IN-Flow: Instance Normalization Flow for Non-stationary Time Series Forecasting

Wei Fan
weifan.oxford@gmail.com
University of Oxford
Oxford, UK

Shun Zheng
shun.zheng@microsoft.com
Microsoft Research Asia
Beijing, China

Pengyang Wang*
pywang@um.edu.mo
University of Macau
Macau SAR, China

Rui Xie
rui.xie@ucf.edu
University of Central Florida
Orlando, US

Kun Yi
kunyi.cn@gmail.com
State Information Center
Beijing, China

Qi Zhang
zhangqi_cs@tongji.edu.cn
Tongji University
Shanghai, China

Jiang Bian
jiang.bian@microsoft.com
Microsoft Research Asia
Beijing, China

Yanjie Fu
yanjie.fu@asu.edu
Arizona State University
Tempe, US

Abstract

Due to the non-stationarity of time series, the distribution shift problem largely hinders the performance of time series forecasting. Existing solutions either rely on using certain statistics to specify the shift, or developing specific mechanisms for certain network architectures. However, the former would fail for the unknown shift beyond simple statistics, while the latter has limited compatibility on different forecasting models. To overcome these problems, we first propose a *decoupled formulation* for time series forecasting, with no reliance on fixed statistics and no restriction on forecasting architectures. This formulation regards the removing-shift procedure as a special transformation between a raw distribution and a desired target distribution and separates it from the forecasting. Such a formulation is further formalized into a *bi-level optimization* problem, to enable the joint learning of the transformation (outer loop) and forecasting (inner loop). Moreover, the special requirements of expressiveness and bi-direction for the transformation motivate us to propose *instance normalization flow* (IN-Flow), a novel invertible network for time series transformation. Different from the classic "normalizing flow" models, IN-Flow does not aim for normalizing input to the prior distribution (e.g., Gaussian distribution) for generation, but creatively transforms time series distribution by stacking normalization layers and flow-based invertible networks, which is thus named "normalization" flow. Finally, we have conducted extensive experiments on both synthetic data and real-world data, which demonstrate the superiority of our method.

*Corresponding Author.

CCS Concepts

• **Computing methodologies → Neural networks**.

Keywords

Time Series Forecasting; Distribution Shift; Normalizing Flows

ACM Reference Format:
Wei Fan, Shun Zheng, Pengyang Wang, Rui Xie, Kun Yi, Qi Zhang, Jiang Bian, and Yanjie Fu. 2025. IN-Flow: Instance Normalization Flow for Non-stationary Time Series Forecasting. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3690624.3709260

1 Introduction

With wide applications in many real-world scenarios, such as electricity consumption planning [2, 14], traffic flow analysis [35], and weather condition estimation [1, 19], time series forecasting has received broad research attention for decades, with rapidly-evolving learning paradigm from traditional statistical approaches [20, 21, 51–53] to modern deep learning based methods [8, 9, 39, 45, 46, 55, 57, 58, 62], resulting in progressively yet significantly improved time series forecasting performance.

It is noteworthy that most of these methods, especially deep learning-based forecasting models, inherently follow a *stationarity* assumption: the temporal dependencies between historical observations and future predictions are stationary such that they can be effectively captured and generalized to future time points. However, this *stationarity* assumption overlooks the *non-stationarity* of real-world time series data, which can be characterized by continuously shifted joint distribution over time [24]. As well-known as the distribution shift problem, it can substantially hinder the performance of modern deep forecasting models [15, 24, 33] due to the resulting discrepancies of underlying distributions between the training and test data. Since time series data are usually collected at a high frequency over a long duration, such non-stationary sequences with millions of timesteps might even lead to a worse situation for forecasting [16, 54].

Nevertheless, reviewing all the achievements in non-stationary time series forecasting, existing studies always explicitly considering the distribution shift issue in time series forecasting either relied on certain statistics, such as rescaling time series with global minimum and maximum [38] and specifying non-stationary information as the mean, standard deviation, and learnable/sliced statistics [15, 24, 34], or focused on developing specific mechanisms for certain network architectures to relieve the shift [12, 33]. The former methods conduct normalizations towards data as preprocessing or in-processing steps, which would probably fail for unknown distribution shifts beyond a simple change of statistics. In contrast, the latter architecture-specific methods have limited compatibility and could be unadaptable for advanced forecasting architectures in the future.

To provide a common understanding of distribution shift and non-strationary forecasting and effectively address its interfering effects on forecasting models, in this paper, we develop a systematic approach following two crucial principles: (1) making no assumption that distribution shifts can be quantified by certain statistics; (2) making the method agnostic to the architectures of forecasting models. To achieve these ends, we develop a *decoupled formulation* for time series forecasting: regarding the procedure of removing distribution shift as a special distribution transformation, we separate the functionalities of removing shift and performing forecasting into a *transformation module* and a *forecasting module* respectively, where the first module is responsible for the transformation between the raw data distribution and the desired target distribution without shifts, and the second module holds the potential to obtain more accurate forecasting on "cleaned" data samples. Then, we naturally formalize such a decoupled formulation into a *bi-level optimization* problem [7, 18]: in order to enable the joint learning of the two modules, we iteratively improve the forecasting ability by minimizing an inner objective (training errors) and optimize the transformation ability by minimizing an outer object (validation errors) that corresponds to the generalization on the shifted data. This optimization procedure stimulates the transformation module to reduce shifts in a data-driven manner, leading to more stable and stationary time series forecasting.

