I})$ distribution to a target distribution p_{target} . It perturbs data distributions by adding

¹<https://github.com/tsinghua-fib-lab/DynaDiff>

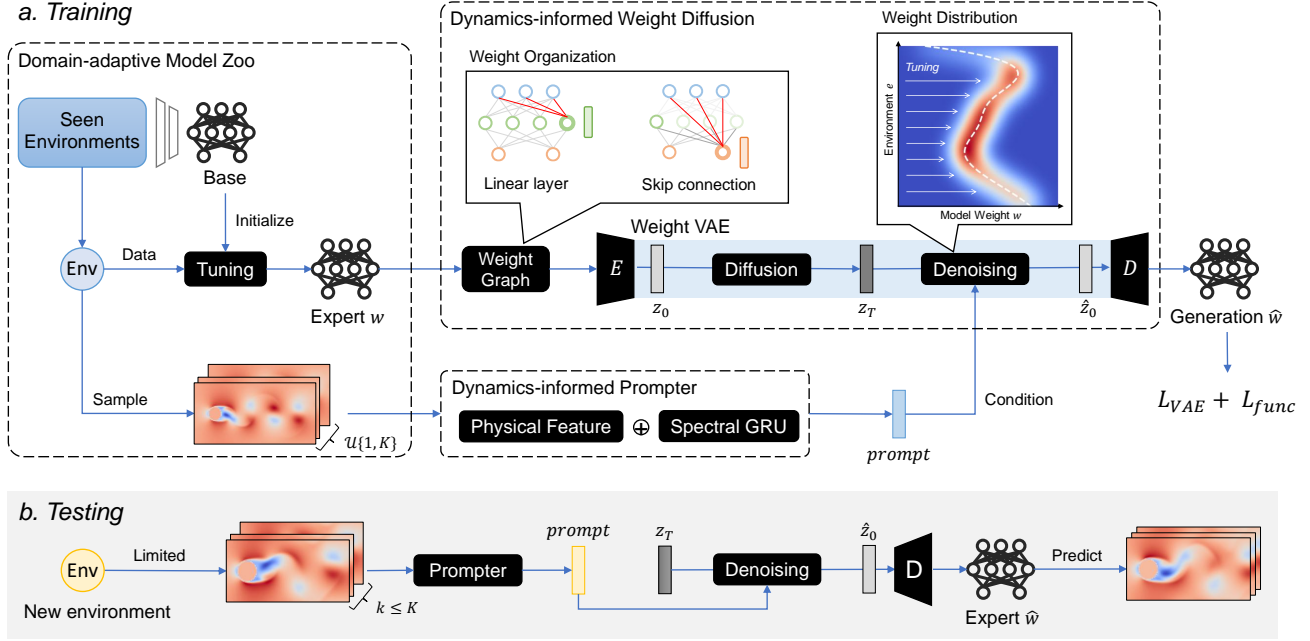


Figure 2. Framework of our Dynamics-informed weight Diffusion.

noise and learn to reverse this process through denoising, demonstrating strong fitting capabilities for data across modalities like images, language, and speech (Croitoru et al., 2023; Tumanyan et al., 2023; Cheng et al., 2025; Liu et al., 2025). We denote the original diffusion sample as x_0 . The forward noising process in standard diffusion models is computed as $x_n = \sqrt{\bar{\alpha}_n}x_0 + \sqrt{1 - \bar{\alpha}_n}\epsilon$, where ϵ and $\{\bar{\alpha}_n\}$ represent the Gaussian noise and noise schedule (Ho et al., 2020), respectively. The reverse process gradually denoises from Gaussian noise to sample data as

$$p_\theta(x_{n-1}|x_n) := \mathcal{N}(x_{n-1}; \mu_\theta(x_n, n), \sigma_n^2 \mathbf{I}), \quad (1)$$

where $\mu_\theta = \frac{1}{\sqrt{\alpha_n}}(x_n - \frac{1-\alpha_n}{\sqrt{1-\alpha_n}}\epsilon_\theta(x_n, n))$ and $\{\sigma_n\}$ are step dependent constants. The noise ϵ_θ is computed by a parameterized neural network, typically implemented as a UNet or Transformer architecture. The network’s parameters are optimized through an objective function (Ho et al., 2020)

$$L_n = \mathbb{E}_{n, \epsilon_n, x_0} \|\epsilon_n - \epsilon_\theta(\sqrt{\bar{\alpha}_n}x_0 + \sqrt{1 - \bar{\alpha}_n}\epsilon_n, n)\|^2 \quad (2)$$

to minimize the negative log-likelihood. To model conditional distributions $p(x|c)$, state-of-the-art methods inject conditional information during noise prediction using techniques like adaptive layer normalization (Peebles & Xie, 2023), as $\epsilon_\theta(x_n, n, c)$.

3. Methodology

In this section, we first introduce the method for modeling the joint distribution of model weights and environments, as illustrated in Figure 2a. Subsequently, we present a dynamics-informed prompter that operates with the limited observation sequence. Then, we detail the efficient

construction of a domain-adaptive model zoo. Finally, we provide a theoretical analysis of the generalization error bound for DynaDiff.

3.1. Dynamics-informed Weight Diffusion

DynaDiff first organizes the expert model weights into a weight graph. It then pretrains a weight VAE, yielding a high-quality latent space. Finally, an dynamics-informed diffusion model is trained within this latent space.

3.1.1. WEIGHT GRAPH

Model weights constitute a novel data modality, inherently structured by the network architecture. A straightforward approach is to break the network structure and flatten weights layer by layer into fixed-length token sequences for representation using sequence models like transformers (Schürholt et al., 2022; 2024). However, it neglects the inherent topological coupling and functional dependencies embedded within the network architecture. Here, we consider the inherent connection structure of the neural network. Specifically, we aggregate layer weights based on the forward data flow through the network topology to construct a weight graph that encapsulates the network’s connection structure.

We focus on designing the weight organization method for the basic computational units of modern AI architectures: linear layers and convolution layers (Kofinas et al., 2024). For a linear layer, learnable parameters include weights $\mathbf{w} \in \mathbb{R}^{D_{out} \times D_{in} \times 1}$ and bias $b \in \mathbb{R}^{D_{out} \times 1}$, where the D_{out} and D_{in} are the dimension of output and input respectively. A convolution layer similarly comprises weights

$\mathbf{w} \in \mathbb{R}^{C_{out} \times C_{in} \times h \times w}$ and bias $b \in \mathbb{R}^{C_{out} \times 1}$, where C_{out} and C_{in} are the channels of output and input, respectively, and $h \times w$ is the kernel size. We treat the output neurons of linear layers and output channels of convolution layers as nodes of the weight graph. Centering on the feature of output nodes, we flatten and concatenate the weights (and corresponding bias) associated with connections leading to each output node within a layer, forming the feature vector $\mathbf{w} \oplus b$ for that output node. Thus, a linear layer’s weights are organized as D_{out} nodes with $(D_{in} + 1)$ -dimensional features (Appendix Figure 12a), and a convolution layer’s weights are organized as C_{out} nodes with $(C_{in} \times h \times w + 1)$ -dimensional features (Appendix Figure 12b).

Considering the prevalence of skip connections in modern deep learning, we incorporate their weights. Following the data flow, we concatenate the weights of the skip connection path as additional features to the feature vector of the node where it merges with the main path, as depicted in Appendix Figure 12c. Consequently, the entire model weights are structured as a weight graph with heterogeneous node features, where the total number of nodes equals the sum of the output neurons/channels across all layers. We normalize weights based on input-output node pairs and biases based on nodes.

The proposed weight graph aggregates weights to nodes. This not only captures inherent connection relationships but also significantly reduces computational overhead compared to maintaining dense edge features. This organization method is applicable to most architectures (Section 4.2).

3.1.2. WEIGHT VAE

We now encode the heterogeneous graph of model weights to build a low-dimensional and informative latent space for diffusion model. We train a node attention-based VAE with a loss function given by

$$L_{VAE} = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{w})}[\log p_\theta(\mathbf{w}|\mathbf{z})] + \beta \text{KL}[q_\phi(\mathbf{z}|\mathbf{w})||p(\mathbf{z})], \quad (3)$$

where \mathbf{w} represents the heterogeneous node features of the weight graph, $\mathbf{z} \in \mathbb{R}^d$ is the latent representation, and the KL divergence term is used to constrain the posterior distribution $q_\phi(\mathbf{z}|\mathbf{w})$. The VAE architecture first employs a layer-wise linear mapping for each layer’s nodes to project them into a same dimension. Subsequently, we utilize a multi-head attention mechanism to model inter-node relationships, capturing interactions among neurons within and across original model layers. The resulting latent representation $\mathbf{z} = E(\mathbf{w})$ is then passed through another layer-wise linear mapping, projecting it back to the original dimensions for reconstruction $\hat{\mathbf{w}} = D(\mathbf{z})$.

We notice that prediction models exhibiting similar performance can possess distinct parameter values (Meynert

et al., 2025). This observation motivates our approach to the reconstruction error term in the VAE objective. The similarity between model weights should be gauged by their functional consistency, rather than merely their identical absolute values. We introduce a function loss,

$$L_{func} = \mathbb{E}_{x_i \in X} \|f_{\hat{\mathbf{w}}}(x_i) - f_{\mathbf{w}}(x_i)\|_2^2, \quad (4)$$

where $f_{\mathbf{w}}(x_i)$ and $f_{\hat{\mathbf{w}}}(x_i)$ are the output values of the original and reconstructed weights, respectively, when applied to an input sample x_i . Intuitively, the function loss allows the VAE to reconstruct weights that may not appear identical to the originals but perform similarly. It relaxes the encoder’s optimization constraints, promoting the learning of a latent space characterized by functional semantics. We theoretically analyze the effect of the function loss on generalization error in Appendix G.

3.1.3. WEIGHT LATENT DIFFUSION MODEL

In the latent space, we instantiate the noise network ϵ_θ using a transformer architecture. Conditioned on dynamics-informed *prompt* (introduced in follow), we inject this information into the network using adaptive layer norm (adaLN) (Peebles & Xie, 2023), forming $\epsilon_\theta(\mathbf{z}_n, n, \text{prompt})$. Compared to performing diffusive generation directly on the raw weights (Yuan et al., 2024), the latent space offers significant dimensionality reduction, which alleviates the computationally intensive nature of the diffusion process and simplifies the generation of representations.

3.2. Dynamics-informed Prompter

In most practical scenarios, only a short observation sequence X_L is available, instead of a known environmental parameter. The central task of the Prompter is to distill a rich, informative *prompt* vector from this limited sequence X_L . To leverage the strengths of both domain knowledge and data-driven feature extraction, we design a hybrid architecture composed of two parallel branches.

First, we extract physical features to capture the system’s macroscopic dynamics. For each state x_i , we compute its first and second-order moment statistics, energy, and entrophy (Taira et al., 2020). For the resulting time series of length L for each statistic, we then compute its temporal mean and trend to form the explicit prompt. Subsequently, we encode the microscopic evolution of the observation sequence. We compute the sequence of spectra for X_L via Fast Fourier Transform (FFT), stacking the real and imaginary parts. A Gated Recurrent Unit (GRU) is then used to capture the evolutionary patterns across frames, with its final hidden state serving as the implicit prompt. We concatenate the explicit and implicit prompts to form the final dynamics-informed *prompt*. Details are provided in the Appendix H.

We sample observation sequences with a variable length ranging from 1 to L for each training epoch. This enables the prompter to handle a flexible number of observation frames at test time. The prompter is trained jointly with the latent diffusion model in an end-to-end manner, where the gradients from the denoising objective backpropagate to guide the extraction of dynamical features.

3.3. Domain-adaptive Model Zoo

DynaDiff is trained on expert model weights, which are collected in a pre-constructed model zoo. While a naive approach would be to train each expert model from scratch (Schürholt et al., 2024), this process is computationally prohibitive and leads to a non-stationary weight distribution. To address this, we introduce an efficient construction process centered on domain-adaptive initialization (Chen et al., 2024b). First, we pretrain a global base model on data from all visible environments, analogous to the environment-shared weights in meta-learning. Subsequently, each environment-specific expert is obtained by rapidly fine-tuning this base model, as illustrated in Figure 2. To encourage sufficient exploration of the weight landscape, we also introduce a minor random noise to one layer of the base model before each fine-tuning process. Since each expert only needs to solve for a specific environment, its size is substantially smaller than a general-purpose foundation model. Therefore, our model zoo trades affordable offline storage (Appendix D) for a massive gain in training efficiency, eliminating the need for the inner-loop optimization common in prior meta-learning approaches (Finn et al., 2017; Dupont et al., 2022). Moreover, this one-time offline investment eliminates the need for any gradient-based computation when adapting to a new environment.

3.4. Generalization Analysis

We provide a theoretical analysis in Appendix G to demonstrate that our framework is principally designed to control its generalization error. First, by training a VAE with a functional loss, we construct a latent space that is functionally smooth, where proximity between latent vectors reflects the functional similarity of the decoded models. Next, a conditional diffusion model then accurately generate representations within this well-behaved space. Coupled with the prompter, this design ensures that each source of the total error is directly governed and minimized by a specific training objective.

