T3R: Deeper Test-Time Adaptation for Graph Neural Networks via Gradient Rotation

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) deployed in real-world systems typically have fixed weights, often leading to degraded performance under distribution shifts. This issue can be mitigated by conventional fine-tuning, but in many real-world cases, collecting labeled data is expensive or infeasible. A potential approach is Test-Time Training (TTT), which adapts models' weights using unlabeled test data, yet it is typically limited to shallow updates that affect only a subset of model parameters. We propose T3R, leveraging multiple Rotograd matrices to improve task affinity between the target and auxiliary tasks, essential for effective test-time training. T3R further introduces a rotation technique that reorients self-supervised signals using these matrices to create surrogate gradients for the target task, allowing deeper adaptation across nearly the entire architecture. Empirically, T3R reduces MAE by 0.172 points over standard inference in regression datasets and achieves at least 9.37% relative improvement on cross-domain OGB classification benchmarks compared to models without adaptation. These results highlight the potential to develop an adaptation pipeline for graph-based systems, particularly in settings where conventional fine-tuning or retraining is infeasible.

1 Introduction

The recent success of deep neural networks has driven advances in model deployment. In practice, these models are typically frozen after training and used unchanged throughout their operational lifetime. However, this static deployment paradigm exposes a risk of performance degradation when encountering unseen data under distribution shifts, commonly referred to as concept drift (Lu et al., 2018). Recently, this paradigm has begun to be challenged by advances in test-time adaptation (Tandon et al., 2025; Niu et al., 2022; Liang et al., 2021; Wang et al., 2021). Among these methods, we focus on Test-Time Training (TTT), a simple yet effective approach (Sun et al., 2020). It employs a Y-shaped model with a shared encoder and two heads, one for the target task and one for an auxiliary Self-Supervised Learning (SSL). At test time, TTT exploits the label-free objective from the auxiliary head to adjust the model weights without access to ground-truth labels before inference. This motivates the potential of such an approach to maintain the model's capability under distribution shifts, which often occur in real-world systems.

However, this challenge is further pronounced in graph-centric, large-scale infrastructures such as 5G networks, computer networks, and water systems, where sensor drift over time can shift the data distribution and labeled samples are scarce (Wang et al., 2022; Ferriol-Galmés et al., 2022; Degeler et al., 2024). In such settings, fine-tuning is impractical. For example, in the water domain, training data are typically synthesized by simulations that require complete knowledge, an assumption rarely met in practice (Truong et al., 2024). Consequently, models deployed on unseen Water Distribution Networks (WDNs) must adapt with limited data, insufficient for additional simulation or fine-tuning.

Besides, since these systems are naturally graph-structured, GNNs provide an appealing inductive bias and are therefore widely used to model them (Zhang et al., 2024; Kerimov et al., 2024; Salem et al., 2024). However, TTT for graphs and GNN remains in infancy. This could be due to the difficulty of choosing a

graph-related SSL task that meaningfully aligns with a specific target objective, and most TTT variants were designed for and validated mainly on image data (Sun et al., 2020; Liu et al., 2021; Gandelsman et al., 2022).

More critically, existing TTT methods are fundamentally constrained, as only one component of the entire architecture, the encoder, is adapted and involved in test-time inference, while the remaining components stay fixed or do not directly participate in decision making. This restricts both efficiency and robustness under distribution shift.

In this paper, we propose Test-Time Training with layer-wise Rotograds (T3R), a novel adaptation approach that utilizes the idea of task-specific gradient rotation from RotoGrad (Javaloy & Valera, 2022) at test-time. In particular, TTT (Sun et al., 2020) often requires a Y-shaped deep model with a shared encoder and two heads: one for the target task and another for an auxiliary SSL objective. During training, T3R simultaneously optimizes the GNN weights and a set of learnable rotation matrices to promote alignment across two tasks. At test time, while typical TTT approaches update only the encoder and SSL decoder, T3R additionally updates the main decoder solely using these matrices and SSL signals, allowing a deeper adaptation over almost the entire architecture. We evaluate T3R on multiple regression datasets from DiTEC-WDN Dataset (DWD) (Truong et al., 2025) and classification datasets from Open Graph Benchmark (OGB) (Hu et al., 2020) under Out-Of-Distribution (OOD) and cross-domain settings. The results show that T3R consistently outperforms baseline methods on the main task.

Overall, our key contributions are as follows:

- We introduce T3R method, with a rotation technique that leverages layer-wise rotation matrices to improve main and auxiliary objectives during training, while producing surrogate gradients that enable adaptation of nearly the entire model at test time.
- We empirically select self-supervised objectives on graphs, particularly when leveraging the adaptability of TTT to improve main task performance at test time.
- We conduct extensive OOD experiments across multiple regression and classification datasets, demonstrating its effectiveness, with up to 38.38% relative improvement in RMSE on DWD and 49.82% in ROC-AUC on OGB, compared to standard inference without adaptation.

2 Related work

2.1 Test-Time Training

TTT (Sun et al., 2020) is a post-training approach emerging in the computer vision community to enhance the model adaptation in unseen scenarios. In the image domain, Sun et al. (2020) employed *2-D rotation prediction*, in which the deep model was trained to predict the rotation angle of an augmented test image. Alternative SSL tasks include Masked Autoencoders, which aim to reconstruct masked images (Gandelsman et al., 2022), and contrastive learning (Liu et al., 2021) with an additional objective on test-time statistical properties such as mean and standard deviation estimation. Nevertheless, it remains unclear whether these methods can be applied to different modalities, such as the graph. This also raises concerns about the design of a compatible auxiliary task for the graph-related primary one.

2.2 Self-Supervised Learning on Graphs

The choices of auxiliary tasks vary from simple generative tasks, such as nodal attribute masking (Feng et al., 2020), edge masking (You et al., 2020), and node degree estimation (Li et al., 2023), to sophisticated tasks, including Deep Graph Infomax (Veličković et al., 2019), Graph Contrastive Coding (Qiu et al., 2020), and GraphMAE (Hou et al., 2022). These advanced Graph SSL methods often improve generalizability but come with higher costs due to increased forward passes and intensive augmentations per sample. Thus, it is crucial to choose a SSL on graphs that can effectively balance exploration (leveraging diverse augmentations) and exploitation (allocating more adaptation steps to test-time data) within limited budgets, a challenge that our proposed method directly addresses.

2.3 Graph Neural Networks

GNNs have become widely used for learning representations from graph-structured data. Early spectral approaches, such as ChebNet (Defferrard et al., 2016) and GCN (Kipf & Welling, 2017), encode graph signals into higher-dimensional representations using spectral filters. In contrast, spatial approaches such as the Message Passing Neural Network (MPNN)(Gilmer et al., 2017) leverage graph topology to propagate and aggregate information from local node neighborhoods. This framework has served as a foundational framework for numerous influential variants, including GAT (Veličković et al., 2018), GATRes (Truong et al., 2024), GIN (Xu et al., 2019), and DeepGCN (Li et al., 2019). In this study, GNNs play the role of the main architecture integrated with TTT to tackle the distribution-shift problem.

2.4 Multi-Task Learning

Multi-Task Learning (MTL) paradigm refers to an efficient training strategy where a single model is designed to address multiple related tasks. Nevertheless, jointly optimizing many objectives introduces negative knowledge transfer attributed to the conflict in gradients. One approach is to break down the joint objective into individual subproblems and devise a Pareto solution (Lin et al., 2019). In addition, such a non-dominated solution could be achieved by gradient re-aggregation (Navon et al., 2022). Alternatively, Suteu & ke Guo (2020) suggested an additional auxiliary loss term to penalize negative gradient conflicts and encourage orthogonality. Likewise, Yu et al. (2020) proposed to dissect such conflicts by adjusting the gradient directions more orthogonally. In the context of TTT, although MTL approaches generally do not contribute to test-time model performance, we identify RotoGrad (Javaloy & Valera, 2022) as an exception, improving both training and test-time model performance. Furthermore, we extend its ability to generate pseudo-gradients for the main head at test time, therefore enhancing the adaptability of GNNs.

3 Methodology

We first describe essential notations for graph modality and briefly revisit TTT. We then integrate RotoGrad, a method that encourages positive task-specific gradient alignment as a separate optimization objective during training, into TTT by adjusting the GNN architecture and auxiliary task selection strategy. In light of this, we propose T3R, which leverages gradient rotation to adapt almost the entire architecture, including the shared encoder, the SSL decoder, and all layers of the main decoder except for the last layer, given only the input at test-time.

3.1 Revisiting TTT on graphs

We operate a joint GNN on a graph $G = \{\mathcal{V}, \mathcal{E}\}$ in which $\mathcal{V}$ and $\mathcal{E}$ denote sets of nodes and edges, respectively. The node set is related to a nodal feature $\mathbf{X}$ while the edge set is associated with an edge feature matrix $\mathbf{E}$. The connectivity between node pairs is denoted as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $|\mathcal{V}|$ represents the number of nodes.