Though the formulation easily accommodates any forecasting architectures, another critical challenge lies in the design of the transformation module. Some special requirements motivate us to introduce certain inductive biases. First, this module should transform distributions effectively (expressiveness) to relieve the shift. Second, the module is required to transform to and fro (bi-direction) between the raw distribution and the target distribution for the input/output of forecasting models. Such two considerations inspire us to leverage the key idea of *normalizing flows* [10, 11, 26], which can reversibly transform data distributions as one kind of *invertible neural networks* [3]. To this end, we propose *instance normalization flow* (IN-Flow), a novel invertible network for time series distribution transformation. In the domain of time series forecasting, the format of multi-variate sequences motivates us to integrate instance normalization [22, 24, 49] together with coupling layers [10, 11, 26]. Different from traditional normalizing flows targeting for the transformation to a prior simple distribution, IN-Flow targets for the transformation to a desired distribution with shifted information largely removed but forecasting-related information preserved, thus facilitating the final forecasting. In summary, our contributions can be summarized as follows:

- We present a *decoupled formulation* for time series forecasting that separates the functionalities of removing shifts and performing forecasting into a distribution transformation module and a forecasting module, which doesn't rely on fixed statistics, and has no restriction on forecasting architectures.
- We formalize such a formulation into a *bi-level optimization* problem to enable the joint learning of forecasting and the transformation in a data-driven manner.
- We propose a novel invertible neural network, IN-Flow, to fulfill the transformation for a desired distribution with shifted information removed but forecasting-related preserved.
- We conduct extensive experiments on both synthetic data and on real-world data, which demonstrate the consistent superiority of IN-Flow with regard to existing methods in non-stationary time series forecasting.

2 Related Work

2.1 Deep Learning for Time Series Forecasting

Time series forecasting has been a longstanding research topic. At an early stage, researchers have proposed statistical modeling approaches, including exponential smoothing [20] and auto-regressive moving averages (ARMA) [52]. With the great successes of deep learning, many deep learning models have been developed for time series forecasting. In recent years, deep learning-based methods have gained prominence in time series forecasting due to their ability to capture nonlinear and complex correlations [29]. These methods have employed various network architectures to learn temporal dependencies, such as recurrent neural network [28, 46], temporal convolution networks [4, 31], etc. One branch of methods has applied pure fully-connected neural networks for forecasting. For example, N-BEATS and DEPTS [17, 39] build residual layers on stacked connected layers for forecasting. N-HiTS [5] integrates multi-rate input sampling and hierarchical interpolation with MLPs to enhance univariate forecasting. DLinear [61] introduces a simple approach using a single-layer linear model to capture temporal relationships between input and output time series data. FreTS [59] utilizes frequency-domain MLP layers for time series forecasting; Filternet [56] adopts the frequency filters with the linear layers for the forecasting. Another branch of methods are built upon Transformer [50] for the time series forecasting task. Many research have improved vanilla transformer in attention computation, memory consumption, etc, to enhance forecasting [55, 62, 63]. To capture intricate dependencies and long-range interactions. PatchTST [37] segments time series into patches as input tokens to the Transformer and maintaining channel independence. iTransformer [32] inverts the Transformer's structure by treating independent series as variate tokens to capture multivariate correlations.

3 Preliminary

3.1 Non-stationary Time Series Forecasting

Time series forecasting suffers from the non-stationarity and the distribution shift issue considering distributions of real-world series change over time [2]. To overcome this problem, some specific mechanisms have been designed for certain network architectures to relieve the distribution shift: for example, [12] proposed Adaptive RNNs to handle the shift problem on top of recurrent neural networks (RNNs); Nonstationary Transformers [33] have proposed series stationarization and de-stationary attention to enhance transformer-based forecasting models. These methods, however, are network-specific architectures that have limited compatibility with future advanced forecasting methods. Another kind of method for the non-stationarity problem is to utilize the normalization techniques. Usually, normalization is model-agnostic and can be coupled with different backbones. For example, the pioneer work, Adaptive Norm [38] rescale time series through global minimum and maximum. DAIN [42] use adaptive z-score normalization by global average and std. for the task of time series classification. Recently, RevIN [24] proposes to use instance normalization to reduce time series distribution shift for deep learning time series forecasting methods. Dish-TS [15] considers both the inter-space shift and intra-space shift and proposes coefficient networks to capture learnable statistics for normalization. SAN [34] considers the sliced statistics of time series and proposes the corresponding adaptive sliced normalization. However, these normalization techniques rely on statistics calculation, which fall short in overcoming unknown shifts beyond the statistics.

3.2 Time Series Forecasting

Let $\boldsymbol{s} = [\boldsymbol{s}_1; \boldsymbol{s}_2; \cdots; \boldsymbol{s}_T] \in \mathbb{R}^{T\times D}$ be regularly sampled multi-variate time series with T timestamps and D variates, where $\boldsymbol{s}_t \in \mathbb{R}^D$ denotes the multi-variate values at timestamp t. Besides, we use $\boldsymbol{x}_t^L \in \mathbb{R}^{L\times D}$ to denote a length-L segment of $\boldsymbol{s}$ ending at timestamp t (exclusive), namely $\boldsymbol{x}_t^L = \boldsymbol{s}_{t-L:t} = [\boldsymbol{s}_{t-L}; \boldsymbol{s}_{t-L+1}; \cdots; \boldsymbol{s}_{t-1}]$. Similarly, we represent a length-H segment of $\boldsymbol{s}$ starting from timestamp t (inclusive) as $\boldsymbol{y}_t^H$, so we have $\boldsymbol{y}_t^H = \boldsymbol{s}_{t:t+H} = [\boldsymbol{s}_t; \boldsymbol{s}_{t+1}; \cdots; \boldsymbol{s}_{t+H-1}]$. The classic time series forecasting formulation is to project historical observations $\boldsymbol{x}_t^L$ into their subsequent future values $\boldsymbol{y}_t^H$. Specifically, a typical forecasting model $f_\theta : \mathbb{R}^{L\times D} \to \mathbb{R}^{H\times D}$ produces forecasts by $\hat{\boldsymbol{y}}_t^H = f_\theta(\boldsymbol{x}_t^H)$ where $\hat{\boldsymbol{y}}_t^H$ stands for the forecasting result, θ encapsulates the model parameters, and H and L are usually referred to as the lengths of horizon (lookahead) windows and lookback windows, respectively.