4. Experiment

Dynamical Systems We validate the model’s effectiveness on four time-dependent PDE systems and one real-world dataset: 1) Cylinder Flow (Li et al., 2025b); 2) Lambda-Omega (Champion et al., 2019; Li et al., 2026b);

3) Kolmogorov Flow (Koupaï et al., 2024); 4) Navier-Stokes Equations (Kirchmeyer et al., 2022); and 5) ERA5 Dataset (Zhang et al., 2025). For the PDE systems, we use equation coefficients or external forcing as environmental variables and simulate multiple trajectories under different environments for training and testing. We train 100 FNO weight sets for each seen environment across all systems to serve as the model zoo of DynaDiff (size 100). Detailed descriptions and data generation procedures for each system are provided in Appendix C and D.

Setup Environmental conditions are unknown for all dynamical systems. We train DynaDiff solely on observed trajectories across diverse visible environments. Test environments are categorized as in-domain (seen during training, novel initial conditions) and out-domain (unseen environments) (Nzoyem et al., 2024). At test time, models autoregressively predict future states given a single initial frame. The prediction horizon is 100 steps for Cylinder Flow and Lambda-Omega, and 50 steps for Kolmogorov Flow and Navier-Stokes. We evaluate prediction quality using root mean square error (RMSE) and structural similarity index (SSIM). By default, the length of the observation sequence for new environments is $L = 10$.

Baselines We compare against two baseline categories: foundation models (One-for-All) and meta-learning approaches (Env-Adaptive). The foundation models are trained via empirical risk minimization (Ayed et al., 2019) on trajectories from all visible environments, including DPOT (Hao et al., 2024), Poseidon (Herde et al., 2024), and MPP (McCabe et al., 2024). The meta-learning methods learn environment-shared weights and update environment-specific contexts on observation sequences, including DyAd (Wang et al., 2022), LEADS (Yin et al., 2021), CoDA (Kirchmeyer et al., 2022), GEPS (Koupaï et al., 2024), and CAMEL (Blanke & Lelarge, 2024). Additionally, we compare against recent weight-space generation methods, CVAE (Shao et al., 2025) and D2NWG (Bedionita et al., 2025), which generate model weights via conditional variational autoencoder and latent diffusion, respectively. Following existing work (Blanke & Lelarge, 2024), we enable zero-shot prediction by conditioning the hypernetwork on environmental conditions e , which assumes ground-truth conditions are known. Additionally, we assume all environments are visible and train a dedicated Fourier neural operator (FNO) (Li et al., 2020) for each environment as a performance upper bound (One-per-Env). We also train an FNO only on all visible environments, but test without any adaptation as a performance lower bound (Not-Adaptive). We use FNO as the expert small model for DynaDiff and other meta-learning methods. Detailed architectural are provided in Appendix E and K.

Table 1. Average RMSE (\pm std from 5 runs) in out- and in-domain environments. Best in bold, underlined for suboptimal. The parameter sizes of predictive models are reported.

Methods	Testing Params	Cylinder Flow (96:400)		Lambda-Omega (12:39)		Kolmogorov Flow (12:39)		Navier-Stokes (24:121)		
		In-domain	Out-domain	In-domain	Out-domain	In-domain	Out-domain	In-domain	Out-domain	
Not-Adaptive		0.124 \pm 0.026	0.159 \pm 0.029	0.214 \pm 0.045	0.232 \pm 0.042	0.135 \pm 0.027	0.149 \pm 0.029	0.129 \pm 0.030	0.144 \pm 0.033	
One-per-Env	$\sim 1M$	0.040 \pm 0.040	0.038 \pm 0.040	0.038 \pm 0.032	0.035 \pm 0.008	0.069 \pm 0.021	0.071 \pm 0.019	0.046 \pm 0.007	0.047 \pm 0.009	
One-for-All	FNO	$\sim 500M$	0.082 \pm 0.025	0.083 \pm 0.023	0.352 \pm 0.041	0.363 \pm 0.040	0.080 \pm 0.020	0.096 \pm 0.016	0.066 \pm 0.009	0.074 \pm 0.015
	DPOT	$\sim 500M$	0.091 \pm 0.008	0.090 \pm 0.007	0.324 \pm 0.007	0.325 \pm 0.007	0.079 \pm 0.012	0.084 \pm 0.017	0.087 \pm 0.021	0.093 \pm 0.020
	Poseidon	$\sim 600M$	0.085 \pm 0.014	0.083 \pm 0.015	0.301 \pm 0.013	0.318 \pm 0.009	0.102 \pm 0.006	0.103 \pm 0.005	0.092 \pm 0.017	0.095 \pm 0.016
	MPP	$\sim 550M$	0.102 \pm 0.020	0.098 \pm 0.019	0.311 \pm 0.054	0.313 \pm 0.055	0.098 \pm 0.017	0.103 \pm 0.022	0.095 \pm 0.026	0.096 \pm 0.028
Env-Adaptive	DyAd		0.096 \pm 0.021	0.094 \pm 0.020	0.138 \pm 0.078	0.137 \pm 0.075	0.099 \pm 0.006	0.098 \pm 0.005	0.091 \pm 0.018	0.096 \pm 0.015
	LEADS		0.101 \pm 0.031	0.115 \pm 0.036	0.121 \pm 0.031	0.123 \pm 0.032	0.107 \pm 0.011	0.105 \pm 0.010	0.091 \pm 0.022	0.094 \pm 0.020
	CoDA		0.099 \pm 0.029	0.100 \pm 0.031	0.119 \pm 0.034	0.116 \pm 0.032	0.097 \pm 0.019	0.098 \pm 0.019	0.096 \pm 0.016	0.098 \pm 0.014
	GEPS		0.079 \pm 0.018	0.082 \pm 0.020	0.094 \pm 0.041	0.092 \pm 0.039	0.089 \pm 0.009	0.086 \pm 0.008	0.098 \pm 0.011	0.099 \pm 0.010
	CAMEL	$\sim 1M$	0.089 \pm 0.018	0.094 \pm 0.016	0.104 \pm 0.018	0.103 \pm 0.018	0.096 \pm 0.013	0.101 \pm 0.016	0.106 \pm 0.018	0.109 \pm 0.015
	CVAE	(+390M)	0.158 \pm 0.009	0.157 \pm 0.009	0.486 \pm 0.019	0.487 \pm 0.019	4.454 \pm 0.128	29.089 \pm 0.561	0.204 \pm 0.008	0.233 \pm 0.014
	D2NWX	(+400M)	0.082 \pm 0.019	0.086 \pm 0.017	0.102 \pm 0.020	0.105 \pm 0.021	0.086 \pm 0.010	0.090 \pm 0.013	0.088 \pm 0.016	0.089 \pm 0.015
DynaDiff	(+380M)	0.063\pm0.023	0.065\pm0.021	0.088\pm0.013	0.091\pm0.015	0.077\pm0.008	0.079\pm0.011	0.060\pm0.013	0.064\pm0.012	

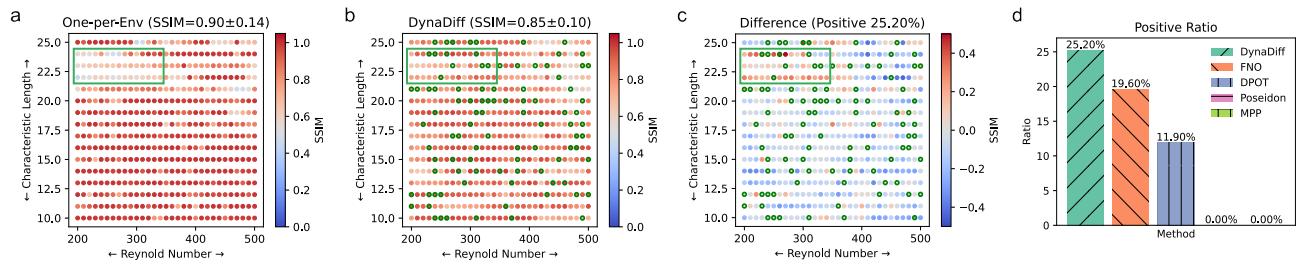


Figure 3. Predicting performance on Cylinder Flow. SSIM distribution of (a) One-per-Env and (b) DynaDiff; (c) Ratio where DynaDiff outperforms One-per-Env; (d) Differences between DynaDiff and One-per-Env. The green circle and box means seen environment during training and highlight region, respectively.

4.1. Main Results

PDE systems We report the generalization performance on 4 PDE systems in Table 1 and Appendix J.1, detailing the number of in/out-domain environments and the parameter size of models for each system during testing. The generative module of DynaDiff has approximately 380M parameters, while the predictive model at test-time has only 1M. Across nearly all systems, DynaDiff achieves the best average performance, demonstrating its ability to model the conditional dependence of the predictive model on environments. Its small, environment-specific expert models outperform foundation models with hundreds of times more parameters. Furthermore, unlike other meta-learning approaches, DynaDiff treats model weights holistically during adaption, without forcing the retention of environment-shared components. This potentially expands DynaDiff’s search space for improved generalization while increasing the parameter size. In Appendix J.9, we demonstrate that scaling up meta-learning baselines yields no comparable accuracy, confirming that DynaDiff’s superiority stems from its generative paradigm rather than mere parameter scaling.

We also find that some generated weights can outperform

One-per-Env in certain environments. This is likely due to the stochasticity of initialization and the training process, as One-per-Env models do not always converge to the optimal point. We illustrate this result with Cylinder Flow (2 environmental variables), as shown in Figure 3. The overall SSIM of One-per-Env is close to 1. However, it exhibits suboptimal performance in certain regions (green box in Figure 3). The FNO weights generated by DynaDiff perform better than One-per-Env in some environments, even unseen ones. This suggests that DynaDiff captures the manifold where the joint distribution of weights and environments lies, whereas the optimizer training process can fail to converge onto this manifold possibly due to getting stuck in local optima (Sclocchi & Wyart, 2024).

We visualize the joint distribution of environments and generated weights of CVAE, D2NWX, and DynaDiff on Cylinder Flow (Figure 4), where over 80% of environments are unseen during training. The x-axis and y-axis represent the first principal component of the environment prompts and generated weights, respectively. DynaDiff’s generated weight manifold closely matches the true expert distribution learned through optimizer training, whereas D2NWX and CVAE produces blurred distributions that fail to capture the

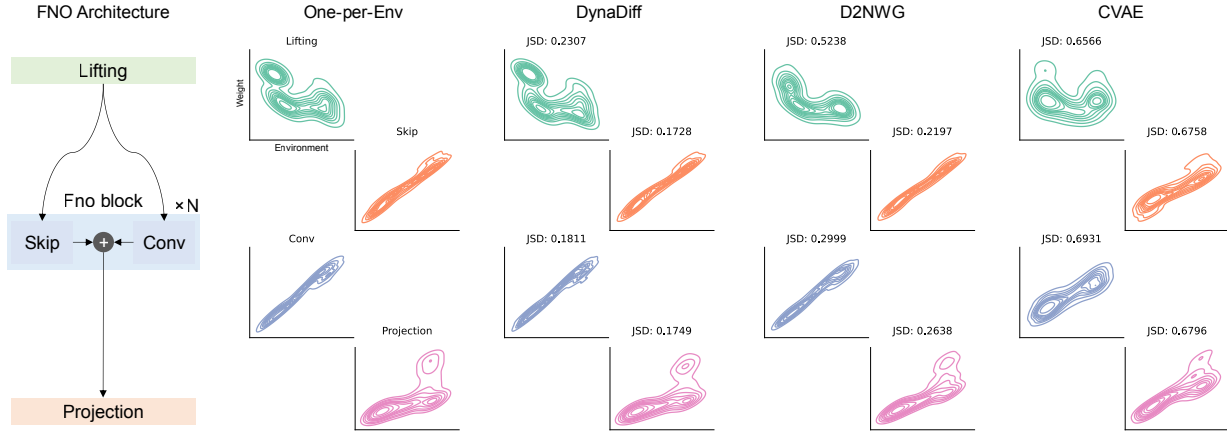


Figure 4. Joint distribution of environments and generated weights on Cylinder Flow, comparing DynaDiff, D2N WG, and CVAE against the true expert distribution.

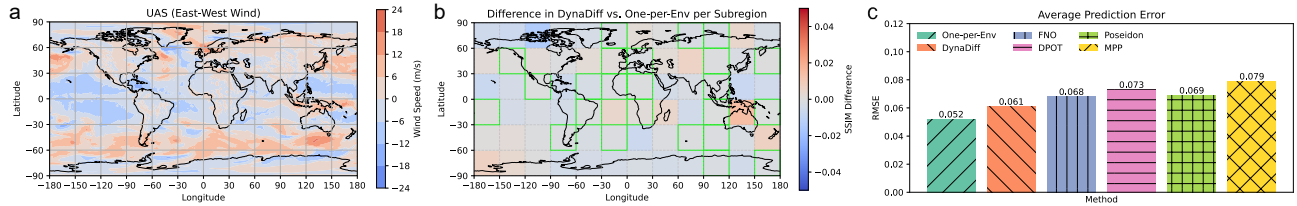


Figure 5. Predicting performance on ERA5 data. (a) One frame of ground true wind speed. (b) SSIM difference between DynaDiff and One-per-Env. The green box means seen environment during training. (c) Average prediction RMSE of DynaDiff and foundation models.