Following TTT, the method relies on a joint architecture consisting of three components: the encoder θ_e, the main head θ_m, and the SSL head θ_s. These components should share the same architecture (e.g., number of layers) except for the first and last number of channels. This architecture either gains the benefits of multi-objective optimization during training or leverages self-supervised signals to partially optimize the model for unseen scenarios during inference. The primary head is designed to optimize the main task, while a secondary head targets an auxiliary objective that does not depend on raw labels and provides guidance to enhance main-task performance at test-time. Mathematically, the objective is described as:

$$\begin{aligned} \min_{\mathbf{\Theta}} \; & \mathcal{L}_{main}(\mathbf{X}, \mathbf{E}, \mathbf{A}, \mathbf{Y}_m; \theta_e, \theta_m) \\ & + \lambda \mathcal{L}_{ssl}(\mathbf{X}, \mathbf{E}, \mathbf{A}, \mathbf{Y}_s; \theta_e, \theta_s) \end{aligned} \tag{1}$$

where we aim to optimize $\mathbf{\Theta} = \{\theta_e, \theta_m, \theta_s\}$, and introduce the coefficient λ to control the contribution of the auxiliary loss $\mathcal{L}_{ssl}$. This loss can be formulated in various ways. For example, in the embedding reconstruction task, the loss is calculated by representations of the original observation $\mathbf{X}$ and the augmented $\tilde{\mathbf{X}}$. In this

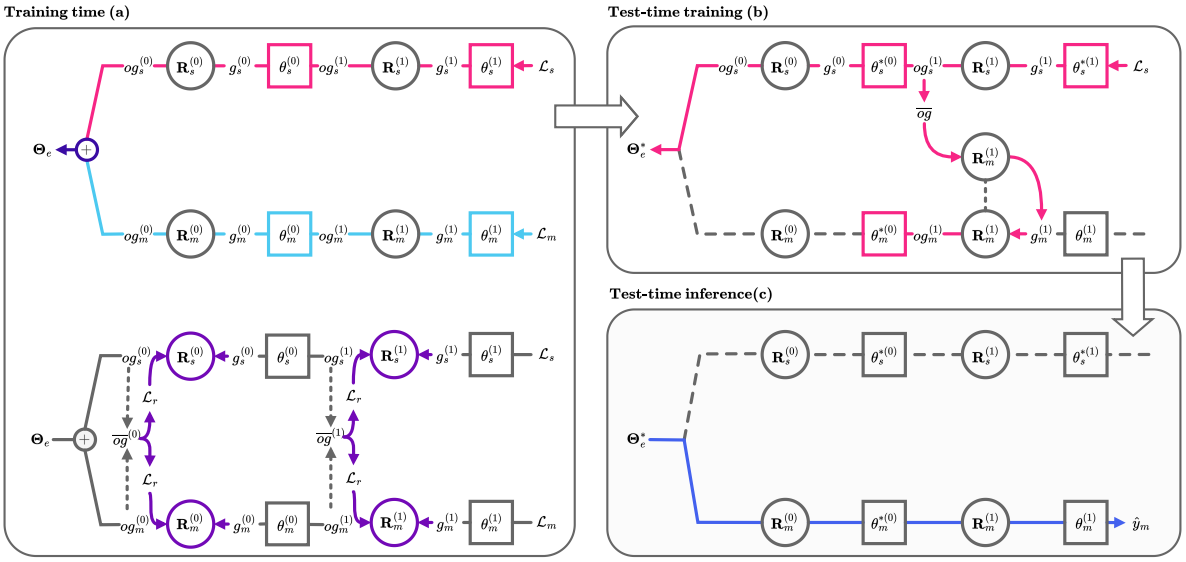


Figure 1: **Illustration of the T3R approach using a Y-shaped architecture, where the shared encoder and both the main and SSL decoders each consist of a two-layer GNN.** Highlight colors indicate the gradient flow and the components updated by each loss: pink for the auxiliary loss $\mathcal{L}_{ssl}$, cyan for the main loss $\mathcal{L}_{main}$, and purple for the rotation loss $\mathcal{L}_{rot}$ (Equation 5). Subfigure (a) shows two simultaneous objectives: updating the GNN weights (`TOP`) and the rotation matrices (`BOTTOM`). Subfigure (b) illustrates the rotation technique used to generate the surrogate gradient for the main branch. Subfigure (c) depicts the final inference involving the updated encoder and the partially updated main decoder with its fixed classifier. Leveraging rotation-aligned surrogate gradients allows a deeper adaptation of the joint model even in the absence of the main task loss.

case, the ground truth Y_s is defined as $\tilde{\mathbf{X}}$. A similar computation can be applied to edge features using the "clean" $\mathbf{E}$ and augmented $\tilde{\mathbf{E}}$.

During deployment, incoming data is assumed to originate from an unknown distribution that might differ from the training distribution. In a standard setting, a TTT step can optimize $\{\theta_s, \theta_e\}$ using the SSL signals given the test sample before the final inference:

$$\theta_e^*, \theta_s^* = \arg\min_{\theta_\mathbf{e}, \theta_\mathbf{s}} \lambda_{test} \mathcal{L}_{ssl}(\mathbf{X}, \mathbf{E}, \mathbf{A}, Y_s; \theta_e, \theta_s) \tag{2}$$

We refer to Equation 2 as a single adaptation step t applied to a single batch, as it can be iterated multiple times. The updated encoder weight θ_e^* is then employed for the final inference by a forward pass f:

$$\hat{\mathbf{X}} = f(\mathbf{X}, \mathbf{E}, \mathbf{A}; \theta_e^*, \theta_m) \tag{3}$$

In TTT, the main head θ_m remains unchanged during testing due to the absence of supervised signals (i.e., no gradient observed). These updated weights are tied to a specific testing sample and must be reverted to the ones at the end of training time before processing the subsequent sample.

3.2 Task affinity measurement

Our goal is to measure task affinity as the angle between gradient vectors of the main and SSL heads, obtained during the training phase (Du et al., 2020). Here, we denote the gradient g_e, g_s, g_m, which correspond to the encoder, the SSL decoder, and the main decoder, respectively. Since the encoder is shared between both heads, its gradient g_e is computed as a weighted sum of the other two gradients from the main and SSL tasks,

implied from Equation 1. The angle α between two gradient vectors is calculated as:

$$\alpha = \frac{\mathbf{g}_m \cdot \mathbf{g}_s}{\|\mathbf{g}_m\|\|\mathbf{g}_s\|} = \langle \mathbf{g}_m, \mathbf{g}_s \rangle \tag{4}$$

In SSL selection, we use cosine similarity to identify the auxiliary task that best aligns with the main task. Specifically, the auxiliary task with the highest positive gradient alignment (α) is prioritized. Additionally, cosine similarity serves as the training objective to align task-specific gradients during training.

3.3 Test-Time Training with layer-wise Rotograds

Previous works have proven that positive gradient alignment α mitigates gradient conflict, responsible for joint-training underperformance, while improving task affinity (Li et al., 2024). In other words, a higher task affinity aligns the task gradients in the same direction, so minimizing one objective also minimizes the other. This is essential, particularly for TTT, as the model components are updated using SSL signals but eventually contribute to the main task's inference at test-time.

However, in a vanilla joint architecture, such an affinity between two tasks remains fixed or slow to improve, possibly due to the gradient interference (Yu et al., 2020) and the near-orthogonality of arbitrary vectors in high-dimensional spaces (Luisto, 2025). To mitigate this, we adopt Rotograd, designed for enhancing task affinity (Javaloy & Valera, 2022). Specifically, an orthogonal rotation matrix $\mathbf{R} \in \mathcal{SO}(d)$, where d is the number of hidden channels, is prepended to each decoder head in the joint model.

We propose Test-Time Training with layer-wise Rotograds (T3R), which extends rotation matrices to every GNN layer of two decoders in the joint model (Figure 1). In the context of a two-task setting, we denote rotation matrices $\mathbf{R}_m^{(l)}$ and $\mathbf{R}_s^{(l)}$ for the main and SSL heads at layer l, respectively.