3.3 Bi-level Optimization

Bi-level optimization problems are one kind of optimization problems in which a set of variables in an objective function are constrained by the optimal solution of another optimization problem [7]. Specifically, given two functions J_{in} and J_{out} as the inner objectives and outer objectives, we consider two set of variables $\theta \in \mathbb{R}^m, \phi \in \mathbb{R}^n$ as the inner variables and outer variables, respectively; the bi-level problem can be given by $\min\limits_{\phi,\theta_\phi} J_{out}(\theta_\phi, \phi)$ such that $\theta_\phi \in \arg\min\limits_{\theta} J_{in}(\theta, \phi)$. Bi-level problems have involved in many applications, including hyperparmeter optimization, multi-task, and meta-learning [7, 18, 44, 48].

3.4 Normalzing Flows

Normalzing flows [40], as one common kind of invertible neural networks, can conduct transformation between distributions. Given a high-dimensional random variable $\mathbf{x} \in \mathcal{X}$, flow-based models [10, 11, 26, 41] transform densities $p_\mathcal{X}$ into certain target distribution $p_\mathcal{Z}$ (e.g., an isotropic Gaussian) on a latent variable $\mathbf{z} \in \mathcal{Z}$. The transformation is a bijection $g : \mathcal{X} \to \mathcal{Z}$ (with inverse g^{-1}). The probability density can be written by the change of variable formula as: $p_\mathcal{X}(\mathbf{x}) = p_\mathcal{Z}(\mathbf{z}) \left|\det\left(\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}\right)\right|$ where $\partial g(\mathbf{x})/\partial \mathbf{x}$ is the Jacobian of g at $\mathbf{x}$. The transformation g is composed of a sequence of differentiable invertible (or bijective) mapping functions $g = g_1 \circ g_2 \circ \cdots \circ g_P$ that map $\mathbf{x} \overset{g_1}{\longleftrightarrow} \mathbf{h}_1 \overset{g_2}{\longleftrightarrow} \mathbf{h}_2 \cdots \overset{g_P}{\longleftrightarrow} \mathbf{z}$ reversibly. Usually, flow-based models are trained via maximum likelihood on training data. Thus, the probability density function of the flows given a data point is $\log p_\mathcal{X}(\mathbf{x}) = \log p_\mathcal{Z}(\mathbf{z}) + \log\left|\det\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right)\right| = \log p_\mathcal{Z}(\mathbf{z}) + \sum_{i=1}^P \log|\det(\partial \mathbf{h}_i/\partial \mathbf{h}_{i-1})|$, where $\mathbf{h}_0 \triangleq \mathbf{x}$ and $\mathbf{h}_P \triangleq \mathbf{z}$; $\log|\det(\partial \mathbf{h}_i/\partial \mathbf{h}_{i-1})|$ is called log-determinant of $(\partial \mathbf{h}_i/\partial \mathbf{h}_{i-1})$. The key idea of designing transformation in normalizing flows is to make the computation of log-determinant simple and easy [27, 40].

4 Methodology

In this section, we elaborate on our principled approach that makes no assumption to specify distribution shifts and accommodates all kinds of forecasting models. First, we introduce a decoupled formulation in Section 4.1; then, we formalize a bi-level optimization problem for the decoupled formulation in Section 4.2, and present our specifically designed IN-Flow model in Section 4.3. Besides, Figure 1 demonstrates an overview of our framework.

4.1 The Novel Decoupled Formulation

It is noteworthy that the aforementioned formulation ($\hat{\boldsymbol{y}}_t^H = f_\theta(\boldsymbol{x}_t^H)$), embracing a large body of forecasting models [36, 39, 55, 60, 62], essentially follows a *stationary* assumption: *the conditional data probability* $P(\boldsymbol{y}_t^H|\boldsymbol{x}_t^L)$ *does not change over time.* However, in practice, some recent studies [12, 24, 33] have well recognized the wide existence of non-stationary properties in real-world time series. Besides, we further note that those distribution shift patterns can be far more sophisticated than simple statistical measures, such as mean and standard deviation used in [24]. Thus arbitrarily specifying non-stationary properties has the risk of removing certain information crucial to forecasting and is limited in tackling more diverse distribution shifts.

To provide a generic approach to address the problem of distribution shift in time-series forecasting, we regard the procedure of removing non-stationary information as a special distribution transformation (*transformation module*) and separate its functionality from the *forecasting module*. Accordingly, we develop a decoupled formulation as:

$$p(\boldsymbol{y}_t^H|\boldsymbol{x}_t^L) = p(\boldsymbol{y}_t^H|\tilde{\boldsymbol{y}}_t^H) \cdot p(\tilde{\boldsymbol{y}}_t^H|\tilde{\boldsymbol{x}}_t^L) \cdot p(\tilde{\boldsymbol{x}}_t^L|\boldsymbol{x}_t^L) \tag{1}$$

where $\tilde{\boldsymbol{x}}_t^L$ and $\tilde{\boldsymbol{y}}_t^H$ denote the transformed variables of $\boldsymbol{x}_t^L$ and $\boldsymbol{y}_t^H$, respectively. We assume that $\tilde{\boldsymbol{x}}_t^L$ and $\tilde{\boldsymbol{y}}_t^H$ come from certain desired distributions with shifted information removed but forecasting-related information preserved. In this way, we decouple the modeling of conditional data distribution $p(\boldsymbol{y}_t^H|\boldsymbol{x}_t^L)$ into three parts: (i) $p(\tilde{\boldsymbol{x}}_t^L|\boldsymbol{x}_t^L)$, performing a transformation on raw input series to