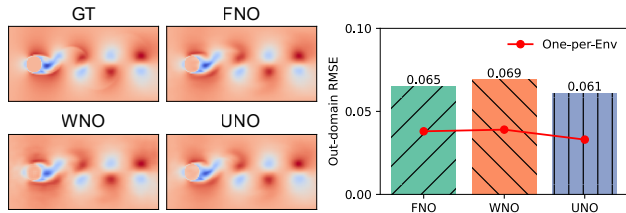


Figure 6. DynaDiff on the Cylinder Flow with different expert models of DynaDiff.

complex distribution of weights. Quantitatively, DynaDiff achieves the lowest Jensen-Shannon Divergence (JSD).

Moreover, we evaluate the physical interpretability of the learned prompts. As detailed in Appendix J.5, the prompter successfully extracts latent features that are highly correlated with ground-truth physical parameters (e.g., Reynolds numbers) solely through the end-to-end generative learning.

4.2. Extensibility

The weight graph structure proposed in Section 3.1.1 is capable of organizing neural networks of arbitrary architectures. Here, we extend to more neural operators as expert models within DynaDiff, including Wavelet Neural Operator (Tripura & Chakraborty, 2023) (WNO), and U-shape Neural Operator (Rahman et al., 2022) (UNO). Our experimental results on Cylinder Flow are presented in Figure 6. DynaDiff, when using different neural operators, consis-

tently achieves excellent generalization performance, with actual performance showing only minor variations depending on the specific operator architecture. This demonstrates that DynaDiff is a model-agnostic framework capable of benefiting from the sophisticated architectural designs of its expert models. Detailed architectures of these neural operators are provided in Appendix K.

Real-world dataset We utilize the ERA5 reanalysis dataset, including east-west and north-south wind speed data at a height of 100 meters. The spatial resolution is 0.25° , and the temporal resolution is 1 hour. We use January 2018 wind speeds as the training set and January 2019 as the test set. To define different environments, we divide the globe into 6×12 grid subregions at 30° intervals (Wang et al., 2022). We randomly select 24 subregions as seen environments, with the remaining 48 as unseen environments. The experimental results are shown in Figure 5. DynaDiff’s outperforms all baselines and is able to surpass One-per-Env in partial unseen subregions.

4.3. Explainability

As illustrated in Figure 7, the attention score matrices across DynaDiff’s encoder layers exhibit a distinct block-like structure, where the boundaries of each block align perfectly with the FNO’s internal hierarchy (e.g., *lifting*, *fno_blocks*,

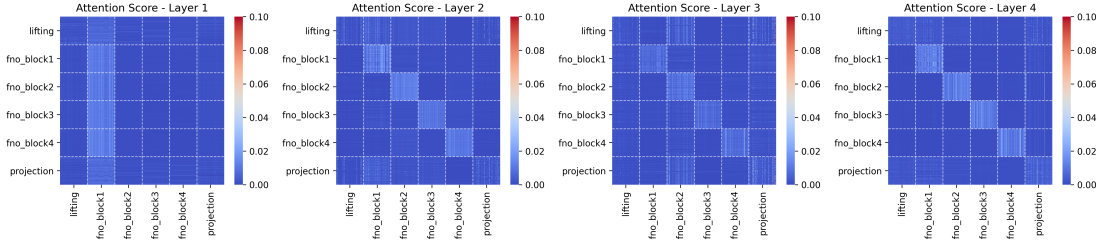


Figure 7. Attention score evolution across 4 distinct encoder layers for a single sample.

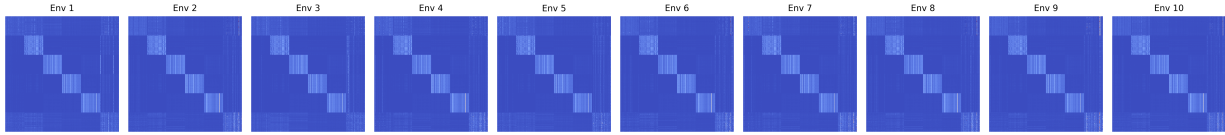


Figure 8. Comparison of the last-layer attention score across 10 different environment samples on Cylinder Flow.

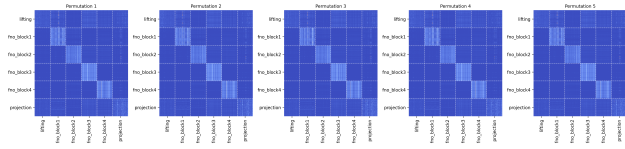


Figure 9. Attention score distribution under 5 permutation on Cylinder Flow.

and *projection*). This suggests that DynaDiff naturally recognizes and respects the functional divisions of the network architecture. Furthermore, the layer-wise evolution reveals a process of hierarchical decoupling within the parameter space. As the scores transition from broader global mixing in Layer 1 to highly localized clusters in Layer 4, DynaDiff increasingly focuses on specific functional modules.

In addition, Figure 8 shows that the attention score matrices share a similar structure across ten distinct environments. This indicates that while numerical weights fluctuate under environmental shifts, DynaDiff identifies the invariant topological dependencies and functional partitions. Such consistency validates the effectiveness of the weight graph, which preserves architectural priors rather than treating weights as flattened sequences. These observations demonstrate that DynaDiff successfully maps complex weights into a representation characterized by stable functional semantics.

4.4. Robustness

Environmental configuration We investigate the impact of the number of seen environments, model zoo size and the length of observation sequence L . We first examine the effect of model zoo size on Cylinder Flow and Lambda-Omega systems, as depicted in Appendix Figures 13a and b. The results indicate that DynaDiff exhibits relatively stable performance with a zoo size of 50. As the zoo size decreases further, performance begins to deteriorate, even within the distribution. Subsequently, we test the influence of the number of seen environments on the Kolmogorov Flow and Navier-Stokes systems. The number of environ-

ments ranged from approximately 5% to 20% of the total. The findings reveal that increasing the number of seen environments reduces prediction error, but the gains become marginal after reaching around 20%. This suggests that DynaDiff learns the underlying joint distribution of weights and environments from a small number of environments, rather than overfitting to trajectory samples within those environments. Finally, we test DynaDiff’s sensitivity to the observation length L . The results in Table 6 show that DynaDiff robustly captures the dynamic context to generate suitable predictors even with fewer frames, a flexibility enabled by our variable-length training strategy (Section 3.2).

Sampling stability As a stochastic generative process, diffusion sampling produces different weight sets for the same observation sequence. We sampled 100 times across 10 OOD environments for the Cylinder Flow and Navier-Stokes systems. As shown in Appendix Table 18, DynaDiff exhibits extremely low prediction variance. Crucially, even its worst-case sample outperforms the best baseline, confirming that DynaDiff rarely generates outlier models. This high stability is likely attributable to the domain-adaptive model zoo, which creates smooth basins in the loss landscape, and the diffusion process sampling on a continuous weight manifold that ensures a high performance floor. In deployment, for extreme stability, one can sample multiple models and select the one with the minimum reconstruction residual on the observation sequence.

Permutation Robustness A natural concern for the weight graph is whether its performance depends on consistent parameterization rather than true function-level structure. To test this, we randomly permuted 10% of neurons across random layers in the model zoo and retrained DynaDiff. As shown in Appendix Table 12, performance remained nearly unchanged, confirming that the global attention mechanism automatically adapts to equivalent weight permutations and captures true hierarchical structures rather than

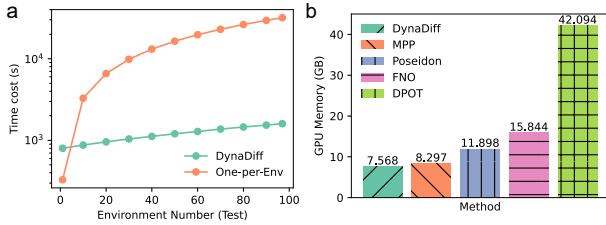


Figure 10. (a) Time cost and (b) Required GPU memory during testing on the Navier-Stokes system.

relying on fixed node ordering. The attention score (Figure 9) reveal that the global attention mechanism recognizes a consistent hierarchical structure, while intra-layer node scores automatically adapt to neuron permutations. This proves that the node features inherently embed sufficient topological discriminability, and the weight graph design is robust to equivalent transformations of the weight space.

Furthermore, in Appendix J.2 and J.3, we evaluate two challenging generalization scenario with a highly skewed distribution of training and testing environments, where DynaDiff consistently outperforms all baselines.

4.5. Ablation Study

Here, we verify the necessity of domain initialization when building the model zoo and the function loss used during VAE training. Experimental results on the Kolmogorov Flow and Navier-Stokes systems are presented in Appendix Table 11. When function loss is omitted, the VAE relies solely on MSE for reconstruction similarity, leading to suboptimal generated weights, particularly in unseen environments. Function loss relaxes VAE encoding constraints and helps prevent overfitting by prioritizing functional consistency over exact reconstruction. Removing domain initialization results in a significant deterioration in generated weight performance. This is attributed to the high complexity of a randomly initialized model zoo, which increases the modeling difficulty. We conclude that for weight generation aimed at generalization, sample quality is far more critical than diversity.

In addition, we compare the performance using our prompter against ground-truth environmental conditions, and analyze the impact of different diffusion architectures in Appendix J.5 and J.4.

4.6. Computational Cost

Time cost We compare the time overhead of DynaDiff and One-per-Env when adapting to new environments, as shown in Figure 10a. One-per-Env requires training weights for each new environment using observational data. DynaDiff’s overhead includes building the model zoo (accelerated by domain initialization) and generating weights for new environments. Though the upfront time cost of preparing

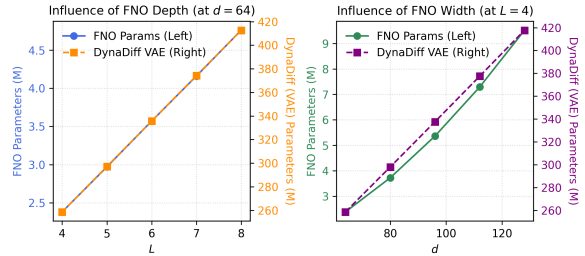


Figure 11. Generator parameter count scales linearly with predictor’s layer L and width d .

the model zoo, DynaDiff generates weights significantly faster than training a new predictor. A detailed end-to-end cost breakdown, including comparisons with test-time fine-tuning overhead, is provided in Appendix J.8. Notably, DynaDiff’s model zoo construction time is comparable to training standard foundation models, while its test-time adaptation is approximately $3\times$ faster than meta-learning baselines that require fine-tuning to achieve comparable accuracy. This highlights the trade-off of our generative meta-learning paradigm: exchanging a moderate offline cost for significant test-time efficiency.

Parameter scale A key concern for weight-space generation is how the generator’s parameter count scales with the target model. Taking an MLP module as an example (with L layers and width d), the total parameters are $O(Ld^2)$. Our weight graph design ensures the number of nodes and feature dimensions are $O(Ld)$ and $O(d)$, respectively. Since the attention mechanism is independent of node count, the generator’s parameter count (primarily projection layers) only changes linearly with L and d , rather than exponentially with the predictor’s parameter count. Figure 11 empirically confirms this linear scaling relationship.

GPU memory We compare the GPU memory usage of DynaDiff and other baselines during inference (Figure 10b). Thanks to the proposed weight graph structure, DynaDiff’s attention computation unfolds along the node dimension, significantly reducing computational overhead. Additional, we detail the cost of the model zoo in Appendix D.

5. Conclusion

We proposed DynaDiff, a framework for cross-environment generalization based on a new generative adaptation paradigm. DynaDiff synthesizes complete expert models from a few observations, guided by a dynamics-informed prompter and a generative model trained on a structured weight space. Our experiments demonstrate that this approach yields lightweight models with superior generalization, improving upon competitive baselines by an average of 10.78%. We conclude that generative weight modeling is a promising direction for scientific machine learning.

Impact Statements

This work introduces DynaDiff, a framework for the rapid adaptation of physical dynamics models to environmental shifts. The potential societal impacts are three-fold: (1) Computational Sustainability: By shifting the paradigm from compute-intensive gradient-based fine-tuning to instant generative weight synthesis, our approach significantly reduces the energy consumption and carbon footprint associated with retraining large-scale physical simulators. (2) Accessibility in Resource-Constrained Settings: The efficiency of DynaDiff enables high-fidelity physical modeling on edge devices and in data-scarce environments, potentially benefiting real-time monitoring in disaster response or remote scientific exploration where high-end compute is unavailable. (3) Safety in Critical Infrastructure: Improved generalization to environmental shifts (e.g., varying viscosity) enhances the reliability of AI-driven systems in weather forecasting, industrial control, and climate modeling, contributing to more resilient public infrastructure.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China under 2024YFC3307603, and the Science and Technology Innovation Program of Xiongan New Area under 2025XAGG0041.