Considering a backward computational graph, we attempt to maximize the cosine similarity between the mean of the original (pre-rotation) gradients and task-specific (post-rotation) gradients:

$$\mathcal{L}_{rot}: \max_{\mathbf{R}_m^{(l)}, \mathbf{R}_s^{(l)}} \langle \mathbf{R}_m^{(l)\top} \mathbf{g}_m^{(l)}, \overline{\mathbf{og}}^{(l)} \rangle + \langle \mathbf{R}_s^{(l)\top} \mathbf{g}_s^{(l)}, \overline{\mathbf{og}}^{(l)} \rangle \tag{5}$$

where $\langle \cdot, \cdot \rangle$ recalls cosine similarity, $\overline{\mathbf{og}}^{(l)} = \frac{1}{2}(\mathbf{og}_m^{(l)} + \mathbf{og}_s^{(l)})$ denotes as the mean of the main original gradient $\mathbf{og}_m^{(l)}$ and the SSL original gradient $\mathbf{og}_s^{(l)}$. When both main and SSL decoders have a single layer ($l = 1$), Equation 5 resembles the traditional Rotograd, which mainly aims to rotate task-specific gradients to an original homogeneous direction and, therefore, makes two task gradients align closely. It is worth noting that this rotation objective $\mathcal{L}_{rot}$ is optimized cyclically alongside the joint objective in Equation 1. This training scheme is guaranteed to converge as long as the learning rate of the rotation optimizer η_{rot} is lower (Javaloy & Valera, 2022).

To optimize the per-layer rotation objectives, one could share a single optimizer across all layers. However, in practice, we use a separate optimizer for each layer, as this empirically yields a more stable learning curve. Accordingly, we scale the rotation learning rate for layer l by a factor of $\frac{1}{l}$ (i.e., $\eta_{rot}^{(l)} = \frac{\eta_{rot}}{l}$).

When multiple Rotograd layers are applied and have converged, the task-specific gradients can be rotated back toward the mean original gradient $\overline{\mathbf{og}}^{(l)}$ at any layer l. During test-time training when the main loss $\mathcal{L}_{main}$ is absent, $\overline{\mathbf{og}}^{(l)}$ is approximately aligned with the SSL original gradient $\mathbf{og}_s^{(l)}$ ($\overline{\mathbf{og}}^{(l)} \approx \mathbf{og}_s^{(l)}$). This allows us to employ a trick of redirecting the gradient $\mathbf{g}_s^{(l)}$ to the next-layer main gradient:

$$\tilde{\mathbf{g}}_m^{(l+1)} = \mathbf{R}_s^{(l)\top} \mathbf{g}_s^{(l)} \mathbf{R}_m^{(l)} \tag{6}$$

Given the surrogate main gradient $\tilde{\mathbf{g}}_m^{(l+1)}$, the optimizer backpropagates to the first layer of the main decoder. Note that all learnable parameters are optimized except for these main rotation matrices $\mathbf{R}_m^{(l)}$ and the last layer (i.e., classifier). We employ these updated weights of the shared encoder and partially updated main decoder following its fixed classifier to perform the final inference.

Lastly, we observe that abrupt hyperparameter shifts during test-time training and inference can be beneficial. In particular, we increase the coefficient λ_{test} and reduce the augmentation intensity, such as lowering the

node masking and edge dropping rates. These shifts amplify the SSL signals for the adaptation and mitigate failure in excessively challenging SSL tasks, for example, when strong augmentations coincide with a small number of TTT samples.

4 Experiments

We evaluate model adaptability across two domains: water and molecules. We describe the selected datasets, experimental setup, and results from key experiments: SSL task selection, regression individual benchmarks and cross-validation for *next-state prediction*, and cross-domain *graph property prediction* for classification. Finally, we present ablation studies of the proposed T3R approach.

4.1 Collections

We outline the two collections in the water and molecule domains. Detailed statistics of their selected datasets are reported in Appendix A.1.

DiTEC-WDN (DWD). (Truong et al., 2025) consists of numerous water networks with scenarios simulated as steady-state snapshots, under distinct configurations. Each snapshot is represented as a static, directed graph. Node features included pressure (m) and demand (m^3/s). Edge features comprise temporal attributes, flow rate (m^3/s), and head loss (m), as well as static attributes: pipe length (m), diameter (mm), and initial status (unitless). Thirteen consecutive snapshots were grouped into non-overlapping windows. The first 12 snapshots were used as inputs, while the last snapshot was used for prediction. Datasets are pre-split into training, validation, and test sets (60/20/20).

Open Graph Benchmark (OGB). (Hu et al., 2020) contains a variety of graph datasets in the wild. Unlike regression data in DWD, we evaluated baselines on molecular graphs with categorical features, primarily focusing on graph property classification. In particular, we chose `ogbg-molbace` and `ogbg-molbbbp` for their similarity in size, input dimensionality, and number of output channels. The former dataset involves binding affinity prediction for molecules against the BACE-1 enzyme, while the latter predicts blood-brain barrier permeability. Their splits follow an 80/10/10 ratio.

4.2 SSL task selection via Gradient Alignment

Datasets. Two small graph datasets, `dwd-Anytown` and `dwd-19pipe` were used for this experiment. In particular, each dataset was downsampled to 10,000 non-overlapping windows. We selected 6,000 training windows from the `dwd-Anytown` training set and tested on 2,000 windows from the `dwd-19pipe` test set.

Baselines. We implemented several SSL tasks as baselines, since TTT has seen limited use in the graph domain. To ensure efficiency, we focused on simple generative-based SSL tasks that require at most two forward passes. For instance, the *node embedding MSE* task computes embeddings for the original and augmented graphs in two forward passes, then measures their difference using the Mean Squared Error (MSE) loss. This constraint helps mitigate the computational limitations typically encountered at inference time. These SSL baselines are further described in the Appendix A.2.

Experiment setup. We chose GAT (Veličković et al., 2018) as a backbone for all components in the joint model, using the configuration shown in Table 7. The only exceptions were final classifiers in both decoders, which differed based on the desired output channels for each task. The main task was fixed as *next-state prediction*, while various SSL tasks were explored, resulting in changes to the final layer of the SSL decoder. Specifically, this layer was followed by either a global mean pooling operation for graph-level SSL tasks or a sparse scattering after a linear transformation to output edge feature predictions for edge-level SSL tasks. We then performed a standard TTT step (t = 1) and compared it to standard inference (t = 0) across three runs. Additionally, we reported the average training Gradient Alignment, along with Mean Absolute Error (MAE) and relative improvement measured on the test set.

SSL Task	Test MAE(t=0)↓	Test MAE(t=1)↓	Relative Gain(%)↑	Train Gradient Alignment
Node mask	5.8103	**3.9924**	**31.29**	0.12001
Node emb Hinge	5.5840	5.2572	5.85	0.01584
Graph emb Hinge	5.3039	5.0863	4.10	0.01310
Edge mask	8.0307	8.4091	-4.71	0.00793
Graph emb MSE	5.0118	4.9935	0.37	0.00029
Edge spd	**4.2763**	4.0688	4.85	-0.00013
Node emb MSE	5.9647	5.9644	0.01	-0.00033
Node emb CLIP	5.5818	5.5763	0.10	-0.00033
Graph emb CLIP	5.6845	5.6555	0.51	-0.00033
Edge emb CLIP	5.8712	5.8727	-0.03	-0.00033
Edge emb MSE	6.5948	6.6065	-0.18	-0.00049
Edge emb Hinge	5.1011	5.3939	-5.74	-0.00894

Table 1: **SSL Task Evaluation Based on Main Task Performance and Gradient Alignment.** The standard deviation was omitted for clarity. $t = 0$ refers to a standard inference, and $t = 1$ denotes a single adaptation step applied. While most SSL tasks exhibit near-zero gradient angle with the main task, *node masking* stands out as the strongest positive gradient alignment and correspondingly the highest relative gain, highlighting its superior adaptability.

Results. Table 1 shows the main-task performance of SSL tasks. While most of the auxiliary tasks are nearly orthogonal to the main task, likely a consequence of the curse of dimensionality, three tasks (*node mask*, *node emb hinge*, and *graph emb hinge*) show a more consistent increase in relative gain when their positive alignments exceed a small threshold (> 0.01).

In addition, the top-scoring task, *node masking*, aligns with Masked Autoencoders, a similar SSL task previously used for TTT in the vision domain (Gandelsman et al., 2022), and was used as the default auxiliary task in subsequent experiments. Alongside main-task performance metrics, we empirically observe that positive gradient alignment correlates with improved model adaptability. A formal theoretical justification of this relationship is left for future work.