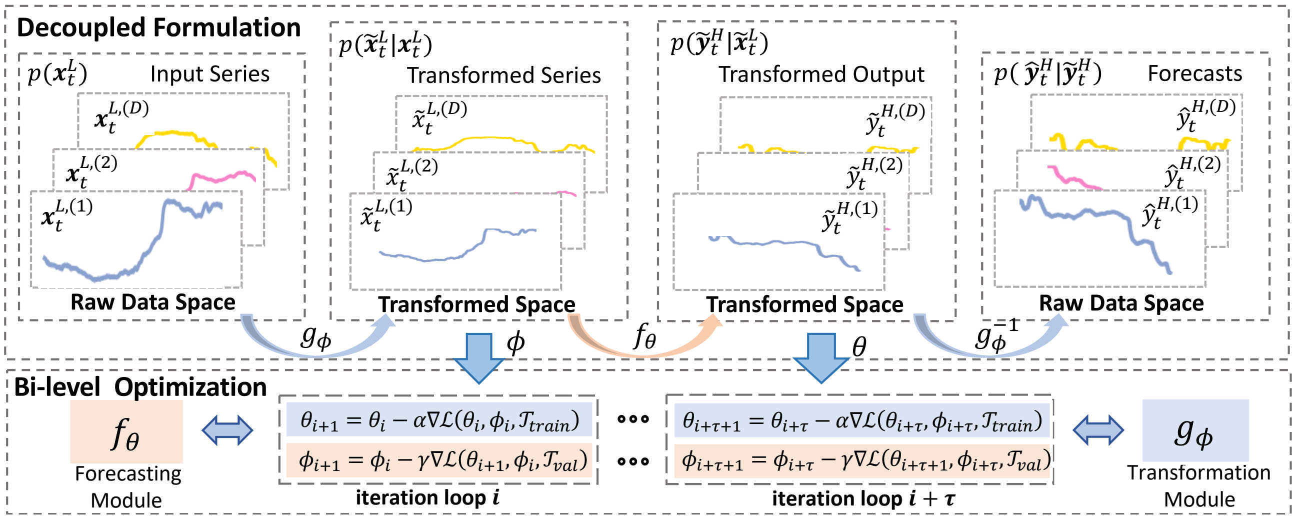

Decoupled Formulation
$p(\boldsymbol{x}_t^L)$ Input Series
$p(\tilde{\boldsymbol{x}}_t^L|\boldsymbol{x}_t^L)$ Transformed Series
$p(\tilde{\boldsymbol{y}}_t^H|\tilde{\boldsymbol{x}}_t^L)$ Transformed Output
$p(\hat{\boldsymbol{y}}_t^H|\tilde{\boldsymbol{y}}_t^H)$ Forecasts
Raw Data Space
Transformed Space
Transformed Space
Raw Data Space
g_ϕ
ϕ
f_θ
θ
g_ϕ^{-1}
Bi-level Optimization
f_θ
Forecasting Module
$\theta_{i+1} = \theta_i - \alpha\nabla\mathcal{L}(\theta_i, \phi_i, \mathcal{T}_{train})$
$\phi_{i+1} = \phi_i - \gamma\nabla\mathcal{L}(\theta_{i+1}, \phi_i, \mathcal{T}_{val})$
iteration loop i
$\theta_{i+\tau+1} = \theta_{i+\tau} - \alpha\nabla\mathcal{L}(\theta_{i+\tau}, \phi_{i+\tau}, \mathcal{T}_{train})$
$\phi_{i+\tau+1} = \phi_{i+\tau} - \gamma\nabla\mathcal{L}(\theta_{i+\tau+1}, \phi_{i+\tau}, \mathcal{T}_{val})$
iteration loop $i + \tau$
g_ϕ
Transformation Module

Figure 1: Framework overview. The upper part is the proposed decoupled formulation (Section 4.1): it separates the time series forecasting into the transformation module (g_ϕ) and the forecasting module (f_θ); the input series with the raw data space is converted into the transformed space by g_ϕ for the forecasting conducted by the forecasting module f_θ; then the predicted results will be recovered by the inverse of g_ϕ to recovered to the raw data space. The lower part is the bi-level optimization (Section 4.2) for the decoupled formulation of time series forecasting.

remove non-stationary (shifted) information; (ii) $p(\tilde{\boldsymbol{y}}_t^H|\tilde{\boldsymbol{x}}_t^L)$, conducting forecasting on the transformed space; (iii) $p(\boldsymbol{y}_t^H|\tilde{\boldsymbol{y}}_t^H)$, performing a reverse transformation to recover certain non-stationary patterns and produce final forecasts.

This decoupled formulation lays the foundation for respective parameterization and optimization of transformation and forecasting modules. Then in addition to the forecasting function f_θ, we introduce another two functions to be responsible for the bi-directional distribution transformations in the transformation module. To be specific, $g_{\phi^L} : \boldsymbol{x}_t^L \to \tilde{\boldsymbol{x}}_t^L$, parameterized by ϕ^L, stands for the transformation from a raw data distribution $\mathcal{X}^L$ to a desired distribution $\tilde{\mathcal{X}}^L$ without non-stationary information, and $g_{\phi^H}^{-1} : \boldsymbol{x}_t^H \to \tilde{\boldsymbol{x}}_t^H$ parameterized by ϕ^H denotes the reverse transformation from $\tilde{\mathcal{X}}^H$ to $\mathcal{X}^H$. Accordingly, we emit a forecast $\hat{\boldsymbol{y}}_t^H$ given an input $\boldsymbol{x}_t^L$ as:

$$\hat{\boldsymbol{y}}_t^H = g_{\phi^H}^{-1}\left(f_\theta\left(g_{\phi^L}\left(\boldsymbol{x}_t^L\right)\right)\right) \tag{2}$$