References

- Alkin, B., Fürst, A., Schmid, S., Gruber, L., Holzleitner, M., and Brandstetter, J. Universal physics transformers: A framework for efficiently scaling neural operators. *Advances in Neural Information Processing Systems*, 37: 25152–25194, 2024.
- Ayed, I., de Bézenac, E., Pajot, A., Brajard, J., and Gallinari, P. Learning dynamical systems from partial observations. *arXiv preprint arXiv:1902.11136*, 2019.
- Azizzadenesheli, K., Kovachki, N., Li, Z., Liu-Schiaffini, M., Kossaifi, J., and Anandkumar, A. Neural operators for accelerating scientific simulations and design. *Nature Reviews Physics*, 6(5):320–328, 2024.
- Bedionita, S., Andreis, B., Lee, H., Jeong, W., Chong, S., Hutter, F., and Hwang, S. J. Diffusion-based neural network weights generation. In *13th International Conference on Learning Representations, ICLR 2025*, pp. 47860–47891. International Conference on Learning Representations, ICLR, 2025.
- Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., and Tian, Q. Accurate medium-range global weather forecasting with 3d neural networks. *Nature*, 619(7970):533–538, 2023.
- Blanke, M. and Lelarge, M. Interpretable meta-learning of physical systems. In *The Twelfth International Conference on Learning Representations*, 2024.
- Champion, K., Lusch, B., Kutz, J. N., and Brunton, S. L. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.
- Charakorn, R., Cetin, E., Tang, Y., and Lange, R. T. Text-to-lora: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025.
- Chen, T., Zhou, H., Li, Y., Wang, H., Gao, C., Shi, R., Zhang, S., and Li, J. Building flexible machine learning models for scientific computing at scale. *arXiv preprint arXiv:2402.16014*, 2024a.
- Chen, W., Song, J., Ren, P., Subramanian, S., Morozov, D., and Mahoney, M. W. Data-efficient operator learning via unsupervised pretraining and in-context learning. *Advances in Neural Information Processing Systems*, 37: 6213–6245, 2024b.
- Cheng, J., Li, R., Wang, H., and Li, Y. Sparse diffusion autoencoder for test-time adapting prediction of complex systems. *arXiv preprint arXiv:2505.17459*, 2025.
- Croitoru, F.-A., Hondru, V., Ionescu, R. T., and Shah, M. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45 (9):10850–10869, 2023.
- Dhariwal, P. and Nichol, A. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.
- Ding, J., Liu, C., Zheng, Y., Zhang, Y., Yu, Z., Li, R., Chen, H., Piao, J., Wang, H., Liu, J., et al. Artificial intelligence for complex network: Potential, methodology and application. *arXiv preprint arXiv:2402.16887*, 2024.
- Dupont, E., Kim, H., Eslami, S., Rezende, D., and Rosenbaum, D. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204*, 2022.
- Erkoç, Z., Ma, F., Shan, Q., Nießner, M., and Dai, A. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 14300–14310, 2023.
- Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

- Gong, Y., Zhan, Z., Li, Y., Idelbayev, Y., Zharkov, A., Aberman, K., Tulyakov, S., Wang, Y., and Ren, J. Efficient training with denoised neural weights. In *European Conference on Computer Vision*, pp. 18–34, 2024.
- Goswami, S., Kontolati, K., Shields, M. D., and Karniadakis, G. E. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 4(12):1155–1164, 2022.
- Hao, Z., Su, C., Liu, S., Berner, J., Ying, C., Su, H., Anandkumar, A., Song, J., and Zhu, J. Dpot: Auto-regressive denoising operator transformer for large-scale pde pre-training. In *International Conference on Machine Learning*, pp. 17616–17635. PMLR, 2024.
- Herde, M., Raonic, B., Rohner, T., Käppeli, R., Molinaro, R., de Bézenac, E., and Mishra, S. Poseidon: Efficient foundation models for pdes. *Advances in Neural Information Processing Systems*, 37:72525–72624, 2024.
- Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- Huang, H., Ding, J., Liao, M., Wang, X., Ban, J., Yuan, Y., Wan, H., and Li, Y. Loragen: Structure-aware weight space learning for lora generation. In *The Fourteenth International Conference on Learning Representations*.
- Kirchmeyer, M., Yin, Y., Donà, J., Baskiotis, N., Rakotomamonjy, A., and Gallinari, P. Generalizing to new physical systems via context-informed dynamics model. In *International Conference on Machine Learning*, pp. 11283–11301. PMLR, 2022.
- Kofinas, M., Knyazev, B., Zhang, Y., Chen, Y., Burghouts, G. J., Gavves, E., Snoek, C. G., and Zhang, D. W. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024.
- Kossaifi, J., Kovachki, N., Li, Z., Pitt, D., Liu-Schiaffini, M., George, R. J., Bonev, B., Azizzadenesheli, K., Berner, J., and Anandkumar, A. A library for learning neural operators, 2024.
- Koupaï, A. K., Benet, J. M., Yin, Y., Vittaut, J.-N., and Gallinari, P. Geps: Boosting generalization in parametric pde neural solvers through adaptive conditioning. *arXiv preprint arXiv:2410.23889*, 2024.
- Li, D., Wu, A., Li, Y., Wang, Y., and Han, Y. Continual adaptation: Environment-conditional parameter generation for object detection in dynamic scenarios. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4434–4443, 2025a.
- Li, R., Cheng, J., Wang, H., Liao, Q., and Li, Y. Predicting the dynamics of complex system via multiscale diffusion autoencoder. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 1493–1504, 2025b.
- Li, R., Liu, J., Wang, H., Liao, Q., and Li, Y. Weightflow: Learning stochastic dynamics via evolving weight of neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 641–649, 2026a.
- Li, R., Yao, J., Hua, Y., Tang, S., Qi, B., Liu, B., Ouyang, W., and Lu, Y. Pixel2phys: Distilling governing laws from visual dynamics. *arXiv preprint arXiv:2602.19516*, 2026b.
- Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- Liang, Z., Tang, D., Zhou, Y., Zhao, X., Shi, M., Zhao, W., Li, Z., Wang, P., Schürholt, K., Borth, D., et al. Drag-and-drop llms: Zero-shot prompt-to-weights. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Liang, Z., Tang, D., Zhou, Y., Zhao, X., Shi, M., Zhao, W., Li, Z., Wang, P., Schürholt, K., Borth, D., et al. Drag-and-drop llms: Zero-shot prompt-to-weights. *arXiv preprint arXiv:2506.16406*, 2025.
- Liu, J., Li, R., Wang, H., Yu, Z., Liu, C., Ding, J., and Li, Y. Beyond equilibrium: Non-equilibrium foundations should underpin generative processes in complex dynamical systems. *arXiv preprint arXiv:2505.18621*, 2025.
- Mardt, A., Pasquali, L., Wu, H., and Noé, F. Vampnets for deep learning of molecular kinetics. *Nature communications*, 9(1):5, 2018.
- McCabe, M., Régaldo-Saint Blancard, B., Parker, L., Ohana, R., Cranmer, M., Bietti, A., Eickenberg, M., Golkar, S., Krawezik, G., Lanusse, F., et al. Multiple physics pre-training for spatiotemporal surrogate models. *Advances in Neural Information Processing Systems*, 37:119301–119335, 2024.
- Meynent, L., Melev, I., Schürholt, K., Kauermann, G., and Borth, D. Structure is not enough: Leveraging behavior for neural network weight reconstruction. *arXiv preprint arXiv:2503.17138*, 2025.
- Nzoyem, R. D., Barton, D. A., and Deakin, T. Neural context flows for meta-learning of dynamical systems. *arXiv preprint arXiv:2405.02154*, 2024.

- Page, J., Norgaard, P., Brenner, M. P., and Kerswell, R. R. Recurrent flow patterns as a basis for two-dimensional turbulence: Predicting statistics from structures. *Proceedings of the National Academy of Sciences*, 121(23):e2320007121, 2024.
- Peebles, W. and Xie, S. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.
- Plattner, M., Berzins, A., and Brandstetter, J. Shape generation via weight space learning. *arXiv preprint arXiv:2503.21830*, 2025.
- Preechakul, K., Chatthee, N., Wizadwongsa, S., and Suwanajakorn, S. Diffusion autoencoders: Toward a meaningful and decodable representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10619–10629, 2022.
- Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. U-no: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.
- Rahman, M. A., George, R. J., Elleithy, M., Leibovici, D., Li, Z., Bonev, B., White, C., Berner, J., Yeh, R. A., Kossaiji, J., et al. Pretraining codomain attention neural operators for solving multiphysics pdes. *Advances in Neural Information Processing Systems*, 37:104035–104064, 2024.
- Redko, I., Habrard, A., and Sebban, M. Theoretical analysis of domain adaptation with optimal transport. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 737–753. Springer, 2017.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.
- Schürholt, K., Knyazev, B., Giró-i Nieto, X., and Borth, D. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.
- Schürholt, K., Mahoney, M. W., and Borth, D. Towards scalable and versatile weight space learning. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 43947–43966, 2024.
- Sclocchi, A. and Wyart, M. On the different regimes of stochastic gradient descent. *Proceedings of the National Academy of Sciences*, 121(9):e2316301121, 2024.
- Shao, Y., Yan, M., Liu, Y., Chen, S., Chen, W., Long, X., Yan, Z., Lei, L., Zhang, C., Sebe, N., et al. In-context meta lora generation. In *IJCAI*, pp. 6138–6146. International Joint Conferences on Artificial Intelligence, 2025.
- Shu, D., Li, Z., and Farimani, A. B. A physics-informed diffusion model for high-fidelity flow field reconstruction. *Journal of Computational Physics*, 478:111972, 2023.
- Soro, B., Andreis, B., Lee, H., Jeong, W., Chong, S., Hutter, F., and Hwang, S. J. Diffusion-based neural network weights generation. *arXiv preprint arXiv:2402.18153*, 2024.
- Subramanian, S., Harrington, P., Keutzer, K., Bhimji, W., Morozov, D., Mahoney, M. W., and Gholami, A. Towards foundation models for scientific machine learning: Characterizing scaling and transfer behavior. *Advances in Neural Information Processing Systems*, 36:71242–71262, 2023.
- Taira, K., Hemati, M. S., Brunton, S. L., Sun, Y., Duraisamy, K., Bagheri, S., Dawson, S. T., and Yeh, C.-A. Modal analysis of fluid flows: Applications and outlook. *AIAA journal*, 58(3):998–1022, 2020.
- Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.
- Tian, B., Chen, W., Li, Z., Lai, S., Wu, J., and Yue, Y. Text2weight: Bridging natural language and neural network weight spaces. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10152–10160, 2025.
- Tripura, T. and Chakraborty, S. Wavelet neural operator for solving parametric partial differential equations in computational mechanics problems. *Computer Methods in Applied Mechanics and Engineering*, 404:115783, 2023.
- Tumanyan, N., Geyer, M., Bagon, S., and Dekel, T. Plug-and-play diffusion features for text-driven image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1921–1930, 2023.
- Vlachas, P. R., Arampatzis, G., Uhler, C., and Koumoutsakos, P. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.
- Wang, H., Fu, T., Du, Y., Gao, W., Huang, K., Liu, Z., Chandak, P., Liu, S., Van Katwyk, P., Deac, A., et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

- Wang, K., Tang, D., Zeng, B., Yin, Y., Xu, Z., Zhou, Y., Zang, Z., Darrell, T., Liu, Z., and You, Y. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024.
- Wang, K., Tang, D., Zhao, W., Schürholt, K., Wang, Z., and You, Y. Recurrent diffusion for large-scale parameter generation. *arXiv preprint arXiv:2501.11587*, 2025.
- Wang, R., Walters, R., and Yu, R. Meta-learning dynamics forecasting using task inference. *Advances in Neural Information Processing Systems*, 35:21640–21653, 2022.
- Xiao, Y., Song, L., Yang, R., Cheng, C., Ge, Y., Li, X., and Shan, Y. Lora-gen: Specializing large language model via online lora generation. In *International Conference on Machine Learning*, pp. 68459–68471. PMLR, 2025.
- Xie, M., Li, S., Xie, B., Liu, C., Liang, J., Sun, Z., Feng, K., and Zhu, C. Weight diffusion for future: Learn to generalize in non-stationary environments. *Advances in Neural Information Processing Systems*, 37:6367–6392, 2024.
- Yang, L., Liu, S., Meng, T., and Osher, S. J. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.
- Yin, Y., Ayed, I., de Bézenac, E., Baskiotis, N., and Gallinari, P. Leads: Learning dynamical systems that generalize across environments. *Advances in Neural Information Processing Systems*, 34:7561–7573, 2021.
- Yuan, Y., Shao, C., Ding, J., Jin, D., and Li, Y. Spatio-temporal few-shot learning via diffusive neural network generation. *arXiv preprint arXiv:2402.11922*, 2024.
- Zhang, B., Luo, C., Yu, D., Li, X., Lin, H., Ye, Y., and Zhang, B. Metadiff: Meta-learning with conditional diffusion for few-shot learning. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 16687–16695, 2024.
- Zhang, Z., Lin, L., Gao, S., Wang, J., Zhao, H., and Yu, H. A machine learning model for hub-height short-term wind speed prediction. *Nature Communications*, 16(1): 3195, 2025.
- Zhou, S., Li, R., Wang, H., and Li, Y. Zero-shot forecasting of network dynamics through weight flow matching. In *Proceedings of the ACM Web Conference 2026*, pp. 1540–1550, 2026.

Supplementary Material

A. Limitations & Future Work

DynaDiff currently generates expert models of a fixed architecture, which may not be optimal for all possible environmental complexities. A promising future direction is to extend the generative paradigm to synthesize heterogeneous model architectures, dynamically tailored to each new environment.