4.3 Next-State Prediction on DWD

Datasets. After determining the best compatible task recalled as *node masking*, we evaluated adaptation baselines on a regression task. In particular, there are two relevant experiments: per-benchmark evaluation and cross-validation. The former includes `dwd-ky4`, `dwd-ky6`, `dwd-ky7`, `dwd-npcl1`, `dwd-wa1`, `dwd-ctown`, `dwd-ky5`, `dwd-ky10`, and `dwd-ky13`, whereas the latter is restricted to a smaller set due to the significantly higher memory cost of training a joint multi-network model. Consequently, we construct a subset of five representative WDNs: (`dwd-anytown`, `dwd-jilin`, `dwd-ctown`, `dwd-wa1`, and `dwd-exn`), sampled to balance diversity across networks for the cross-validation experiment.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	0.9051 ± 0.0783	0.5023 ± 0.1172	0.9497 ± 0.0419	0.3916 ± 0.0969	-2.3600 ± 5.1430
Joint Training	0.8831 ± 0.1123	0.4782 ± 0.2135	0.9376 ± 0.0616	0.3621 ± 0.2051	-1.0399 ± 3.8281
Tent	0.8985 ± 0.0842	0.5135 ± 0.1154	0.9459 ± 0.0452	0.3915 ± 0.0989	-2.6587 ± 5.5148
TTT	0.8832 ± 0.1115	0.4806 ± 0.2145	0.9376 ± 0.0612	0.3640 ± 0.2045	-1.0140 ± 3.8964
TTT + Rotograd	0.8541 ± 0.1310	0.5195 ± 0.2562	0.9136 ± 0.0867	0.4122 ± 0.2211	**-0.1343** ± 5.0412
T3R (ours)	**0.9492** ± 0.0827	**0.3095** ± 0.2591	**0.9721** ± 0.0510	**0.2196** ± 0.2115	-0.9875 ± 4.9584

Table 2: **Average results over nine benchmark datasets**. A single adaptation step is applied for all methods except the first two (no-adaptation baselines). Values show mean ± pooled standard deviation over three runs across nine datasets. In terms of RMSE, T3R achieves a 38.38 % relative improvement over ERM, while most other adaptation methods underperform ERM in this regression setting.

Experiment setup. All experiments were conducted on a cluster equipped with an A100 40GB GPU. We used GAT as the backbone for all components in the joint model, following the same configuration as in the previous experiment. We reported MAE, Root Mean Squared Error (RMSE), Pearson Correlation Coefficient (PCC), Coefficient of Determination (R^2), and Nash-Sutcliffe Efficiency (NSE) (ranging from -inf to 1, higher is better), the latter being a widely used metric in hydrological modeling (Legates & McCabe Jr., 1999). For data augmentation, we applied node and edge dropout by default, with specific settings for the training and TTT phases detailed in Table 7. Notably, we used a stronger augmentation rate during training and a weaker rate during TTT phase, as this empirically led to better main-task performance.

Baselines. All baselines share the same main task (*next-state prediction*) and its compatible SSL task (*node masking*). Their training and test-time training (TTT) procedures are summarized below:

- **Empirical Risk Minimization (ERM)**: an I-shaped GNN encoder-decoder architecture aims to optimize the primary objective. During inference time, ERM produces the final prediction without test-time training.
- **Joint Training**: a Y-shaped architecture with two heads for the main and auxiliary objectives, optimized jointly. Like ERM, it does not perform any adaptation at test time.
- **Tent** (Wang et al., 2021): an I-shaped GNN architecture aims to optimize the primary objective. At test-time, an entropy minimization loss is applied to optimize the batch norm statistics (mean and standard deviation) while other parameters remain fixed.
- **TTT** (Sun et al., 2020) is trained with a standard multi-objective optimization. At test time, only the encoder and SSL decoder were updated via the SSL loss. The final prediction is performed using an updated encoder and a fixed main decoder.
- **TTT + Rotograd** (Javaloy & Valera, 2022) is identical to TTT at test-time, but training objective additionally included the rotation loss $\mathcal{L}_{\text{rot}}$ to learn two rotation matrices prepended to decoders, which remain frozen during TTT phase.
- **T3R (ours)** extends the GNN architecture with layer-wise rotation matrices. At test-time, T3R leverages the rotation technique to update nearly all model weights. The final prediction is produced by the updated encoder, the partially updated main decoder, and the fixed classifier.

Per-benchmark average result. Table 2 showcases baseline performance over nine benchmark datasets. Prepending two rotation matrices in the TTT + Rotograd setup offers no measurable improvement in the GNN's adaptability. By contrast, extending to layer-wise matrices and applying the rotation technique, T3R consistently outperforms all other baselines on every metric except NSE, likely due to anomalous results on the `dwd-wa1` dataset (see individual results in Appendix A.6). Surprisingly, ERM remains competitive on some metrics (e.g., R^2 and PCC), while other adaptation methods struggle in this regression setting.

Method	Step	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
Tent	$0^{\dagger}$	0.9307 ± 0.0729	0.3604 ± 0.1503	0.9637 ± 0.0391	0.2227 ± 0.0869	-1.8122 ± 4.7780
	1	0.9224 ± 0.0762	0.3924 ± 0.1539	0.9592 ± 0.0409	0.2422 ± 0.0958	-2.0129 ± 5.1665
	10	0.9224 ± 0.0762	0.3923 ± 0.1540	0.9592 ± 0.0409	0.2421 ± 0.0960	-2.0127 ± 5.1661
	20	0.9222 ± 0.0763	0.3926 ± 0.1541	0.9591 ± 0.0410	0.2424 ± 0.0959	-2.0134 ± 5.1650
	40	0.9223 ± 0.0762	0.3923 ± 0.1540	0.9592 ± 0.0409	0.2422 ± 0.0959	-2.0139 ± 5.1663
TTT	$0^{\ddagger}$	0.9336 ± 0.0764	0.3321 ± 0.1579	0.9652 ± 0.0407	0.2075 ± 0.1004	-0.6480 ± 2.4796
	1	0.9340 ± 0.0763	0.3330 ± 0.1570	0.9654 ± 0.0406	0.2078 ± 0.1001	-0.6484 ± 2.4803
	10	0.9355 ± 0.0750	0.3493 ± 0.1412	0.9662 ± 0.0399	0.2196 ± 0.0905	-0.6561 ± 2.4817
	20	0.9363 ± 0.0726	0.3459 ± 0.1354	0.9667 ± 0.0385	0.2175 ± 0.0867	-0.6543 ± 2.4769
	40	0.9357 ± 0.0707	0.3425 ± 0.1314	0.9664 ± 0.0375	0.2158 ± 0.0837	-0.6536 ± 2.4724
TTT + Rotograd	0	0.9525 ± 0.0388	0.2797 ± 0.1022	0.9756 ± 0.0200	0.1699 ± 0.0630	0.0486 ± 1.4518
	1	0.9526 ± 0.0380	0.2812 ± 0.0981	0.9757 ± 0.0196	0.1712 ± 0.0607	0.0533 ± 1.4483
	10	0.9599 ± 0.0282	0.2696 ± 0.0793	0.9795 ± 0.0144	0.1635 ± 0.0488	0.0469 ± 1.4710
	20	0.9612 ± 0.0257	0.2694 ± 0.0738	0.9801 ± 0.0131	0.1633 ± 0.0443	0.0350 ± 1.4924
	40	0.9622 ± 0.0235	0.2742 ± 0.0719	0.9807 ± 0.0120	0.1674 ± 0.0441	0.0119 ± 1.5374
T3R (ours)	0	0.9673 ± 0.0254	0.2467 ± 0.0686	0.9833 ± 0.0130	0.1523 ± 0.0416	0.0071 ± 1.8002
	1	0.9672 ± 0.0259	**0.2373** ± 0.0715	0.9832 ± 0.0133	**0.1446** ± 0.0426	**0.0158** ± 1.7817
	10	0.9687 ± 0.0245	0.2410 ± 0.0740	0.9840 ± 0.0125	0.1491 ± 0.0473	0.0106 ± 1.6943
	20	0.9689 ± 0.0251	0.2522 ± 0.0814	0.9841 ± 0.0129	0.1604 ± 0.0564	-0.1068 ± 1.7659
	40	**0.9696** ± 0.0243	0.2969 ± 0.0939	**0.9845** ± 0.0125	0.1959 ± 0.0649	-0.8078 ± 3.6360

$0^{\dagger}$ and $0^{\ddagger}$ indicate ERM and Joint Training without adaptation, respectively.

Table 3: **Cross-validation results with more adaptation steps.** Although T3R consistently outperforms the baselines, we observe a common trend across all methods: increasing the number of adaptation steps can lead to degraded performance in this regression task.

Cross-validation result. In the per-benchmark experiment, the GNN equipped with the proposed adaptation methods still encounters test-set topologies that are similar to those seen during training. To assess robustness more rigorously, we further conduct a five-fold cross-validation study. Specifically, we trained baselines on the training sets of four WDNs and evaluated them on the test set of a held-out network.

Table 3 highlights the performance of baselines on unseen topologies with varied adaptation steps. With a single step ($t = 1$), T3R achieves its best performance, particularly in RMSE and MAE. However, deeper adaptation tends to reduce performance relative to a shallow adaptation across all methods, except for R^2 and PCC, which consistently improve for T3R.

4.4 Graph Property Prediction on OGB

One-one adaptation. We reused the same baselines and assessed on a classification problem in the molecule domain: `ogbg-molbace` and `ogbg-molbbbp`. In addition, multiple adaptation steps were enabled to observe the performance changes. In particular, we trained GIN (Xu et al., 2019) models on one dataset, then performed TTT and evaluated on the test set of another. As the input and output features were categorical and the main task was *graph-level binary classification*, we prepended an OGB-provided atom encoder to the shared encoder and appended a global addition readout to the main decoder. In addition, the main task was evaluated using ROC-AUC and relative gain w.r.t. the standard inference without any adaptation.