4.2 Bi-level Optimization for the Decoupled Transformation and Forecasting

Given the decoupled formulation, we further consider a learning procedure to fulfill the needs of 1) removing distribution shifts that hinder the forecasting performance and 2) performing accurate forecasting based on samples from a transformed stationary distribution. Motivated by these two-sided objectives, we naturally develop a *bi-level optimization* problem [7]:

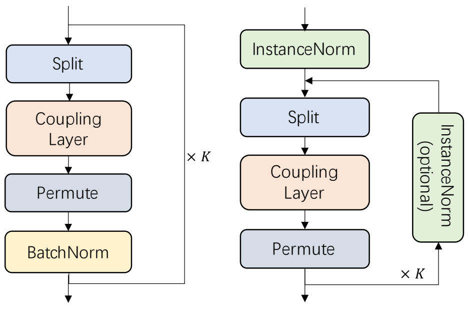
$$\begin{aligned} \phi_* &= \arg\min_\phi \mathcal{L}\left(\theta_*(\phi), \phi, \mathcal{T}_{val}\right) \\ \text{s.t.} \quad \theta_*(\phi) &= \arg\min_\theta \mathcal{L}\left(\theta, \phi, \mathcal{T}_{train}\right) \end{aligned} \tag{3}$$

where ϕ encapsulates ϕ^H and ϕ^L, $\mathcal{T}_{train}$ and $\mathcal{T}_{val}$ denote the sets of time steps in training and validation data, respectively, and $\mathcal{L}(\theta, \phi, \mathcal{T}) = \sum_{t\in\mathcal{T}} \ell(\boldsymbol{y}_t^H, g_{\phi^H}^{-1}(f_\theta(g_{\phi^L}(\boldsymbol{x}_t^L))))$ is the summation of losses, specified by $\ell(\boldsymbol{y}_t^H, \hat{\boldsymbol{y}}_t^H)$, over all time steps in $\mathcal{T}$. The inner loop optimization indicates that given any transformation model parameterized by ϕ, the solution $\theta_*(\phi)$ minimizes the forecasting losses on the training data, which ensures the effective learning of the forecasting module on the transformed space. The outer loop optimization aims to identify the transformation solution ϕ_* that contributes to the best generalization performance on the validation data, which aligns with our motivation to remove uncertain shifted information hindering forecasting.

Moreover, a standard approach to solve the bi-level optimization problem of Equation (3) is to introduce hyper gradients [6]. Nevertheless, the calculation of hyper gradients involves the computation of $\frac{\partial\mathcal{L}(\theta_*(\phi),\phi,\mathcal{T}_{val})}{\partial\theta_*(\phi)}\frac{\partial\theta_*(\phi)}{\partial\phi}^T$, which can be prohibitive when performing exact inner optimization. Therefore, we adopt the first-order approximation, as did in [30], to cut off the gradient dependency introduced by $\frac{\partial\theta_*(\phi)}{\partial\phi}$ and alternate between the following two gradient updates:

$$\begin{aligned} \theta_{i+1} &= \theta_i - \alpha\frac{\partial\mathcal{L}(\theta_i, \phi_i, \mathcal{T}_{train})}{\partial\theta_i}, \\ \phi_{i+1} &= \phi_i - \gamma\frac{\partial\mathcal{L}(\theta_{i+1}, \phi_i, \mathcal{T}_{val})}{\partial\phi_i}. \end{aligned} \tag{4}$$

where i means the i-th iteration; α and γ are the learning rate for the forecasting module and the transformation module respectively. In this way, we actually turn the original bi-level optimization into alternate optimization [6]. While in practice, we find that applying such an approximation to our case brings very fast computation with negligible performance compromise.



(a) RealNVP (b) IN-Flow

Figure 2: The specific architecture of RealNVP [11] and our IN-Flow. We discuss the difference of pre- and post-norm in Section 5.3.1 and take the pre-norm version as IN-Flow.

4.3 Instance Normalizaiton Flow

Though the decoupled formulation with bi-level optimization can accommodate any f_θ, another critical challenge is the design of the transformation module and optimization for ϕ. First, the module should have adequate ability to transform data distributions effectively (expressiveness). Second, the module is responsible for the bi-directional transformation between $\mathcal{X}$ $(\mathcal{X}^L, \mathcal{X}^H)$ and $\tilde{\mathcal{X}}$ $(\tilde{\mathcal{X}}^L, \tilde{\mathcal{X}}^H)$. Such considerations inspire us to leverage the key idea of *normalizing flows* [27], one kind of invertible networks [3] that transform data distributions.

Existing normalizing flows conduct reversible transformation on one raw data distribution (see Section 3.4). In contrast, we notice our formulation is composed of two transformations, $g_{\phi^L} : \boldsymbol{x}_t^L \longrightarrow \tilde{\boldsymbol{x}}_t^L$ and $g_{\phi^H}^{-1} : \tilde{\boldsymbol{x}}_t^H \longrightarrow \boldsymbol{x}_t^H$, which actually operate in two variable spaces (g_{ϕ^L} on $\mathcal{X}^L \in \mathbb{R}^{L\times D}$ and g_{ϕ^H} on $\mathcal{X}^H \in \mathbb{R}^{H\times D}$), due to the difference of lookback length (L) and horizon length (H). Nonetheless, these two distributions, $p_{\mathcal{X}^L}(\boldsymbol{x}^L)$ and $p_{\mathcal{X}^H}(\boldsymbol{x}^H)$, undoubtedly share many common parts in distribution because their samples are generated by segmenting the same time series ($\boldsymbol{s}$). Therefore, to fulfill bi-directional transformations on two variable spaces and capture the common patterns between them, we develop a novel invertible neural architecture, namely *instance normalization flow* (IN-Flow), that instantiates g_{ϕ^L} and g_{ϕ^H} as a single network and works on variable length of time series, also referred to as g_ϕ for brevity.