B. Related Work

B.1. Dynamics Prediction across Environments

Developing dynamic prediction models with cross-environment generalization is a crucial problem in scientific machine learning and has garnered significant research interest. We review the main approaches and related work in this area. The first category trains large-parameter neural solvers as foundation models using extensive simulated data (Rahman et al., 2024; Alkin et al., 2024; Chen et al., 2024a). Subramanian et al. (2023) explore the generalization performance of classical FNO architectures across different parameter sizes. Subsequently, models such as MPP (McCabe et al., 2024), DPOT (Hao et al., 2024), and Poseidon (Herde et al., 2024) employed more advanced architectures to improve computational efficiency and approximation capabilities. The second approach is meta-learning (Finn et al., 2017). These methods capture cross-environment invariants through environment-shared weights and fine-tune environment-specific weights or contexts on limited data from new environments for adaptation, including DyAd (Wang et al., 2022), LEADS (Yin et al., 2021), CoDA (Kirchmeyer et al., 2022), GEPS (Koupaï et al., 2024), CAMEL (Blanke & Lelarge, 2024), and NCF (Nzoyem et al., 2024). Additionally, other methods exist, like in-context learning (Chen et al., 2024b). Yang et al. (Yang et al., 2023) frame differential equation forward and inverse problems as natural language statements, pre-train transformers, and provide solution examples for new environments as context to enhance model performance. Compared to these works, we innovatively treat the complete model weights as generated objects and explicitly model their joint distribution with the environment.

B.2. Diffusion for Network Weight Generation

Generating neural network weights is a relatively nascent research area (Wang et al., 2024; Huang et al.). An initial line of work involved training MLPs to overfit implicit neural fields, distilling them into model weights, and subsequently generating these MLP weights as an alternative to directly generating the fields (Erkoç et al., 2023; Li et al., 2026a). Another category proposes using generated weights to replace hand-crafted initialization, thereby accelerating and improving the neural network training process (Gong et al., 2024; Schürholt et al., 2022; 2024). These efforts primarily focus on image modalities. More recent studies leverage diffusion models to address generalization in various domains. Yuan et al. (Yuan et al., 2024) employ urban knowledge graph as prompts to guide diffusion for generating spatio-temporal prediction model weights for new cities. Zhang et al. (2024) replace the inner loop gradient updates of the meta learning with diffusion-generated weights. Xie et al. (2024) improve test-time generalization on time-varying systems by weight generation. Recent works (Soro et al., 2024; Charakorn et al., 2025; Liang et al., 2025) explore extracting features from unseen datasets and controlling diffusion to generate adapted model weights for them. However, most of these methods exhibit limited zero-shot performance. This may stem from them disrupting the neural network’s inherent topological connections by directly flattening the weights, which constrains the representational capacity of the generative model. We contrast DynaDiff with the latest weight generation methods (Table 2) to highlight our technical distinctions, including the structured weight graph, the dynamics-informed prompt, and the functional loss.

Table 2. Comparison of DynaDiff with recent weight-space generation methods.

	Task	Prompt Feature	Prompt Encoder	Generation Content	Data Structure	Generator	Supervision Signal	Model zoo
DnD (Liang et al.)	LLM adaptation	Task description + samples	Text encoder	Only LoRA	Regular weights	Convolutional decoder	Weight MSE	Yes
LoRA-Gen (Xiao et al., 2025)	LLM adaptation	Task description + samples	Text encoder	Only LoRA	Regular weights	Expert combination	Task loss + balancing	No
ICM-LoRA (Shao et al., 2025)	LLM adaptation	Task samples	Text encoder	Only LoRA	Regular weights	Conditional Variational Autoencoder	VAE Loss	Yes
CTTAOD (Li et al., 2025a)	Object detection	Task samples	CNN encoder	Only LoRA	Regular weights	Latent Diffusion	VAE + Diffusion Loss	No
RPG (Wang et al., 2025)	Image classification	Binary category vector	—	Full weights	Flattened weight vectors	Diffusion	Diffusion Loss	Yes
D2N WG (Bedionita et al., 2025)	Image classification	Task samples	CNN encoder	Full weights	Flattened weight vectors	Latent Diffusion	VAE + Diffusion Loss	Yes
T2W (Tian et al., 2025)	Image classification	Task samples	Text encoder	Full weights	Flattened weight vectors	Diffusion	Diffusion Loss	Yes
DynaDiff (Ours)	Dynamics prediction	Task samples	Dynamics-informed Prompter	Full weights	Structured weight graph	Latent Diffusion	VAE + Diff. + Function loss	Yes

C. Data Generation

Cylinder Flow system (Li et al., 2025b) is governed by:

$$\begin{cases} \dot{u}_t = -u \cdot \nabla u - \frac{1}{\alpha} \nabla p + \frac{\beta}{\alpha} \Delta u, \\ \dot{v}_t = -v \cdot \nabla v + \frac{1}{\alpha} \nabla p - \frac{\beta}{\alpha} \Delta v. \end{cases} \quad (5)$$

In this system, we use the Reynolds number Re and characteristic length r as two environmental variables. The Reynolds number and characteristic length influence the lattice viscosity, which in turn affects the collision frequency, leading to different dynamic behaviors.

Lambda–Omega system (Champion et al., 2019) is governed by

$$\begin{cases} \dot{u}_t = \mu_u \Delta u + (1 - u^2 - v^2)u + \beta(u^2 + v^2)v \\ \dot{v}_t = \mu_v \Delta v + (1 - u^2 - v^2)v - \beta(u^2 + v^2)u, \end{cases} \quad (6)$$

where Δ is the Laplacian operator. For this system, we use β as a 1-dimensional environmental variable. μ_u and μ_v are both set to 0.5.

Kolmogorov Flow system (Page et al., 2024) is governed by

$$\begin{aligned} \partial_t \omega + (\mathbf{u} \cdot \nabla) \omega &= \frac{1}{Re} \Delta \omega - n \cos(ny), \\ \nabla^2 \psi &= -\omega, \\ \mathbf{u} &= (u, v) = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right), \\ \omega &= (\nabla \times \mathbf{u}) \cdot \hat{\mathbf{z}} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}, \end{aligned}$$

For this system, we use Re as a 1-dimensional environmental variable. n is set to 3.

Navier-Stokes system (Takamoto et al., 2022) is governed by

$$\begin{aligned} \frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla) \omega &= \nu \Delta \omega + f, \\ \nabla^2 \psi &= -\omega, \\ \mathbf{u} &= (u, v) = \left(\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x} \right), \\ \omega &= (\nabla \times \mathbf{u}) \cdot \hat{\mathbf{z}} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}, \end{aligned}$$

where $f = A (\sin(2\pi(x + y + s)) + \cos(2\pi(x + y + s)))$ is the driving force. We use amplitude A and phase s as the two-dimensional environmental variables for this system, and the viscosity coefficient is set to 0.01.

The range of environmental values and simulation settings for each equation are listed in Table 3.

The Cylinder flow system is simulated using the lattice Boltzmann method (LBM) (Vlachas et al., 2022), with dynamics governed by the Navier-Stokes equations for turbulent flow around a cylindrical obstacle. The system is discretized using a lattice velocity grid, and the relaxation time is determined based on the kinematic viscosity and Reynolds number. Data collection begins once the turbulence has stabilized.

For Lambda–Omega system, the system’s reaction-diffusion equations are numerically integrated over time using an ODE solver.

For Kolmogorov Flow and Navier-Stokes systems, we perform numerical simulations based on the vorticity form equations. The process includes calculating the velocity field from vorticity by solving a Poisson equation using Fourier transforms,

Generative Adaptation of Dynamics to Environmental Shifts via Weight-space Diffusion

Table 3. Simulation settings of each PDE system.

	Cylinder flow	Lambda–Omega	Kolmogorov Flow	Navier-Stokes
Spatial Domain	—	$[-10, 10]^2$	$[-\pi, \pi]^2$	$[-32, 32]^2$
Grid Num	128×64	64×64	64×64	64×64
dt	200	0.04	0.2	0.025
T	45,000	40.0	40.0	50.0
Environments	$Re : [200, 500, 31], r : [10, 25, 16]$	$\beta : [1.0, 1.5, 51]$	$Re : [50, 250, 51]$	$A : [0.1, 0.3, 11], s : [0.0, 1.0, 11]$

employing numerical methods to handle spatial derivatives, and subsequently using an ODE solver for time integration to simulate the evolution of vorticity over time.

For each environment of each system, we predict 100 trajectories from different initial conditions for training and 20 trajectories for testing. For Cylinder Flow and Lambda–Omega systems, autoregressive prediction is performed for 100 steps during testing, while for Kolmogorov Flow and Navier-Stokes systems, prediction is performed for 50 steps during testing.

D. Model Zoo

In our main experiments, the settings for all meta-learning methods (except for DyAd, which uses a UNet by default) and the basic model of DynaDiff are shown in Table 4. Additionally, we report the storage overhead of the model zoo and the hyperparameter settings during generation. During training, we uniformly use the Adam optimizer with a learning rate of $1e-4$, and other parameters are set to their default values.

Table 4. Detailed settings of the model zoo for each systems.

	Cylinder flow	Lambda–Omega	Kolmogorov Flow	Navier-Stokes	ERA5
Channel Num	2	2	3	3	2
N_modes	[12, 6]	[8, 8]	[8, 8]	[8, 8]	[8, 8]
N_layers	4	4	4	8	4
Hidden	64	64	64	64	64
Domain Pretraining (epochs)	20	100	10	10	50
Finetuning (epochs)	50	50	50	50	20
Storage Space (GB)	18.0	3.5	3.5	7.1	3.4
Time Cost per Expert (s)	6.7	28.6	5.15	56.28	30.47

E. Model Architecture

The learnable parameters of DynaDiff consist of a weight VAE and a weight latent transformer diffusion model. The VAE includes layer-wise linear projection layers at the start and end stages, and inter-node attention layers in between. The diffusion model includes a noise network with a transformer architecture, where conditions are injected through adaptive layer normalization. The core model hyperparameters are configured as follows:

```

# --- VAE Hyperparameters ---
internal_dim = 1024      # Common internal dimension (D)
latent_dim = 512        # Latent dimension (h)
num_heads = 8           # Attention heads
num_attn_layers = 4     # Renamed from num_gnn_layers
# --- DiT Hyperparameters ---
hidden_size = 768       # Transformer hidden states
depth = 12              # Number of transformer blocks/layers
num_heads = 12          # Number of attention heads
\end{minted}

```

Taking the FNO configuration of the NS system as an example, the VAE with the above settings has $193.16M$ parameters, and the DiT has $131.53M$ parameters.

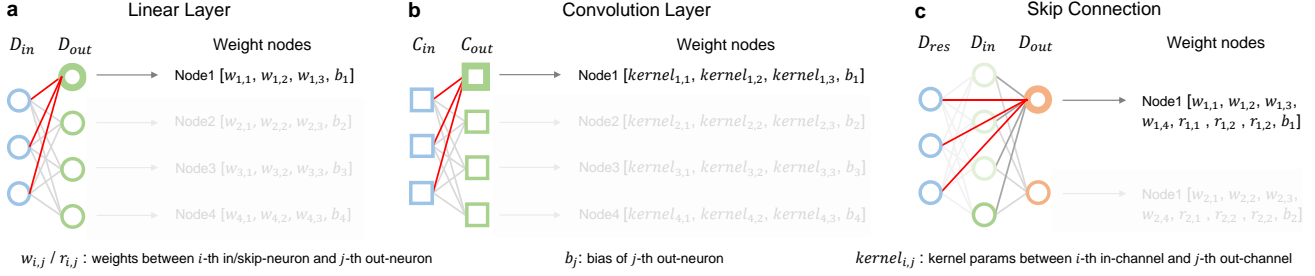


Figure 12. Layer-wise weight aggregation via forward data flow.

F. Baseline Implementation

The same training settings were used for all models, including training for 100 epochs using the Adam optimizer with a learning rate of $1e-4$. Regarding the selection of foundation model parameters, we uniformly adjusted the embedding dimension, number of layers, and number of heads based on the dimensions suggested in the original papers to ensure comparable parameter counts for all models. For environment-adaptive models, we primarily used the default hyperparameters.

G. Generalization Error Analysis

In this section, we provide a theoretical analysis of the generalization error for our proposed framework, DynaDiff. Our goal is to bound the expected functional error of a generated model \hat{w} for a new, unseen environment, given a short observation sequence X_L . Let w^* be the weights of an ideal expert model for this environment. The total generalization error can be expressed as $\mathcal{E}_{total} = \mathbb{E}[L_{func}(\hat{w}, w^*)]$, where the expectation is taken over the distribution of unseen environments and their corresponding observation sequences.

Our framework can be conceptualized as a composition of two modules: a prompter P , which maps an observation sequence to a conditional prompt, $prompt = P(X_L)$, and a generator G , which maps the prompt to the final model weights, $\hat{w} = G(prompt)$. The generator G itself is a composition of the latent diffusion model and the VAE decoder D . The total error arises from imperfections in both of these modules.

To formalize the analysis, we introduce the concept of an oracle prompt, $prompt^*$, which perfectly encapsulates all necessary information about the new environment. The total error can then be decomposed using the triangle inequality:

$$\sqrt{\mathcal{E}_{total}} \leq \sqrt{\mathbb{E}[L_{func}(G(prompt), G(prompt^*))]} + \sqrt{\mathbb{E}[L_{func}(G(prompt^*), w^*)]} \quad (7)$$

This decomposition separates the total error into two terms: the error induced by the imperfect prompter, and the inherent error of the generator even when given a perfect prompt.

Bounding the Prompter-induced Error. We first analyze the error originating from the prompter. We posit a key assumption that the generative process is smooth with respect to its conditioning. Specifically, we assume the generator G is γ -Lipschitz continuous in its functional output space with respect to the prompt.