For the SSL decoder, we reused *node masking* as the SSL task, based on its stable loss signals and highest positive cosine similarity between task gradients (see Appendix A.3). The auxiliary loss $\mathcal{L}_{ssl}$ was computed as the sum of cross-entropy losses between reconstructed and "clean" input atom features, whereas the main loss $\mathcal{L}_{main}$ was defined as a simple binary cross-entropy between predicted and true labels.

Results. Table 4 presents the performance of the T3R approach evaluated after a fixed number of TTT steps. Results are averaged over five independent runs. T3R achieved the highest ROC-AUC scores, 0.5608, when adapting from `ogbg-molbace` to `ogbg-molbbbp`, and 0.6682 in the reverse direction. In addition, the proposed approach yielded significant relative improvement of 8.65%-69.56% over standard inference without adaptation. These gains could be attributed to the weight updates of a partial main encoder. However, this improved accuracy comes at the cost of increased runtime due to the additional gradient updates, as shown in Appendix A.5.

Adapt. Method	Step	ogbg-molbace→ ogbg-molbbbp		ogbg-molbbbp → ogbg-molbace	
		Test ROC-AUC↑	Rel. Gain (%)↑	Test ROC-AUC↑	Rel. Gain (%)↑
Tent	$0^{\dagger}$	0.4621 ± 0.0586	0	0.4460 ± 0.2032	0
	1	0.4682 ± 0.0646	1.32	0.4352 ± 0.2194	-2.43
	10	0.4678 ± 0.0641	1.23	0.4356 ± 0.2194	-2.32
	20	0.4677 ± 0.0640	1.21	0.4360 ± 0.2192	-2.24
	40	0.4678 ± 0.0640	1.23	0.4364 ± 0.2191	-2.15
TTT	$0^{\ddagger}$	0.4582 ± 0.0519	0	0.5228 ± 0.1538	0
	1	0.4587 ± 0.0522	0.11	0.5294 ± 0.1420	1.26
	10	0.4656 ± 0.0634	1.62	0.5399 ± 0.1308	3.28
	20	0.4595 ± 0.0562	0.29	0.5366 ± 0.1326	2.64
	40	0.4507 ± 0.0585	-1.64	0.5192 ± 0.1550	-0.69
TTT + Rotograd	0	0.4386 ± 0.0495	0	0.4352 ± 0.1420	0
	1	0.4384 ± 0.0494	-0.04	0.4421 ± 0.1480	1.60
	10	0.4310 ± 0.0417	-1.73	0.4882 ± 0.1464	12.19
	20	0.4307 ± 0.0444	-1.78	0.5260 ± 0.1384	20.86
	40	0.4339 ± 0.0495	-1.07	0.5411 ± 0.1302	24.34
T3R (ours)	0	0.4651 ± 0.0561	0	0.3941 ± 0.0755	0
	1	0.5054 ± 0.0478	8.65	0.5234 ± 0.1274	32.81
	10	0.5243 ± 0.0826	12.71	**0.6682** ± 0.0217	**69.56**
	20	0.5391 ± 0.0391	15.90	0.6248 ± 0.0797	58.53
	40	**0.5608** ± 0.0516	**20.57**	0.6420 ± 0.0667	62.90

$0^{\dagger}$ and $0^{\ddagger}$ indicate ERM and Joint Training without adaptation, respectively.

Table 4: **Cross-domain adaptation from `ogbg-bace` to `ogbg-bbbp` and vice versa.** Each method was evaluated over five independent runs. Careful tuning of the adaptation step led to greater relative improvement across methods.

For other adaptation techniques, unlike in the regression setting, performance improved slightly on this classification task. Under longer adaptation, Tent tended to converge early, while TTT + Rotograd showed a roughly linear increase or decrease depending on the difficulty of the cross-domain shift. Similar to T3R, TTT required careful tuning of the adaptation step t to maximize the benefit from adaptability. Otherwise, over-adaptation caused an abrupt drop in main-task performance.

4.5 Ablation studies

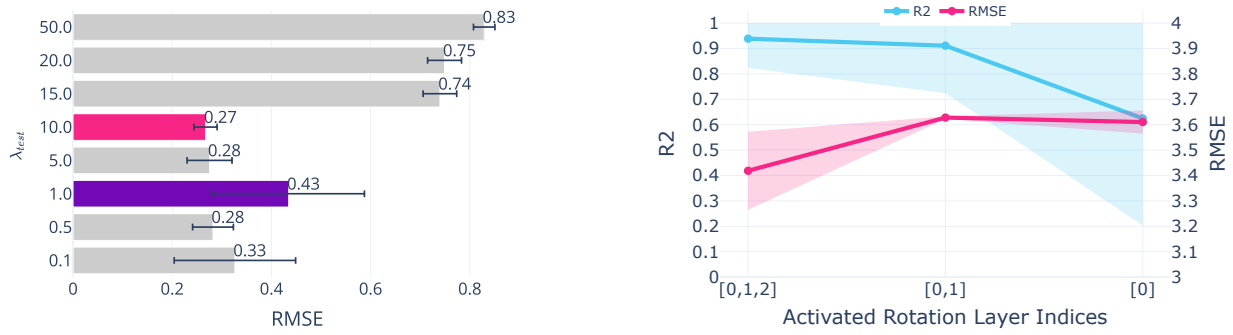
General setup. The experiments followed an adaptation from `dwd-anytown` (1-year, smallest topology) to `dwd-exn` (24-hour, largest topology). We assessed the impact of hyperparameter shifts at test-time and layer-wise rotation matrices on T3R. For additional visualizations, we refer the reader to Appendix A.7.

The impact of auxiliary loss coefficient λ. We fixed $\lambda_{\text{train}} = 1$ during training and varied λ_{test} across multiple values at inference. Notably, this choice affected the SSL signals from Equation 2 used to update the model's weights. Figure 2a shows the impact on RMSE, with similar trends observed across other metrics. Surprisingly, $\lambda_{\text{test}} = 10$ achieves the lowest RMSE. Compared to using the same value for TTT and inference ($\lambda_{\text{test}} = \lambda_{\text{train}} = 1$), tuning λ_{test} improves main task. However, tuning λ to excessively large values leads to a decline in performance, likely due to over-adaptation.

The impact of layer-wise rotation layers. We employed a default 3-layer GAT, sequentially ablated the Rotograd layers in both main and SSL decoders, and reported their mean and std. values over 3 runs (Figure 2b). Fewer rotation layers increased uncertainty and degraded R^2 and RMSE performance.

5 Conclusion

We propose T3R, a novel adaptation method on graphs that enhances affinity between the main and auxiliary tasks during training. At test-time training, T3R applies a rotation-based transformation to redirect gradients



(a) **Impact of the Auxiliary Loss Coefficient.**

(b) **Impact of rotation layers.**

Figure 2: **Ablation studies.** Subfigure 2a illustrates the effect of the auxiliary-loss coefficient λ_{test}. The purple bar corresponds to the fixed setting without coefficient shifting (i.e., $\lambda_{train} = \lambda_{test} = 1$), while the pink bar highlights the best RMSE performance. Slight variations in λ_{test} at inference tend to improve the target task. Subfigure 2b showcases the impact of rotation layers in T3R. Indices indicate the layers at which rotation matrices are attached, trained and test-time activated in the main and SSL decoders. Removing rotation layers increases uncertainty and degrades main-task performance in terms of R^2 and RMSE.

from the auxiliary loss toward the main-task decoder, deeply adapting nearly the entire model without requiring access to ground-truth labels. Empirically, T3R consistently outperforms standard baselines on both regression and classification datasets. These results highlight its potential for building robust, label-free adaptation pipelines in real-time graph-based systems, such as water networks, and in scenarios where fine-tuning or retraining is impractical.

References

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Victoria Degeler, Mostafa Hadadian, Erkan Karabulut, Alexander Lazovik, Hester van het Loo, Andrés Tello, and Huy Truong. Ditec: Digital twin for evolutionary changes in water distribution networks. In Tiziana Margaria and Bernhard Steffen (eds.), *Leveraging Applications of Formal Methods, Verification and Validation. Application Areas*, pp. 62–82. Springer Nature Switzerland", oct 2024. ISBN 978-3-031-75390-9. URL `https://link.springer.com/chapter/10.1007/978-3-031-75390-9_5`.

Yunshu Du, Wojciech M. Czarnecki, Siddhant M. Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity, 2020. URL `https://arxiv.org/abs/1812.02224`.

Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural network for semi-supervised learning on graphs. In *NeurIPS'20*, 2020.