Flow-based models [10, 11, 26, 41] usually adopt batch normalization in their architectures for a stable training. However, a batch in time series forecasting may consist of samples of various distributions due to distribution shift [24]. In such cases, batch normalization could meet with instability in calculating batch statistics and influence the transformation [47]. Moreover, we notice instance normalization has recently been effective in computer vision [22, 49] and can be an alternative to get over the instability [13]. And in the domain of time series, the sequence format can make distribution shift relieved with the application of instance normalization [24]. Inspired by them, our IN-Flow includes parametric instance normalization layer which is written given d-th variate by:

$$\boldsymbol{h'}_t^{L,(d)} = \left(\boldsymbol{h}_t^{L,(d)} - \mu_t^{(d)}\right)\left(\sigma_t^{2(d)} + \epsilon\right)^{-\frac{1}{2}} \exp(\gamma_t^{(d)}) + \beta_t^{(d)} \tag{5}$$

where $\boldsymbol{h'}_t^{L,(d)}$ are the corresponding mappings of $\boldsymbol{h}_t^{L,(d)} \in \mathbb{R}^{L\times 1}$.

For brevity, supposing the current layer is the first layer of g_ϕ, we have $\boldsymbol{h}_t^{L,(d)} \triangleq \boldsymbol{x}_t^{L,(d)}$, we have $\boldsymbol{h}_t^{L,(d)} \triangleq \boldsymbol{x}_t^{L,(d)}$, $\mu_t^{(d)} = \frac{1}{L}\sum \boldsymbol{x}_t^{L,(d)} = \frac{1}{L}\sum_{t-L}^{t-1} s_t^{(d)}$ as the instance mean, and we have $\sigma_t^{2(d)} = \frac{1}{L}\sum_{t-L}^{t-1}\left(s_t^{(d)} - \mu_t^{(d)}\right)^2$ as square of instance standard deviation, where $s_t^{(d)}$ as the value of d-th variate at timestamp t. And $\gamma_t^{(d)}$, $\beta_t^{(d)}$ are affine parameters for transformation in normalization layers. If it's in the middle layer of IN-Flow, we have $\mu_t^{(d)} = \frac{1}{L}\sum \boldsymbol{h}_t^{L,(d)}$ and $\sigma_t^{2(d)} = \frac{1}{L}\sum_{t-L}^{t-1}\left(h_t^{L,(d)} - \mu_t^{(d)}\right)^2$. While Equation (5) is the forward transformation of $p(\tilde{\boldsymbol{x}}_t^L | \boldsymbol{x}_t^L)$, since it is an invertible network, we can also write the inverse transformation of the above operation corresponding for $p(\boldsymbol{y}_t^H | \tilde{\boldsymbol{y}}_t^H)$ in IN-Flow. Specifically, in the inverse transformation, the instance normalization given the d-th variate is by:

$$\boldsymbol{h}_t^{H,(d)} = \left(\boldsymbol{h'}_t^{H,(d)} - \beta_t^{(d)}\right)\exp(-\gamma_t^{(d)})\left(\sigma_t^{2(d)} + \epsilon\right)^{\frac{1}{2}} + \mu_t^{(d)} \tag{6}$$

where $\boldsymbol{h}_t^{H,(d)}$ are the corresponding forecasts of $\boldsymbol{h}_t^{L,(d)}$ in the transformed space; $\boldsymbol{h}_t^{H,(d)}$ are inverse transformed results of current instance normalization layer; other parameters can be corresponding for those in Equation (5).

Moreover, to make the transformation more expressive, we follow previous models to include one common reversible design of flows, *affine coupling layers* [10, 11] as part of IN-Flow. In implementation, we let the coupling layers work on the feature (variate) dimension by:

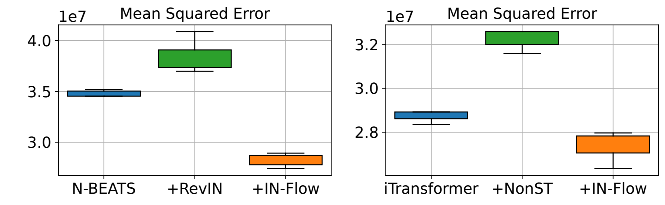
$$\begin{cases} \boldsymbol{h'}_t^{L,(1:d_c)} = \boldsymbol{h}_t^{L,(1:d_c)} \\ \boldsymbol{h'}_t^{L,(d_c:D)} = \boldsymbol{h}_t^{L,(d_c:D)} \odot s\left(\boldsymbol{h}_t^{L,(1:d_c)}\right) + t\left(\boldsymbol{h}_t^{L,(1:d_c)}\right) \end{cases} \tag{7}$$

where usually we let $d_c = \lceil \frac{D}{2} \rceil$ stand for the split position; $\boldsymbol{h}_t^{L,(1:d_c)} \in \mathbb{R}^{L\times d_c}$ denotes data composed of first d_c variates of $\boldsymbol{h}_t^L$ and $\boldsymbol{h}_t^{L,(d_c:D)} \in \mathbb{R}^{L\times(D-d_c)}$ denotes the left $D - d_c$ variates; s and t are scale and translation functions mapping from $\mathbb{R}^{d_c}$ to $\mathbb{R}^{D-d_c}$; $\boldsymbol{h'}_t^L$ is the processed output. Accordingly, the inverse transformation of the above coupling layers for $p(\boldsymbol{y}_t^H | \tilde{\boldsymbol{y}}_t^H)$ in IN-Flow can be written by:

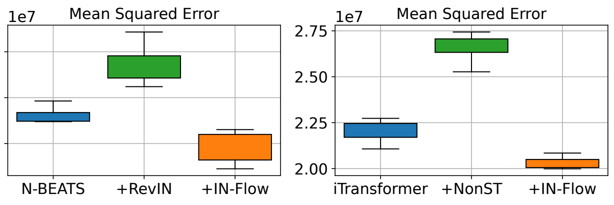
$$\begin{cases} \boldsymbol{h}_t^{H,(1:d_c)} = \boldsymbol{h'}_t^{H,(1:d_c)} \\ \boldsymbol{h}_t^{H,(d_c:D)} = \left(\boldsymbol{h'}_t^{H,(d_c:D)} - t\left(\boldsymbol{h'}_t^{H,(1:d_c)}\right)\right) / s\left(\boldsymbol{h'}_t^{H,(1:d_c)}\right) \end{cases} \tag{8}$$

where $\boldsymbol{h'}_t^H$ are the forecasting results of $\boldsymbol{h'}_t^L$ in the hidden space; $\boldsymbol{h}_t^H$ is the processed results of $\boldsymbol{h'}_t^H$ by the affine coupling layers. Especially, if the coupling layers are the final layers of IN-Flow, we can have $\hat{\boldsymbol{y}}_t^H = \boldsymbol{h}_t^H$. In stacking IN-Flow layers, we find that the pre-norm (IN-Flow) can achieve better performance than post-norm (IN-Flow-T in Section 5.3.1). Thus we take the pre-norm variant as the final design of IN-Flow.

Overall, the main architecture of IN-Flow is by stacking *instance normalization* and *coupling layers* iteratively, similar to other classic normalizing flows such as RealNVP [11], as shown in Figure 2. We also adopt *Split* and *Permute* operations [10, 11, 26] to assist for stacking layers. Note that both Equation (5) and Equation (7) belong to the transformation of $p(\tilde{\boldsymbol{x}}_t^L | \boldsymbol{x}_t^L)$. We can also accordingly write the *inverse* transformation of above layers for $p(\boldsymbol{y}_t^H | \tilde{\boldsymbol{y}}_t^H)$ in IN-Flow.



(a) Comparison of forecasting on synthetic data-1



(b) Comparison of forecasting on synthetic data-2

Figure 3: Evaluation of forecasting on the synthetic data. NonST means non-stationary transformer.

5 Experiments

Our empirical studies in this section aim to answer the following questions: 1) With some solutions existing, why should we further study distribution shifts and non-stationary time series forecasting? 2) How much benefit can our framework gain for time series forecasting compared with existing state-of-the-art models in real-world settings? 3) Does each part of IN-Flow designs (including optimization) count for good forecasting performance?

Baselines. We mainly consider several kinds of state-of-the-art methods for non-stationary time series forecasting. For the model-agnostic normalization technique, we adopt RevIN [24], Dish-TS [15], and SAN [34]. For the backbones, we adopt the recently well-performed forecasting methods, i.e., iTransformer [32], PatchTST [37], Autoformer [55], and N-BEATS [39]. For the model-specific mechanisms, we adopt Non-tationary Transformer [33] and we take vanilla Transformer [50], Informer [62] and Autoformer [55] as the backbones More baseline details are in Appendix A.

Evaluation Details. We evaluate time series forecasting performance on the classic mean squared error (MSE) and mean absolute error (MAE). To directly reflect distribution shifts in time series, all the evaluations are conducted on original data without data scaling or normalization following settings of previous works [15]. The reported metrics on real-world data are scaled for readability. More evaluation details are included in Appendix C.2.

5.1 Evaluation on Synthetic Data

To intuitively illustrate the importance of modeling distribution shifts in forecasting and the superiority of IN-Flow, we generate synthetic time series data with distributions shifted. We first group each neighbored τ points as a segment and let every segment follows one distribution. Supposing to synthesize one series $\{s_1, \cdots, s_T\}$ of length T, this series can be regarded to have $U = \lceil T/\tau \rceil$ segments, where the u-th segment follows its distribution $\mathcal{P}_u$. We can thus control the degree of time series distribution shift through adjusting τ and $\mathcal{P}_u$. Under such settings, both $p(\boldsymbol{x}_t^L)$ and $p(\boldsymbol{y}_t^H|\boldsymbol{x}_t^L)$ keep shifted over time if lookback length $L > \tau$ and lookahead length $H > \tau$. Then for each $\mathcal{P}_u$, we simply make a regular cosine wave signal to model simple periodic patterns in time series. Specifically, we let $s_t = A_u \cos(2\pi \frac{1}{T_u} t + B_u) + C_u$ if time index t belongs to u-th segment, where A_u, T_u, B_u, C_u indicate amplitude, period, phase and level parameters for the u-th segment.

Following the above rules, we make three synthetic datasets, called Synthetic-1/2 by varying different τ and $\mathcal{P}_u$. Due to space limit, we include more details about synthetic datasets in Appendix B. Figure 3 shows two different comparisons of different backbones on Synthetic-1 and Synthetic-2 datasets. We notice that in our simulation settings, both two state-of-the-art techniques RevIN and non-stationary Transformers would fail significantly, causing a more than 10% performance decrease on iTransformer and N-BEATS respectively. These results demonstrate their limitations in overcoming distribution shifts in forecasting when meeting some severely shifted time series, while IN-Flow can still improve the forecasting ability of backbones by a large margin which shows our superiority in non-stationary forecasting.