Assumption 1 (Functional Lipschitz Continuity). There exists a constant $\gamma > 0$ such that for any two prompts, $prompt_1$ and $prompt_2$, the following holds:

$$\mathbb{E}[L_{func}(G(prompt_1), G(prompt_2))] \leq \gamma \cdot \|prompt_1 - prompt_2\|_2^2 \quad (8)$$

This assumption is encouraged by the smooth nature of the denoising process in diffusion models (Preechakul et al., 2022). Given this, the prompter-induced error is bounded by the prompter’s own generalization error, $\mathcal{E}_{prompt} = \mathbb{E}[\|P(X_L) - prompt^*\|_2^2]$. We assume a direct supervision to the prompter via an auxiliary regression loss, $\mathcal{L}_{aux} = \|e - \text{linear}(prompt)\|_2^2$ (see Appendix J.6), which forces the prompt to contain physically meaningful information correlated with the ground-truth environment e , thereby helping to minimize \mathcal{E}_{prompt} . While \mathcal{L}_{aux} is introduced here as a tractable surrogate to facilitate the

theoretical bounding of \mathcal{E}_{prompt} , the framework remains robust in label-free settings where task-driven gradients from the diffusion objective provide sufficient regularization to capture the underlying physics.

Bounding the Inherent Generator Error. The second term represents the generator’s error even under ideal conditioning. This error can be understood through the lens of domain adaptation theory (Redko et al., 2017; Wang et al., 2022), where the model zoo serves as the source domain and the unseen environments constitute the target domain. This error is primarily bounded by the generator’s empirical performance on the model zoo, which we denote as $\mathcal{E}_{empirical}(G) = \mathbb{E}_{w \sim \Theta_{tr}}[L_{func}(G(prompt_w), w)]$, where $prompt_w$ is the prompt corresponding to an expert model w .

To demonstrate that this empirical error is itself bounded, we analyze the two-stage generative process. *Assumption 2 (Latent Diffusion Effectiveness)*. An effectively trained conditional diffusion model can reverse the noising process in the latent space with high fidelity. This implies that the expected reconstruction error in the latent space is small. Let $z = E(w)$ be the latent representation of an expert model w . The expected squared error between z and its reconstruction \hat{z} after the full forward-and-reverse diffusion process is bounded by a small constant ϵ_z :

$$\mathbb{E}[\|z - \hat{z}\|_2^2] \leq \epsilon_z \quad (9)$$

This is a standard assumption, as minimizing the denoising objective L_n (Eq. 2) directly optimizes for this reconstruction capability. *Assumption 3 (Functional Smoothness of VAE Decoder)*. The VAE decoder D learns a smooth mapping from the latent space back to the functional space. This is a direct consequence of incorporating the functional loss L_{func} (Eq. 4) during its training. This implies the decoder is L_D -Lipschitz continuous in a functional sense:

$$\mathbb{E}[L_{func}(D(z_1), D(z_2))] \leq L_D \cdot \|z_1 - z_2\|_2^2 \quad (10)$$

The functional loss explicitly regularizes the mapping to ensure that small perturbations in the latent space do not lead to drastic changes in model behavior, thus encouraging a small L_D .

With these assumptions, we can bound the generator’s empirical error. Using the triangle inequality on the square root of the functional loss:

$$\sqrt{\mathcal{E}_{empirical}(G)} = \sqrt{\mathbb{E}[L_{func}(D(\hat{z}), w)]} \leq \sqrt{\mathbb{E}[L_{func}(D(\hat{z}), D(z))]} + \sqrt{\mathbb{E}[L_{func}(D(z), w)]} \quad (11)$$

The first term on the right-hand side is the error from latent space generation, bounded by $\sqrt{L_D \cdot \epsilon_z}$ due to Assumptions 2 and 3. The second term is precisely the VAE’s functional reconstruction error on the training data, which is minimized by the L_{func} term in the VAE objective (Eq. 4). Let us denote the value of this minimized loss as ϵ_{recon} .

This yields the final bound on the generator’s empirical error:

$$\mathcal{E}_{empirical}(G) \leq (\sqrt{L_D \cdot \epsilon_z} + \sqrt{\epsilon_{recon}})^2 \quad (12)$$

This inequality demonstrates that the generator’s performance on the training data is directly controlled by two terms that are actively minimized during our training procedure: the VAE’s functional reconstruction loss and the diffusion model’s denoising loss. This provides a strong theoretical justification for the stability and effectiveness of our framework.

H. Computational Details of the Dynamics-informed Prompter

Here, we detail the computational procedure for the Dynamics-informed Prompter module. The prompter takes a short observation sequence $X_L = \{x_0, x_1, \dots, x_{L-1}\}$ as input, where each state $x_i \in \mathbb{R}^{C \times H \times W}$ is a multi-channel spatial field. The entire sequence has a shape of (L, C, H, W) .

H.1. Explicit Physical Feature Extractor

This branch computes a set of macroscopic physical statistics to capture the global dynamics. Let $S_k \in \mathbb{R}^L$ be the time series for the k -th statistic.

Instantaneous Statistics. For each frame x_i in the sequence, we compute four statistics. After calculation, we sum the values over the channel dimension C to obtain a scalar value for each frame.

- **Spatial Mean (1st Moment):** The average value over the spatial domain.

$$\mu(x_i) = \frac{1}{H \times W} \sum_{h,w} x_{i,:,h,w}$$

- **Spatial Variance (2nd Moment):** The variance over the spatial domain.

$$\sigma^2(x_i) = \frac{1}{H \times W} \sum_{h,w} (x_{i,:,h,w} - \mu(x_i))^2$$

- **Energy (L2 Norm Squared):** A proxy for the total energy of the system.

$$E(x_i) = \sum_{h,w} \|x_{i,:,h,w}\|_2^2$$

- **Enstrophy (Squared Gradient Norm):** A proxy for the energy in the smallest scales, indicating turbulence.

$$\Omega(x_i) = \sum_{h,w} \|\nabla x_{i,:,h,w}\|_2^2$$

Temporal Aggregation. For each of the four statistic time series S_k (where $k \in \{\mu, \sigma^2, E, \Omega\}$), we compute two features to summarize its temporal evolution:

- **Temporal Mean:** The average value of the statistic over the sequence length L .

$$\bar{S}_k = \frac{1}{L} \sum_{i=0}^{L-1} S_{k,i}$$

- **Temporal Trend:** A simple approximation of the overall trend, calculated as the difference between the last and first values.

$$\Delta S_k = \frac{1}{L} (S_{k,L-1} - S_{k,0})$$

The final explicit prompt, p_{explicit} , is formed by concatenating these features for all statistics. If we compute $N_{\text{stats}} = 4$ statistics, the resulting vector has a shape of $N_{\text{stats}} \times 2 = 8$.

$$p_{\text{explicit}} = [\bar{S}_\mu, \Delta S_\mu, \bar{S}_{\sigma^2}, \Delta S_{\sigma^2}, \bar{S}_E, \Delta S_E, \bar{S}_\Omega, \Delta S_\Omega] \in \mathbb{R}^8$$

H.2. Implicit Spatiotemporal Encoder

This branch learns a latent representation of the microscopic dynamics from the raw data sequence.

Spectral Transformation. Each frame x_i is transformed into its frequency representation s_i using a 2D Fast Fourier Transform (FFT).

$$s_i = \text{FFT}(x_i) \in \mathbb{C}^{C \times H \times W}$$

We then stack the real and imaginary parts of the complex-valued spectra to form a real-valued tensor $\hat{s}_i \in \mathbb{R}^{2C \times H \times W}$, which is then flattened into a vector.

Temporal Encoding with GRU. The sequence of flattened spectra vectors $\{\hat{s}_0, \hat{s}_1, \dots, \hat{s}_{L-1}\}$ is fed into a Gated Recurrent Unit (GRU). The GRU iteratively updates its hidden state h_i based on the current input \hat{s}_i and the previous hidden state h_{i-1} :

$$h_i = \text{GRU}(\hat{s}_i, h_{i-1})$$

The final hidden state, h_{L-1} , which encapsulates the temporal evolution of the entire spectral sequence, is taken as the implicit prompt, p_{implicit} . If the GRU's hidden dimension is D_{hidden} , the shape of the implicit prompt is D_{hidden} .

$$p_{\text{implicit}} = h_{L-1} \in \mathbb{R}^{D_{\text{hidden}}}$$

H.3. Final Prompt Concatenation

The final dynamics-informed prompt p is obtained by concatenating the explicit and implicit vectors.

$$p = \text{concat}(p_{\text{explicit}}, p_{\text{implicit}})$$

The resulting prompt vector has a shape of $(8 + D_{\text{hidden}})$. This vector serves as the condition for the diffusion model.

I. Computational Details of the Weight VAE

Here, we elaborate on the architecture of the "node attention-based VAE". This architecture consists of a symmetric encoder E and decoder D , designed to process the weight graph W defined by heterogeneous node features. A key design choice is that this VAE does not use global pooling along the node dimension. Instead, it learns a dedicated latent variable for every node in the weight graph (i.e., each neuron or channel in the FNO model).

Input The input to the VAE is the weight graph W , which consists of L node feature tensors from different layers (e.g., lifting, FNO blocks, projection), denoted as $W = \{W_1, \dots, W_L\}$.

- $W_i \in \mathbb{R}^{B \times N_i \times D_i}$ is the node feature tensor for the i -th layer.
- B is the batch size.
- N_i is the number of nodes in the i -th layer (e.g., the number of output channels).
- D_i is the original feature dimension of the nodes in the i -th layer (e.g., $D_i = (C_{\text{in}} \times k_h \times k_w + 1)$).
- The total number of nodes is $N_{\text{total}} = \sum_{i=1}^L N_i$.

Encoder $E(W) \rightarrow (\mu_z, \sigma_z^2)$ The encoder E compresses the input heterogeneous weight graph W into per-node Gaussian distribution parameters.

- **Step 1: Node Projection.** To handle the heterogeneous dimensions D_i , we first use a set of layer-specific linear maps, $\text{MLP}_{\text{enc},i}$, to project all node features into a uniform, homogeneous embedding dimension d_{model} :

$$H_i = \text{GELU}(\text{MLP}_{\text{enc},i}(W_i)) \in \mathbb{R}^{B \times N_i \times d_{\text{model}}}$$

- **Step 2: Graph Re-assembly.** We concatenate all projected node tensors H_i along the node dimension ($\text{dim}=1$) to form a unified tensor H_{unified} containing all nodes in the graph:

$$H_{\text{unified}} = \text{Concat}[H_1, \dots, H_L] \in \mathbb{R}^{B \times N_{\text{total}} \times d_{\text{model}}}$$

- **Step 3: Node Attention Blocks.** H_{unified} is then passed through K standard Transformer encoder blocks to capture complex relationships between nodes. For the k -th Transformer block ($k = 1 \dots K$), the input is $H^{(k-1)}$ (where $H^{(0)} = H_{\text{unified}}$), and the computation proceeds as follows:

- **QKV Computation:** The block uses standard Multi-Head Self-Attention (MHA). The Query, Key, and Value are all derived from the same normalized input tensor $H_{\text{norm1}}^{(k)}$:

$$H_{\text{norm1}}^{(k)} = \text{LayerNorm}(H^{(k-1)})$$

$$Q^{(k)}, K^{(k)}, V^{(k)} \text{ are all derived from } H_{\text{norm1}}^{(k)}$$

- **Attention and Feedforward:**

$$H_{\text{attn}}^{(k)} = \text{MHA}(H_{\text{norm1}}^{(k)}, H_{\text{norm1}}^{(k)}, H_{\text{norm1}}^{(k)})$$

$$H_{\text{res1}}^{(k)} = H^{(k-1)} + H_{\text{attn}}^{(k)}$$

$$H_{\text{norm2}}^{(k)} = \text{LayerNorm}(H_{\text{res1}}^{(k)})$$

$$H_{\text{ffn}}^{(k)} = \text{FeedForward}(H_{\text{norm2}}^{(k)})$$

$$H^{(k)} = H_{\text{res1}}^{(k)} + H_{\text{ffn}}^{(k)}$$

After K layers, we obtain the encoder output $H_{\text{enc.out}} = H^{(K)}$.

- Step 4: Per-Node Latent Projection. $H_{\text{enc.out}}$ is directly projected into the per-node latent variable parameter space (dimension $2 \cdot d_z$):

$$\text{Params}_{\text{latent}} = \text{MLP}_{\text{latent}}(H_{\text{enc.out}}) \in \mathbb{R}^{B \times N_{\text{total}} \times (2 \cdot d_z)}$$

Finally, we split this tensor along the last dimension to get the mean μ_z and log-variance $\log \sigma_z^2$:

$$\mu_z, \log \sigma_z^2 = \text{Split}(\text{Params}_{\text{latent}}) \in \mathbb{R}^{B \times N_{\text{total}} \times d_z}$$

Reparameterization We use the standard reparameterization trick, sampling on a per-node basis:

$$\begin{aligned} \sigma_z &= \exp(0.5 \cdot \log \sigma_z^2) \\ \epsilon &\sim \mathcal{N}(0, I) \quad (\text{with shape } \mathbb{R}^{B \times N_{\text{total}} \times d_z}) \\ z &= \mu_z + \epsilon \cdot \sigma_z \in \mathbb{R}^{B \times N_{\text{total}} \times d_z} \end{aligned}$$

Decoder $D(z) \rightarrow \hat{W}$ The decoder D has a symmetric structure to the encoder.