Miquel Ferriol-Galmés, José Suárez-Varela, Jordi Paillissé, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Building a digital twin for network optimization using graph neural networks. *Computer Networks*, 217:109329, 2022. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2022.109329. URL `https://www.sciencedirect.com/science/article/pii/S1389128622003681`.

Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei A Efros. Test-time training with masked autoencoders. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=SHMi1b7sjXk.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=T8wHz4rnuGL.

Bulat Kerimov, Riccardo Taormina, and Franz Tscheikner-Gratl. Towards transferable metamodels for water distribution systems with edge-based graph neural networks. *Water Research*, 261:121933, 2024. ISSN 0043-1354. doi: https://doi.org/10.1016/j.watres.2024.121933. URL https://www.sciencedirect.com/science/article/pii/S0043135424008340.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

David R. Legates and Gregory J. McCabe Jr. Evaluating the use of "goodness-of-fit" measures in hydrologic and hydroclimatic model validation. *Water Resources Research*, 35(1):233–241, 1999. doi: https://doi.org/10.1029/1998WR900018. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/1998WR900018.

Dongyue Li, Aneesh Sharma, and Hongyang R. Zhang. Scalable multitask learning using gradient-based estimation of task affinity. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 1542–1553, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671835. URL https://doi.org/10.1145/3637528.3671835.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9266–9275, 2019. doi: 10.1109/ICCV.2019.00936.

Jintang Li, Ruofan Wu, Wangbin Sun, Liang Chen, Sheng Tian, Liang Zhu, Changhua Meng, Zibin Zheng, and Weiqiang Wang. What's behind the mask: Understanding masked graph modeling for graph autoencoders. In *KDD*, pp. 1268–1279. ACM, 2023.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 4465–4478. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/2f73168bf3656f697507752ec592c437-Paper.pdf.

Jian Liang, Dapeng Hu, Yunbo Wang, Ran He, and Jiashi Feng. Source data-absent unsupervised domain adaptation through hypothesis transfer and labeling transfer, 2021. URL https://arxiv.org/abs/2012.07297.

Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qingfu Zhang, and Sam Kwong. Pareto multi-task learning. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, pp. 12037–12047, 2019.

Yuejiang Liu, Parth Kothari, Bastien van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21808–21820. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/b618c3210e934362ac261db280128c22-Paper.pdf.

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

Rami Luisto. A short survey on almost orthogonal vectors in a few specific large dimensions. *arXiv preprint arXiv:2510.23609*, 2025.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16428–16446. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/navon22a.html.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16888–16905. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/niu22a.html.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 1150–1160, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403168. URL https://doi.org/10.1145/3394486.3403168.

Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

Aly K. Salem, Ahmad F. Taha, and Ahmed A. Abokifa. Graph neural networks-based dynamic water quality state estimation in water distribution networks. *Engineering Applications of Artificial Intelligence*, 138:109426, 2024. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2024.109426. URL https://www.sciencedirect.com/science/article/pii/S0952197624015847.

Yu Sun, Xiaolong Wang, Liu Zhuang, John Miller, Moritz Hardt, and Alexei A. Efros. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020.

Mihai Suteu and Yi ke Guo. Regularizing deep multi-task networks using orthogonal gradients, 2020. URL https://openreview.net/forum?id=SJeXJANFPr.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context, 2025. URL https://arxiv.org/abs/2512.23675.

Huy Truong, Andrés Tello, Alexander Lazovik, and Victoria Degeler. Graph neural networks for pressure estimation in water distribution systems. *Water Resources Research*, 60(7):e2023WR036741, 2024. doi: https://doi.org/10.1029/2023WR036741. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2023WR036741. e2023WR036741 2023WR036741.

Huy Truong, Andrés Tello, Alexander Lazovik, and Victoria Degeler. Ditec-wdn: A large-scale dataset of hydraulic scenarios across multiple water distribution networks. *Scientific Data*, 12(1):1733, 2025. doi: 10.1038/s41597-025-06026-0. URL https://doi.org/10.1038/s41597-025-06026-0.

Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rklz9iAcKQ.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

Haozhe Wang, Yulei Wu, Geyong Min, and Wang Miao. A graph neural network-based digital twin for network slicing management. *IEEE Transactions on Industrial Informatics*, 18(2):1367–1376, 2022. doi: 10.1109/TII.2020.3047843.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5812–5823. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/3fe230348e9a12c13120749e3f9fa4cd-Paper.pdf.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

Zhiyu Zhang, Wenchong Tian, Chenkaixiang Lu, Zhenliang Liao, and Zhiguo Yuan. Graph neural network-based surrogate modelling for real-time hydraulic prediction of urban drainage networks. *Water Research*, 263:122142, 2024. ISSN 0043-1354. doi: https://doi.org/10.1016/j.watres.2024.122142. URL https://www.sciencedirect.com/science/article/pii/S0043135424010418.

A Appendix

A.1 Dataset Description

Both DWD and OGB include several datasets served for a variety of tasks. However, only datasets with compatible inputs and identical tasks are suitable for model adaptation. Table 5 summarizes these datasets along with key statistics, as presented in Section 4.1.

Dataset	#records	#nodes	#edges
dwd-19pipe	673,846	12	21
dwd-anytown	673,846	19	41
dwd-jilin	673,846	27	34
dwd-wa1	673,846	121	169
dwd-npcl1	673,846	337	399
dwd-ky7	673,846	481	604
dwd-ky6	673,846	543	647
dwd-ky4	673,846	959	1158
dwd-ctown	1,846	388	444
dwd-ky5	1,846	420	505
dwd-ky13	1,846	778	944
dwd-ky10	1,846	920	1061
dwd-exn	1,846	1891	2467
ogbg-molbace	2,039	24.1	51.9
ogbg-molbbbp	1,513	34.1	73.7

Table 5: **Summary of evaluation datasets.** For the water domain, we select eight networks simulated over one year and five networks simulated over 24 hours, where each record is processed as non-overlapping graph windows. In OGB, each record is a unique molecule graph.

A.2 SSL Task Description

We describe the graph-related SSL tasks used in the Experiment 4.2.

- **Attribute Masking** (Feng et al., 2020; You et al., 2020): Reconstruct masked attributes from augmented graphs. We refer to this as *node masking* (node-level) and *edge masking* (edge-level).

- **MSE reconstruction in Embedding Space**: Instead of reconstructing raw attributes, we compute MSE loss between embeddings. This includes *node emb MSE*, *edge emb MSE*, and *graph emb MSE*.

- **One-Sided Contrastive (Hinge) Loss**: To avoid the cost of full contrastive learning, we apply a Hinge loss that pulls augmented (positive) embeddings toward the clean (anchor) embedding. Named as *{node, edge, graph} emb Hinge*, depending on the applied level.

- **One-Sided CLIP Loss**: Inspired by CLIP (Radford et al., 2021), we treat graph indices as pseudo labels and train the SSL decoder to predict them using the inner-product of its output embedding vector. Denoted as *{node, edge, graph} emb CLIP*.

- **Shortest Path Distance Regression** (Li et al., 2020): The SSL decoder regresses shortest path distances between nodes (computed via Dijkstra algorithm) based on edge weights. Referred to as *edge SPD*.

A.3 SSL task selection on OGB

We re-ran the same task selection experiment but on the OGB benchmark (Section 4.4). In particular, we trained a GIN model on `ogbg-molbbbp` and performed TTT on `ogbg-molbace`. Table 6 confirms a consistent observation: *node masking* remains the top-performing SSL objective with the strongest train gradient alignment.

SSL Task	Test MAE(t=0)↓	Test MAE(t=1)↓	Relative Gain(%)↑	Train Gradient Alignment
Node mask	0.51	0.56	**10.07**	**0.10**
Node emb CLIP	**0.31**	**0.26**	-15.94	0.07
Node emb Hinge	0.46	0.47	1.43	0.02
Graph emb CLIP	0.41	0.41	0.35	0.02
Graph emb MSE	0.52	0.51	-2.83	0.00
Node emb MSE	0.48	0.49	1.32	0.00
Graph emb Hinge	0.42	0.34	-19.17	-0.03

Table 6: **SSL Task Selection on OGB with GIN.** Note that GIN does not support edge attributes, so edge-related SSL tasks are excluded.

A.4 Hyperparameter setting

Table 7 reports the optimal hyperparameters for T3R and other baselines introduced in Section 4.3. Note that baselines use different normalization types: ERM and Tent employ BatchNorm, whereas TTT and TTT + Rotograd use LayerNorm for both tasks, as this empirically yields better performance.