- Step 1: Decoder Latent Projection. The per-node latent variable z is first projected back to the d_{model} dimension:

$$H_{\text{dec.in}} = \text{MLP}_{\text{dec.latent}}(z) \in \mathbb{R}^{B \times N_{\text{total}} \times d_{\text{model}}}$$

- Step 2: Decoder Attention Blocks. $H_{\text{dec.in}}$ is then passed through K Transformer decoder blocks (with independent weights). The computation is identical to the encoder’s Step 3, ultimately producing the decoder output $H_{\text{dec.out}} \in \mathbb{R}^{B \times N_{\text{total}} \times d_{\text{model}}}$.
- Step 3: Split and Inverse Projection. $H_{\text{dec.out}}$ is first split back into L tensors corresponding to the original layers, $H_{\text{dec.out},i} \in \mathbb{R}^{B \times N_i \times d_{\text{model}}}$. Then, each tensor is mapped back to its original, heterogeneous dimension D_i via its layer-specific inverse projection $\text{MLP}_{\text{dec},i}$:

$$\hat{W}_i = \text{MLP}_{\text{dec},i}(H_{\text{dec.out},i}) \in \mathbb{R}^{B \times N_i \times D_i}$$

The final reconstructed weight graph $\hat{W} = \{\hat{W}_1, \dots, \hat{W}_L\}$ matches the dimensions of the input W .

J. Additional Results

In this section, we provide additional experiments to further validate the robustness, generalization capabilities and design choices of our DynaDiff framework.

J.1. Statistical Significance Analysis of Main Results

We supplemented the key results in Table 1 with a statistical significance analysis, as suggested by the reviewer. We conducted pairwise Welch’s t-tests on the RMSE scores between DynaDiff and the best-performing baseline (underlined in Table 1) for each environment. The p -values, computed using `scipy.stats.ttest_ind`, are presented in Table 5.

On the Cylinder Flow and Navier-Stokes systems, the improvements by DynaDiff are statistically significant ($p < 0.05$). In the Kolmogorov Flow system, DPOT (0.079 ± 0.012) is closely to DynaDiff (0.077 ± 0.008) in the in-domain setting. However, in the more critical out-domain generalization task, DynaDiff’s (0.079 ± 0.011) advantage over GEPS (0.084 ± 0.017) approaches statistical significance ($p = 0.016$). In the Lambda-Omega system, DynaDiff’s advantage is not statistically prominent, indicating its performance is comparable to the SOTA baseline.

It is worth noting that DynaDiff consistently achieves strong generalization performance across all systems. Conversely, no single baseline demonstrates outstanding performance across all systems. This significantly indicates that DynaDiff, as a novel paradigm of weight-space learning, can generalize stably across different types of systems. This may be attributed to the fact that although the data for each PDE system varies greatly, DynaDiff models the weight distribution of a uniformly structured predictive model (e.g., FNO), making it more robust to dataset-level shifts. This cross-scenario stability highlights the superiority of DynaDiff’s weight-space learning paradigm.

Table 5. p -values for the statistical significance test (Welch’s t -test) of the RMSE difference between DynaDiff and the best-performing baseline from Table 1.

Cylinder Flow (96:400)		Lambda-Omega (12:39)		Kolmogorov Flow (12:39)		Navier-Stokes (24:121)	
In-domain	Out-domain	In-domain	Out-domain	In-domain	Out-domain	In-domain	Out-domain
3.26×10^{-9}	2.90×10^{-27}	0.15	0.53	0.49	0.016	0.002	3.79×10^{-18}

Table 6. Average out-domain RMSE of various observation length L .

L	2	4	6	8	10
Cylinder Flow	$0.069_{\pm 0.034}$	$0.068_{\pm 0.030}$	$0.066_{\pm 0.024}$	$0.064_{\pm 0.023}$	$0.065_{\pm 0.021}$
Navier-Stokes	$0.080_{\pm 0.034}$	$0.071_{\pm 0.033}$	$0.068_{\pm 0.026}$	$0.065_{\pm 0.013}$	$0.064_{\pm 0.012}$

J.2. Robustness to Environmental Extrapolation

Standard out-of-domain tests often involve interpolating between seen parameter values. A more challenging test is extrapolation, where the model must predict system behavior in a region of the parameter space far from the training data.

We conducted a difficult extrapolation experiment on the Cylinder Flow system, which is governed by two environmental parameters. We constructed a biased training set containing only environments from the top-right quadrant of the parameter space (i.e., where both parameters had high values). The model was then tested on the unseen bottom-left quadrant (i.e., where both parameters had low values).

The results are summarized in Table 7. As expected, this task is extremely challenging for all methods, and performance degrades as the training distribution becomes more biased (i.e., the seen environment ratio decreases). However, our method, DynaDiff, consistently maintains a significant performance advantage over the strong baseline models. This demonstrates that by learning a coherent representation of the weight-environment manifold, DynaDiff is more robust to extrapolation and less prone to catastrophic failure when faced with significant distributional shifts.

Table 7. Out-of-domain RMSE on the Cylinder Flow extrapolation task. The models were trained on a biased (top-right quadrant) subset of environments and tested on the unseen opposite quadrant.

Seen Env. Ratio	100%	90%	80%	70%	60%	50%
Poseidon	0.083	0.098	0.128	0.214	0.568	0.674
GEPS	0.082	0.126	0.136	0.143	0.183	0.654
DynaDiff (Ours)	0.065	0.077	0.095	0.107	0.121	0.228

J.3. Generalization to Unseen Governing Equations

To rigorously test the upper limits of our framework’s generalization ability, we designed a challenging experiment where the model must generalize to a completely unseen physical system with different governing equations.

We trained a single, unified generative model on three distinct PDE systems: Cylinder Flow, Lambda-Omega, and Navier-Stokes. The test was then performed on a completely held-out system: Kolmogorov Flow. To create a unified conditioning space for the prompter, we treated the combination of the PDE type and its specific physical coefficients as a single, high-dimensional environmental descriptor. When combining data from different systems with varying channel numbers, we padded the input channels with zero to maintain a consistent tensor shape.

We evaluated the performance of all methods in both zero-shot and few-shot settings. For a fine-grained comparison, we measured the average prediction length (number of autoregressive steps) for which the Structural Similarity (SSIM) index remains above 0.8. As shown in Table 8, our method demonstrates superior performance in both scenarios. In the zero-shot case, DynaDiff achieves the longest accurate prediction horizon. For the few-shot setting, where each model was fine-tuned on a single trajectory from the held-out system, DynaDiff still maintained its advantage, showcasing its ability to generate high-quality initial models that benefit more from minimal fine-tuning. This result suggests that our framework captures a more fundamental and transferable representation of dynamical systems, extending beyond simple parameter interpolation to the structure of the dynamics itself.

Table 8. Performance on the held-out Kolmogorov Flow system, measured by the average prediction length with SSIM $>$ 0.8. DynaDiff demonstrates superior generalization to a completely unseen physical law.

Method	Zero-shot	Few-shot (1 trajectory)
Poseidon	12.7	33.2
DPOT	14.0	41.4
MPP	10.1	38.7
DynaDiff (Ours)	15.9	46.5

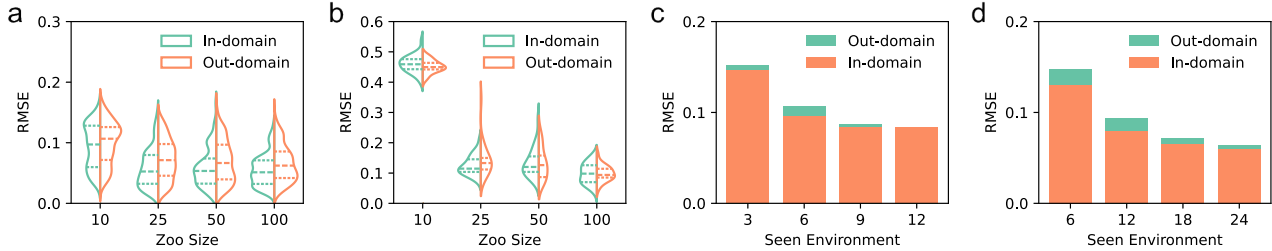


Figure 13. Robustness experiments. Impact of model zoo size on DynaDiff’s performance on (a) Cylinder Flow and (b) Lambda-Omega. Impact of the number of seen environments (e) on DynaDiff’s performance on (c) Kolmogorov Flow and (d) Navier-Stokes.

J.4. Ablation on Framework Design Choices

Our framework is composed of a two-stage generative stack (VAE + Latent Diffusion) that operates on a graph-based representation of weights. Here, we provide ablation studies to justify these key design choices against simpler alternatives.

Two-Stage vs. Single-Stage Generation. One could bypass the VAE and train a conditional diffusion model directly on the weight graphs. We compare our two-stage approach against such a single-stage, graph-structured conditional diffusion baseline (same to our Graph VAE’s architecture). As shown in Table 9, our method’s superior performance highlights the advantage of our design. The VAE first learns a semantically rich and low-dimensional manifold, which makes the subsequent generation task for the diffusion model more tractable and effective. This decoupling of representation learning from generation is crucial.

Graph vs. Sequence Representation. An alternative to our weight graph is to flatten the weights into a sequence and use a powerful sequence model like a Transformer. We compare our graph-based VAE against a sequence-based Transformer VAE. The results in Table 10 show that our graph-based approach is significantly more effective and parameter-efficient. By explicitly injecting the network’s architectural prior, the graph representation provides a much stronger and more suitable inductive bias for this task compared to relying on positional embeddings in a sequence.

Table 9. Ablation on the generative stack. Our two-stage (VAE + Latent Diffusion) approach significantly outperforms a direct, single-stage graph diffusion model.

Method	Cylinder Flow (RMSE)	Lambda-Omega (RMSE)
Single-Stage Graph Diffusion	0.112	0.238
Two-Stage (Ours)	0.065	0.091

J.5. Prompter

Here we conduct an experiment to validate the prompter’s ability to capture physically meaningful information from limited observations. We perform this analysis on the Cylinder Flow and Lambda-Omega systems. Specifically, we use the dynamics-informed prompt extracted by the prompter to regress the ground-truth environmental coefficients using a Random Forest regressor. The target coefficients are the Reynolds number (Re) and characteristic length (r) for Cylinder Flow, and the coefficient $Beta$ for the Lambda-Omega system. Figure 14 reports the regression performance on out-of-distribution environments. As shown, the predicted values correlate strongly with the true values, demonstrating that the prompter can reliably infer the physical parameters. This result indicates that the prompter has successfully learned to

Table 10. Ablation on weight representation. Our graph-based approach is more effective and parameter-efficient than a sequence-based Transformer approach.

Representation	VAE Params	CF (RMSE)	LO (RMSE)	KF (RMSE)	NS (RMSE)
Sequence-based	~1200M	0.129	0.208	0.152	0.143
Graph-based (Ours)	~380M	0.065	0.091	0.079	0.064

Table 11. Average RMSE of ablation study on domain initialization and function loss. 'w/o' stands for 'without'.

	Kolmogorov Flow		Navier-Stokes	
	In-domain	Out-domain	In-domain	Out-domain
w/o Domain Init	0.156 \pm 0.082	0.188 \pm 0.102	0.197 \pm 0.0102	0.201 \pm 0.098
w/o Function Loss	0.098 \pm 0.034	0.104 \pm 0.038	0.104 \pm 0.045	0.110 \pm 0.046
DynaDiff	0.077\pm0.008	0.079\pm0.011	0.060\pm0.013	0.064\pm0.012

extract the underlying dynamic signature from limited observation frames. It can therefore encode a discriminative and physically-grounded prompt to effectively guide the diffusion-based weight generation.

We also compare the performance of DynaDiff when using real environmental conditions e versus learned *prompt*, as shown in Table 13. Experimental results indicate that there is little difference in DynaDiff’s performance under the two settings. This demonstrates that the prompter effectively helps DynaDiff distinguish different environments for generating suitable weights.

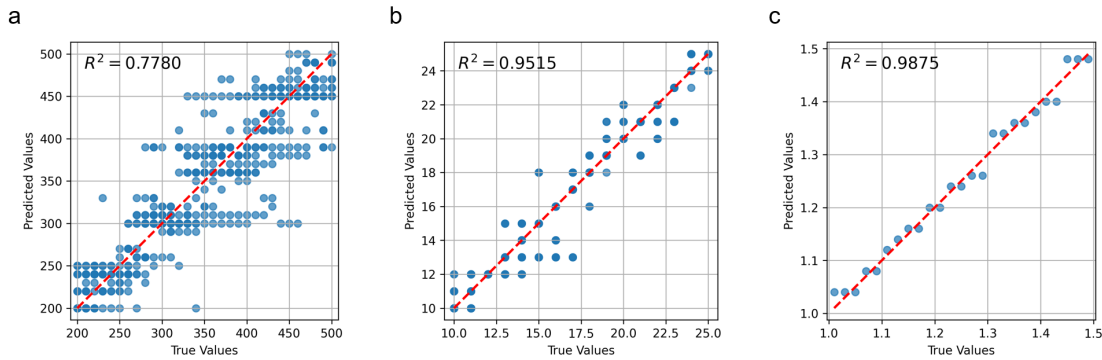


Figure 14. Prompter performance on the (a, b) Cylinder Flow and (c) Lambda-Omega systems.