Hyperparameter	NSP	GPP
Graph Conv.	GAT	GIN
#layers	3	3
# hidden channels	64	300
Norm type	z-score	-
Normalization layer	LayerNorm	BatchNorm
Main Readout	-	Global add pool
Main loss	MSE	BCE
SSL loss	Node mask	Sum of CE losses
Epochs	30	50
Batch size	128	480
Train | TTT weight decay	1e-5 | 0.	1e-6 | 1e-6
Train | TTT learning rate	5e-3 | 5e-4	0.01 | 1e-3
Train | TTT Aug. NodeDrop	0.95 | 0.1	0.95 | 0.1
Train | TTT Aug. EdgeDrop	0.1| 0.1	0.1 | 0.1
Train | TTT coefficient λ	1.0 | 10.	1.0 | 1.0
Dropout rate	0.5	0.5
Activation	ReLU	ReLU
Optimizer	Adam	Adam
Scheduler	Cosine Annealing	-

Table 7: **Hyperparameter settings for the Next-State Prediction (NSP) and Graph Property Prediction (GPP) tasks.**

A.5 Relative time measurement

Extending layer-wise rotation matrices in GNNs increases computational cost. We report the computation overhead of adaptation methods on OGB in Section 4.4. In particular, T3R is 1.35x-9.35x slower than ERM at inference, depending on the adaptation step (t) (Table 8). The trade-off seems acceptable given the substantial performance gains (+8.65% to +69.56% on OGB) and the fact that GNNs often perform best with shallow architectures. In addition, this accuracy-efficiency trade-off can be managed by tuning t; better improvements are often achieved in the first step (t=1).

	Relative runtime in seconds (slowdown relative to ERM)				
Method	**t=0**	**t=1**	**t=10**	**t=20**	**t=40**
Tent	**2.99 (1.00x)**	**3.47 (1.16x)**	**4.80 (1.61x)**	**6.04 (2.02x)**	**8.72 (2.91x)**
TTT	3.29 (1.10x)	3.75 (1.25x)	7.20 (2.40x)	9.99 (3.34x)	15.54 (5.19x)
TTT+Rotograd	3.50 (1.17x)	4.10 (1.37x)	7.90 (2.64x)	11.32 (3.78x)	17.80 (5.95x)
T3R (Ours)	4.05 (1.35x)	5.24 (1.75x)	12.74 (4.25x)	19.45 (6.50x)	28.27 (9.45x)

Table 8: **Computation overhead on OGB**. t denotes the number of adaptation steps. Results may vary depending on the hardware configuration. We conducted this experiment on a local NVIDIA GeForce RTX 3060 Laptop GPU (6GB VRAM).

A.6 Per-dataset Next-State Prediction results

The full results of Experiment 4.3 across five metrics for each water-related dataset are presented in the following tables:

Method	**R2↑**	**RMSE↓**	**PCC↑**	**MAE↓**	**NSE↑**
ERM	0.9297 ± 0.0030	0.5844 ± 0.0303	0.9623 ± 0.0019	0.4284 ± 0.0198	-11.8799 ± 1.2133
Joint Training	0.9405 ± 0.0018	**0.4137** ± 0.0077	0.9687 ± 0.0011	**0.2655** ± 0.0117	-10.6401 ± 2.3255
Tent	0.9279 ± 0.0013	0.6027 ± 0.0143	0.9611 ± 0.0008	0.4196 ± 0.0108	-12.9159 ± 0.6678
TTT	0.9408 ± 0.0017	0.4169 ± 0.0125	0.9689 ± 0.0011	0.2681 ± 0.0135	-10.4934 ± 2.0832
TTT + Rotograd	**0.9427** ± 0.0109	0.4420 ± 0.2380	**0.9699** ± 0.0062	0.3600 ± 0.2704	**-5.0650** ± 4.8243
T3R (ours)	0.9352 ± 0.0104	0.5966 ± 0.0177	0.9656 ± 0.0062	0.4348 ± 0.0168	-13.8653 ± 1.9707

Table 9: **Results for next-state prediction on** `dwd-wa1`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	**R2↑**	**RMSE↓**	**PCC↑**	**MAE↓**	**NSE↑**
ERM	0.7821 ± 0.0419	0.5097 ± 0.0369	0.8839 ± 0.0238	0.4224 ± 0.0424	0.7070 ± 0.0410
Joint Training	0.8510 ± 0.0232	0.4708 ± 0.0885	0.9223 ± 0.0126	0.3566 ± 0.1318	0.7416 ± 0.0905
Tent	0.7896 ± 0.0260	0.4989 ± 0.0553	0.8883 ± 0.0146	0.4144 ± 0.0658	0.7175 ± 0.0596
TTT	0.8509 ± 0.0234	0.4707 ± 0.0881	0.9223 ± 0.0127	0.3565 ± 0.1322	0.7417 ± 0.0902
TTT + Rotograd	0.8222 ± 0.0195	0.6235 ± 0.1246	0.9066 ± 0.0108	0.4983 ± 0.0093	0.5447 ± 0.1925
T3R (ours)	**0.9724** ± 0.0003	**0.1661** ± 0.0047	**0.9861** ± 0.0001	**0.1042** ± 0.0068	**0.9684** ± 0.0022

Table 10: **Results for next-state prediction on** `dwd-ky5`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	**R2↑**	**RMSE↓**	**PCC↑**	**MAE↓**	**NSE↑**
ERM	0.9698 ± 0.0038	0.6459 ± 0.0104	0.9842 ± 0.0022	0.4487 ± 0.0068	0.3321 ± 0.2544
Joint Training	**0.9798** ± 0.0058	0.4014 ± 0.1729	0.9896 ± 0.0030	0.2888 ± 0.1360	0.5926 ± 0.1483
Tent	0.9706 ± 0.0032	0.6282 ± 0.0245	0.9846 ± 0.0018	0.4354 ± 0.0176	0.3302 ± 0.2299
TTT	**0.9798** ± 0.0058	0.4116 ± 0.1869	**0.9897** ± 0.0030	0.2967 ± 0.1452	0.5923 ± 0.1707
TTT + Rotograd	0.9716 ± 0.0157	**0.3417** ± 0.1092	0.9852 ± 0.0085	**0.2679** ± 0.1038	**0.8086** ± 0.1464
T3R (ours)	0.9700 ± 0.0090	0.3783 ± 0.0705	0.9843 ± 0.0050	0.2867 ± 0.0570	0.7010 ± 0.1479

Table 11: **Results for next-state prediction on** `dwd-ky6`. A single adaptation step is applied for all methods except the first two (no-adaptation baselines). Values show mean ± standard deviation over three runs. Best-performing metrics are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	0.8300 ± 0.0211	0.5219 ± 0.0324	0.9109 ± 0.0116	0.4689 ± 0.0363	0.6922 ± 0.0383
Joint Training	0.7989 ± 0.1035	0.5441 ± 0.2618	0.8925 ± 0.0572	0.4340 ± 0.2570	0.6127 ± 0.3231
Tent	0.7677 ± 0.0512	0.5292 ± 0.0387	0.8757 ± 0.0295	0.4406 ± 0.0444	0.6832 ± 0.0459
TTT	0.7973 ± 0.1053	0.5449 ± 0.2622	0.8916 ± 0.0583	0.4334 ± 0.2570	0.6116 ± 0.3241
TTT + Rotograd	0.7406 ± 0.0602	0.8212 ± 0.1643	0.8600 ± 0.0354	0.6914 ± 0.2368	0.2184 ± 0.2873
T3R (ours)	**0.9760** ± 0.0089	**0.1699** ± 0.0469	**0.9879** ± 0.0045	**0.1270** ± 0.0514	**0.9656** ± 0.0195

Table 12: **Results for next-state prediction on** `dwd-ky13`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	0.9429 ± 0.0036	0.4529 ± 0.0511	0.9695 ± 0.0021	0.3679 ± 0.0394	-0.2282 ± 0.0399
Joint Training	0.9608 ± 0.0017	0.3436 ± 0.0504	0.9798 ± 0.0009	0.2430 ± 0.0356	-0.0033 ± 0.3232
Tent	0.9407 ± 0.0030	0.4723 ± 0.0243	0.9682 ± 0.0018	0.3715 ± 0.0287	-0.3173 ± 0.1090
TTT	**0.9608** ± 0.0018	0.3552 ± 0.0508	**0.9798** ± 0.0009	0.2521 ± 0.0361	0.0027 ± 0.3227
TTT + Rotograd	0.9234 ± 0.0629	**0.2326** ± 0.0273	0.9053 ± 0.1273	**0.1539** ± 0.0309	**0.6495** ± 0.0267
T3R (ours)	0.9461 ± 0.0312	0.3772 ± 0.1163	0.9722 ± 0.0161	0.2745 ± 0.0802	-0.3626 ± 1.7306