J.6. Ablation on Auxiliary Supervisory Signal

While DynaDiff is capable of learning solely from the generative task, an optional auxiliary loss $\mathcal{L}_{aux} = \|e - \text{linear}(\text{prompt})\|_2^2$ can be integrated when ground-truth environmental conditions e are available. This signal can serve as a beneficial learning bias that aligns the learned latent features with explicit physical parameters, thereby enhancing the interpretability of the *prompt*.

Here, we conduct an ablation study on \mathcal{L}_{aux} using the Cylinder Flow and Lambda-Omega systems to examine whether the auxiliary loss can enhance the prompter to capture dynamic information from observation frames. The experimental results are shown in Table 14. We find that adding \mathcal{L}_{aux} only yields a marginal improvement in DynaDiff’s generalization performance. This strongly demonstrates that DynaDiff’s core generalization ability primarily stems from the dynamical information extracted from the observation sequence X_L , rather than a dependency on the ground-truth environment e . The role of \mathcal{L}_{aux} is essentially to introduce a beneficial learning bias to guide training, without providing extra knowledge. With or without \mathcal{L}_{aux} , DynaDiff acquires physical information through the L observation frames.

We also find that with \mathcal{L}_{aux} , the regression performance of the dynamics-informed *prompt* learned by the Prompter on the true physical coefficients increases for the Cylinder Flow system (Figure 15). This suggests that while \mathcal{L}_{aux} may not add extra dynamical information to the *prompt*, its constraint during training helps the prompter extract more interpretable

Table 12. Out-domain RMSE under random neuron permutation (10%).

Method	CF	LO
DynaDiff + Permutation	0.068 \pm 0.027	0.090 \pm 0.013
DynaDiff	0.065 \pm 0.021	0.091 \pm 0.015

Table 13. Average RMSE of ablation study on prompter.

	Cylinder Flow		Navier-Stokes	
	In-domain	Out-domain	In-domain	Out-domain
Prompter	0.063 \pm 0.023	0.065 \pm 0.021	0.060 \pm 0.013	0.064\pm0.012
Environmental condition	0.050\pm0.014	0.061\pm0.027	0.058\pm0.007	0.065 \pm 0.006

dynamical representations.

Table 14. Average RMSE of ablation study on \mathcal{L}_{aux} .

	Cylinder Flow		Lambda-Omega	
	In-domain	Out-domain	In-domain	Out-domain
DynaDiff	0.063 \pm 0.023	0.065 \pm 0.021	0.088 \pm 0.013	0.091 \pm 0.015
with \mathcal{L}_{aux}	0.059 \pm 0.028	0.065 \pm 0.025	0.090 \pm 0.011	0.089 \pm 0.013

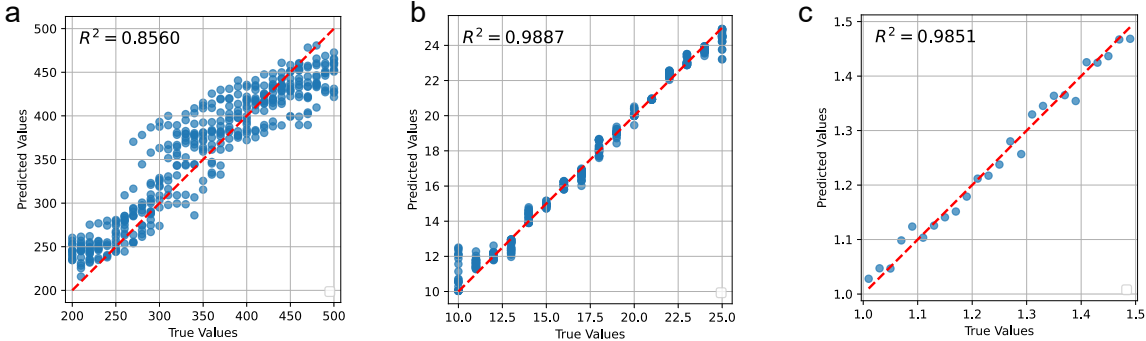


Figure 15. Prompter performance with \mathcal{L}_{aux} on the (a, b) Cylinder Flow and (c) Lambda-Omega systems.

J.7. Time and Memory Cost of Meta-learning Methods

We compare the inference cost of DynaDiff against all meta-learning methods for a single environment on the Navier-Stokes system, as shown in Table 15. Compared to other adaptive methods, the additional inference cost of DynaDiff comes from the generator, which includes latent space denoising and VAE decoding. The results show that, thanks to the low dimensionality of the latent space, the actual incremental cost is very small. Note that the GPU memory for DynaDiff reported in Figure 10b of the main paper was calculated during parallel inference across multiple environments, which is why it appears larger.

J.8. End-to-End Computational Cost Analysis

We provide a detailed breakdown of the end-to-end computational costs, using the Cylinder Flow system (96 training environments, 400 test environments) as an example on a single A100 GPU.

Training and Testing Overhead. Table 16 compares the total training and testing costs across all methods. DynaDiff’s model zoo construction time (3,432s) is comparable to or less than the training time of several foundation models

Table 15. Time and memory costs of meta-learning methods on the Navier-Stokes system.

	DyAd	LEADS	CoDA	GEPS	CAMEL	DynaDiff
GPU Memory (GB)	0.796	0.902	1.122	1.028	0.846	2.168
Time Cost (s)	5.96	29.23	20.88	28.80	20.25	31.96

(e.g., DPOT requires 11,392s). The total training overhead of DynaDiff remains in the same order of magnitude as foundation models, while its test-time adaptation is highly efficient.

Inference Efficiency vs. Test-Time Fine-Tuning. To demonstrate DynaDiff’s deployment advantage, we designed a challenge experiment: all meta-learning methods were allowed to perform test-time fine-tuning using observation frames until they reached performance comparable to DynaDiff ($\text{RMSE} \leq 0.075$) at a learning rate of $1e^{-5}$. As shown in Table 17, existing adaptive methods require approximately $3\times$ additional inference overhead to achieve comparable performance, while DynaDiff requires no gradient computation at test time.

These results demonstrate that DynaDiff’s generative paradigm offers a significant efficiency advantage in time-sensitive deployment scenarios, achieving comparable or better accuracy with approximately $3\times$ faster adaptation.

Table 16. Training and testing cost (seconds) and GPU memory (GB) on Cylinder Flow.

Method	Training cost (s) & GPU memory	Testing cost (s) & GPU memory
FNO	4,225 (53G)	828 (28G)
DPOT	11,392 (51G)	1,855 (23G)
Poseidon	3,334 (47G)	1,634 (4.8G)
MPP	6,194 (63G)	906 (4.4G)
DyAd	674 (2.9G)	273 (2.3G)
LEADS	1,422 (5.1G)	401 (2.3G)
CoDA	790 (3.8G)	719 (3.8G)
GEPS	902 (3.3G)	425 (2.1G)
CAMEL	772 (3.4G)	310 (2.3G)
DynaDiff	3,432 + 2,217 (6.6G)	629 (4.2G)

 Table 17. Test-time fine-tuning cost to reach DynaDiff-level accuracy ($\text{RMSE} \leq 0.075$) on Cylinder Flow.

Metric	DyAd	LEADS	CoDA	GEPS	CAMEL
Finetuning cost (s)	1,812	1,903	2,195	1,858	1,765
Avg. Adaptation Time (s / env)	4.53	4.76	5.49	4.65	4.41

J.9. Analysis of Generator Parameter Count and Baseline Scalability

The parameter count of DynaDiff’s generator is approximately 380M, which is positioned between prior meta-learning methods ($< 50M$) and foundation models ($> 500M$). This difference in generator size is determined by the methodological paradigm: prior meta-learning approaches typically use a hypernetwork to generate low-dimensional context vectors, whereas our approach generates the complete weights of a 1M-parameter FNO model.

We conduct an experiment, using CAMEL and GEPS as representatives, where we increased their hypernetwork depth (3 layers) and width (15,000-dim) to scale them to a comparable size. The performance on the Cylinder Flow and Lambda-Omega systems is shown in Table 19. The results indicate that simply increasing the parameter count of the meta-learning baselines does not effectively improve their performance ceiling. This suggests that the performance bottleneck for these methods is not the parameter count itself. In contrast, our framework provides a new alternative that achieves higher generalization performance.

Table 18. RMSE statistics over 100 samples on 10 environments.

Method	Mean	Std	Worst	Best
DynaDiff (CF)	0.061	0.002	0.068	0.056
DynaDiff (NS)	0.063	0.003	0.067	0.058

Table 19. RMSE performance of scaled-up meta-learning baselines vs. DynaDiff.

	Cylinder Flow		Lambda-Omega	
	In-domain	Out-domain	In-domain	Out-domain
GEPS-10M	0.079	0.082	0.094	0.092
GEPS-450M	0.083	0.084	0.097	0.100
CAMEL-5M	0.089	0.094	0.104	0.103
CAMEL-400M	0.097	0.099	0.102	0.104
DynaDiff-380M	0.063	0.065	0.088	0.091

K. Architectures of Expert Models

In our main experiments, we deploy three neural operators as expert models for DynaDiff: FNO, UNO, and WNO. Here, taking the Cylinder Flow system as an example, we list the parameter composition and hyperparameter settings of these operators.

FNO We adopt the code from the open-source repository (Kossaifi et al., 2024) as the implementation for FNO. For the NS system, the weight composition of FNO is as follows:

```
lifting.fcs.0: torch.Size([128, 5])
lifting.fcs.1: torch.Size([64, 129])
fno_blocks.convs.0: torch.Size([64, 6209])
fno_blocks.channel_mlp.0.0: torch.Size([32, 65])
fno_blocks.channel_mlp.0.1: torch.Size([64, 34])
fno_blocks.convs.1: torch.Size([64, 6209])
fno_blocks.channel_mlp.1.0: torch.Size([32, 65])
fno_blocks.channel_mlp.1.1: torch.Size([64, 34])
fno_blocks.convs.2: torch.Size([64, 6209])
fno_blocks.channel_mlp.2.0: torch.Size([32, 65])
fno_blocks.channel_mlp.2.1: torch.Size([64, 34])
fno_blocks.convs.3: torch.Size([64, 6209])
fno_blocks.channel_mlp.3.0: torch.Size([32, 65])
fno_blocks.channel_mlp.3.1: torch.Size([64, 34])
projection.fcs.0: torch.Size([128, 65])
projection.fcs.1: torch.Size([2, 129])
% \end{minted}
```

UNO We adopt the code from the open-source repository (Kossaifi et al., 2024) as the implementation for UNO. For the NS system, the weight composition of UNO is as follows:

```
lifting.fcs.0: torch.Size([256, 5])
lifting.fcs.1: torch.Size([64, 257])
fno_blocks.0.convs.0: torch.Size([64, 5185])
fno_blocks.0.channel_mlp.0: torch.Size([32, 65])
fno_blocks.0.channel_mlp.1: torch.Size([64, 34])
fno_blocks.1.convs.0: torch.Size([64, 5185])
fno_blocks.1.channel_mlp.0: torch.Size([32, 65])
fno_blocks.1.channel_mlp.1: torch.Size([64, 34])
fno_blocks.2.convs.0: torch.Size([128, 10369])
fno_blocks.2.channel_mlp.0: torch.Size([64, 129])
fno_blocks.2.channel_mlp.1: torch.Size([128, 66])
horizontal_skips.0.conv.weight: torch.Size([64, 64])
```

```
projection.fcs.0: torch.Size([256, 129])
projection.fcs.1: torch.Size([2, 257])
% \end{minted}
```

WNO We adopt the code from the open-source repository (Tripura & Chakraborty, 2023) as the implementation for WNO. For the NS system, the weight composition of WNO is as follows:

```
conv.0.weights_a1 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_a2 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_h1 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_h2 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_v1 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_v2 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_d1 | Shape: torch.Size([40, 40, 5, 5])
conv.0.weights_d2 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_a1 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_a2 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_h1 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_h2 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_v1 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_v2 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_d1 | Shape: torch.Size([40, 40, 5, 5])
conv.1.weights_d2 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_a1 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_a2 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_h1 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_h2 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_v1 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_v2 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_d1 | Shape: torch.Size([40, 40, 5, 5])
conv.2.weights_d2 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_a1 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_a2 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_h1 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_h2 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_v1 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_v2 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_d1 | Shape: torch.Size([40, 40, 5, 5])
conv.3.weights_d2 | Shape: torch.Size([40, 40, 5, 5])
w.0.weight | Shape: torch.Size([40, 40, 1, 1])
w.0.bias | Shape: torch.Size([40])
w.1.weight | Shape: torch.Size([40, 40, 1, 1])
w.1.bias | Shape: torch.Size([40])
w.2.weight | Shape: torch.Size([40, 40, 1, 1])
w.2.bias | Shape: torch.Size([40])
w.3.weight | Shape: torch.Size([40, 40, 1, 1])
w.3.bias | Shape: torch.Size([40])
fc0.weight | Shape: torch.Size([40, 5])
fc0.bias | Shape: torch.Size([40])
fc1.weight | Shape: torch.Size([128, 40])
fc1.bias | Shape: torch.Size([128])
fc2.weight | Shape: torch.Size([3, 128])
fc2.bias | Shape: torch.Size([3])
% \end{minted}
```

Since the parameters of normalization layers are determined by the dataset and are not controlled by the environment, we do not enable normalization layers in all operators (they are also disabled by default in the original code).