Table 13: **Results for next-state prediction on** `dwd-npcl1`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	0.9717 ± 0.0102	0.3901 ± 0.1600	0.9852 ± 0.0056	0.3268 ± 0.1645	0.1836 ± 0.6739
Joint Training	**0.9776** ± 0.0071	**0.3128** ± 0.0910	0.9885 ± 0.0038	**0.2302** ± 0.0878	-0.1863 ± 0.7491
Tent	0.9693 ± 0.0106	0.3912 ± 0.1224	0.9839 ± 0.0058	0.3185 ± 0.1315	-0.0647 ± 0.5163
TTT	0.9778 ± 0.0073	0.3148 ± 0.0988	**0.9886** ± 0.0039	0.2318 ± 0.0929	-0.1858 ± 0.7592
TTT + Rotograd	0.9748 ± 0.0042	0.4092 ± 0.1704	0.9872 ± 0.0022	0.3264 ± 0.1423	**0.5820** ± 0.3215
T3R (ours)	0.8747 ± 0.1670	0.3345 ± 0.1259	0.9194 ± 0.1140	0.2538 ± 0.0948	0.5359 ± 0.4260

Table 14: **Results for next-state prediction on** `dwd-ky7`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	**0.9778** ± 0.0120	0.3130 ± 0.0722	**0.9885** ± 0.0064	**0.2307** ± 0.0593	0.1304 ± 0.8040
Joint Training	0.9720 ± 0.0054	0.4917 ± 0.1514	0.9857 ± 0.0028	0.4076 ± 0.1192	-1.4581 ± 1.4893
Tent	0.9737 ± 0.0116	0.3254 ± 0.0897	0.9863 ± 0.0063	0.2382 ± 0.0682	-0.0816 ± 0.8438
TTT	0.9724 ± 0.0057	0.4888 ± 0.1465	0.9859 ± 0.0029	0.4061 ± 0.1176	-1.3802 ± 1.4800
TTT + Rotograd	0.9098 ± 0.0851	**0.2791** ± 0.0402	0.9366 ± 0.0718	0.2422 ± 0.0473	**0.3661** ± 0.4753
T3R (ours)	0.9755 ± 0.0063	0.3070 ± 0.2043	0.9874 ± 0.0034	0.2309 ± 0.1476	0.2824 ± 0.9126

Table 15: **Results for next-state prediction on** `dwd-ky4`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	0.8122 ± 0.0819	0.5186 ± 0.0150	0.9004 ± 0.0455	0.4024 ± 0.0694	0.7024 ± 0.0169
Joint Training	0.7834 ± 0.0342	0.6189 ± 0.2543	0.8849 ± 0.0194	0.4911 ± 0.2911	0.5281 ± 0.3830
Tent	0.8187 ± 0.0746	0.5705 ± 0.0769	0.9041 ± 0.0412	0.4653 ± 0.1311	0.6359 ± 0.0934
TTT	0.7850 ± 0.0360	0.6173 ± 0.2568	0.8857 ± 0.0204	0.4899 ± 0.2928	0.5294 ± 0.3853
TTT + Rotograd	0.7008 ± 0.0911	0.7632 ± 0.1331	0.8357 ± 0.0556	0.5849 ± 0.0089	0.3435 ± 0.2332
T3R (ours)	**0.9672** ± 0.0059	**0.1782** ± 0.0140	**0.9835** ± 0.0030	**0.1183** ± 0.0093	**0.9647** ± 0.0054

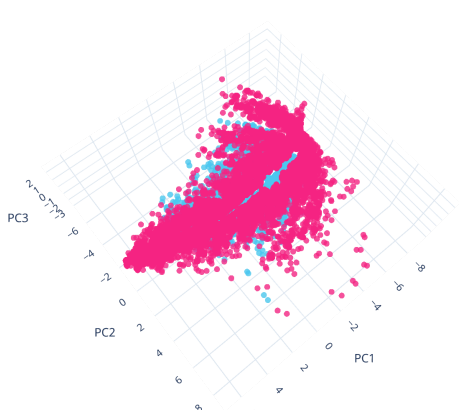
Table 16: **Results for next-state prediction on** `dwd-ky10`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

Method	R2↑	RMSE↓	PCC↑	MAE↓	NSE↑
ERM	**0.9297** ± 0.0030	0.5844 ± 0.0303	**0.9623** ± 0.0019	0.4284 ± 0.0198	-11.8800 ± 1.2133
Joint Training	0.6836 ± 0.0568	0.7063 ± 0.2614	0.8263 ± 0.0348	0.5417 ± 0.2413	0.4538 ± 0.4100
Tent	0.9279 ± 0.0013	0.6027 ± 0.0143	0.9611 ± 0.0008	0.4196 ± 0.0108	-12.9159 ± 0.6678
TTT	0.6837 ± 0.0568	0.7054 ± 0.2599	0.8264 ± 0.0347	0.5411 ± 0.2400	0.4556 ± 0.4070
TTT + Rotograd	0.7008 ± 0.0911	0.7632 ± 0.1331	0.8357 ± 0.0556	0.5849 ± 0.0089	0.3435 ± 0.2332
T3R (ours)	0.9258 ± 0.0099	**0.2778** ± 0.0136	0.9622 ± 0.0052	**0.1463** ± 0.0224	**0.9225** ± 0.0076

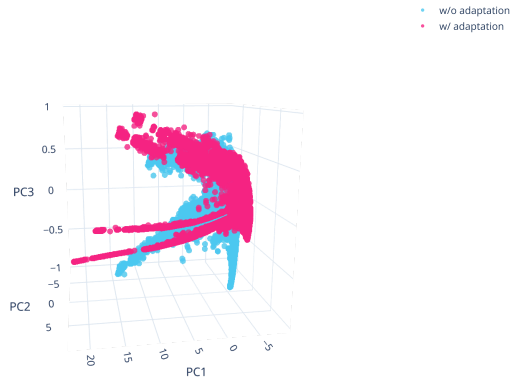
Table 17: **Results for next-state prediction on** `dwd-ctown`. Values show mean ± standard deviation over multiple runs. Best-performing metrics in each column are highlighted in bold.

A.7 Visualization

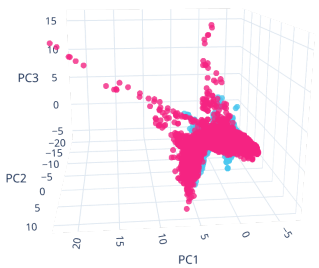
To observe the effects of test-time training, we visualize 3D PCA projections of embeddings at key positions, including the encoder's final layer and the second-to-last layers of both the main and SSL decoders, as shown in Figure 3. Surprisingly, although TTT does not employ a rotation matrix, its embedding space appears visually symmetrical. However, this symmetry does not expand further after adaptation, particularly in the encoder's final layer. In contrast, T3R amplifies the symmetric structure more intensively. This could be attributed to rotation matrices. In terms of SSL heads, the embedding space is unlikely to change under either approach, as the updates are still driven by the original SSL loss at test-time.



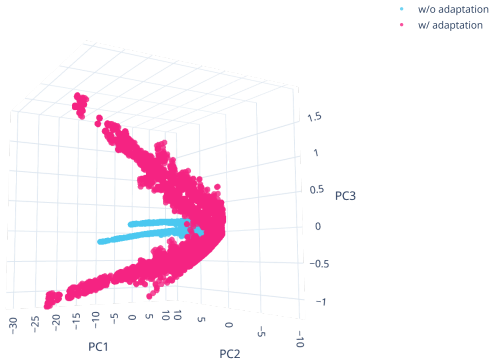
(a) TTT- Encoder's last layer emb.



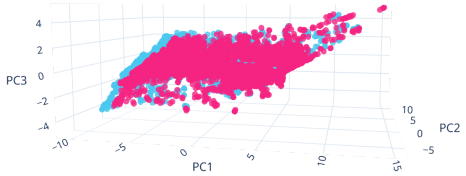
(b) T3R- Encoder's last layer emb.



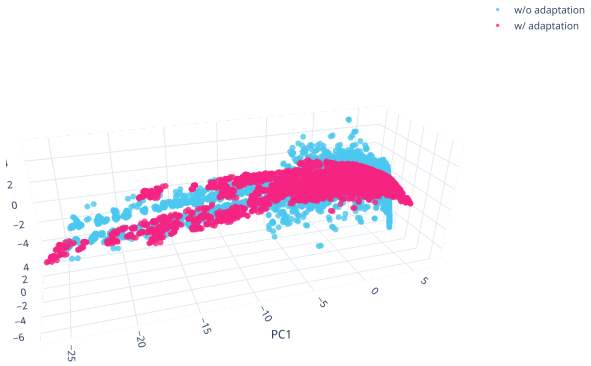
(c) TTT- Main decoder's penultimate emb.



(d) T3R- Main decoder's penultimate emb.



(e) TTT- SSL decoder's penultimate emb.



(f) T3R- SSL decoder's penultimate emb.

Figure 3: **3D PCA visualization**. Cyan showcases reduced features without adaptation, while pink highlights adapted ones. The viewpoint is adjusted for clearer visualization. The feature points appear approximately symmetric about a central axis. After adaptation, this symmetry becomes more apparent: the separation widens, and the tails grow sharper and more elongated. The exceptions are the SSL heads, whose shape remains nearly identical.