5.2 Evaluation on Real-world Data

Apart from simulation experiments, we further perform experiments on real-world datasets. We adopt six existing shifted datasets following previous forecasting literature: Electricity transformer temperature datasets include time series of oil temperature and power load collected from electricity transformers in China. ***ETTm2*** dataset is recorded at a fifteen-minute frequency ranging from July 2016 to July 2018. In contrast, ***ETTh1*** dataset dataset is recorded every hour. ***Electricity*** dataset contains the hourly electricity consumption of 321 customers from 2012 to 2014. ***Weather*** dataset contains meteorological measurements of 21 weather indicators, which are collected 10-minutely in 2020. ***CAISO*** dataset includes hourly electricity load series in different zones of California ranging from 2017 to 2020. ***NordPool*** dataset includes hourly energy production volume series of several European countries in 2020.

For a stable evaluation, we perform z-score normalization for each backbone model and inverse the forecasting results to report metrics. We rerun all the models under four different seeds to report the average MSE/MAE on the testset of each dataset. For data split, we follow [62] and split data into train/validation/test set by the ratio 6:2:2 towards *ETTh1* dataset *ETTm2* dataset. We also adopt the ratio 6:2:2 to split the data for *CAISO* dataset and *NordPool* dataset. For *Weather* dataset and Electricity dataset, we follow [55] to split data by the ratio of 7:1:2 for the train/validation/test set. To cover short/long-term forecasting, we set the length of the lookback and horizon window from 48 to 336. We train the forecasting models using L2 loss and Adam [25] on a single NVIDIA A100 40GB GPU.

5.2.1 *Main Results.* Table 1 demonstrates the overall performance comparison of the basic models and their IN-Flow-equipped versions on four representative time series forecasting models. Based on the results, we can observe that IN-Flow consistently improves backbone models under different settings across the datasets. The average improvements of IN-Flow on the backbone models come to 28.3% on PatchTST, 21.3% on iTransformer, 28.5 on Autoformer, and 12.1% on N-BEATS across different datasets, which can show the

Table 1: Performance comparison of time series forecasting on backbone models and IN-Flow. The length of look-back/horizon windows is prolonged from 48 to 336 to cover short and long forecasting settings.

Method		PatchTST		+IN-Flow		iTransformer		+IN-Flow		Autoformer		+IN-Flow		N-BEATS		+IN-Flow	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	48	1.128	1.865	**0.936**	**1.635**	1.051	1.717	**0.811**	**1.244**	1.237	2.010	**1.215**	**1.937**	0.959	1.637	**0.951**	**1.624**
	96	1.320	2.109	**1.025**	**1.782**	1.188	1.933	**0.896**	**1.542**	1.710	2.347	**1.428**	**2.141**	1.072	1.789	**1.027**	**1.768**
	168	2.001	2.679	**1.077**	**1.888**	1.917	2.624	**0.901**	**1.925**	2.139	2.739	**1.303**	**2.095**	1.223	1.989	**1.139**	**1.914**
	336	1.784	2.613	**1.121**	**2.021**	1.734	2.568	**1.107**	**1.189**	2.444	2.965	**1.502**	**2.305**	1.222	2.083	**1.146**	**2.044**
ETTm2	48	1.116	2.204	**0.925**	**1.908**	1.094	2.168	**0.958**	**1.972**	1.477	2.469	**1.459**	**2.432**	1.329	2.308	**1.229**	**2.182**
	96	1.503	2.699	**0.992**	**1.965**	1.470	2.178	**0.996**	**1.973**	1.962	2.901	**1.869**	**2.748**	1.628	2.604	**1.570**	**2.496**
	168	1.708	2.783	**1.213**	**2.202**	1.352	2.472	**1.248**	**2.275**	1.898	2.919	**1.834**	**2.831**	2.148	3.021	**1.752**	**2.681**
	336	2.664	3.465	**1.564**	**2.482**	2.124	2.713	**1.678**	**2.532**	2.271	3.235	**2.113**	**3.003**	2.376	3.339	**1.928**	**2.847**
Electricity	48	0.632	0.366	**0.554**	**0.342**	0.513	0.319	**0.421**	**0.312**	0.686	0.398	**0.592**	**0.381**	0.698	0.431	**0.594**	**0.422**
	96	0.882	0.698	**0.589**	**0.341**	0.710	0.558	**0.479**	**0.313**	0.936	0.763	**0.669**	**0.397**	0.912	0.751	**0.872**	**0.723**
	168	1.678	0.541	**0.882**	**0.389**	1.341	0.413	**0.790**	**0.329**	1.989	0.576	**0.943**	**0.437**	2.031	0.582	**1.521**	**0.503**
	336	2.482	0.814	**2.104**	**0.768**	2.012	0.654	**1.678**	**0.594**	2.934	0.898	**2.521**	**0.813**	2.835	0.872	**2.671**	**0.852**
Weather	48	1.003	3.291	**0.413**	**1.710**	1.001	3.288	**0.447**	**1.897**	1.297	3.217	**0.402**	**1.844**	0.981	3.226	**0.432**	**1.698**
	96	0.335	1.886	**0.279**	**1.541**	0.334	1.886	**0.301**	**1.629**	0.658	3.327	**0.285**	**1.663**	0.246	1.642	**0.223**	**1.328**
	168	0.338	2.138	**0.207**	**1.421**	0.337	2.138	**0.263**	**1.581**	0.398	2.260	**0.253**	**1.708**	0.192	1.392	**0.182**	**1.217**
	336	0.385	2.375	**0.192**	**1.475**	0.384	2.373	**0.228**	**1.535**	0.570	2.996	**0.255**	**1.712**	0.194	1.520	**0.179**	**1.311**
CAISO	48	0.986	0.499	**0.641**	**0.383**	0.906	0.425	**0.627**	**0.377**	1.789	0.740	**0.823**	**0.467**	1.099	0.684	**0.514**	**0.341**
	96	0.898	0.451	**0.828**	**0.422**	0.833	0.424	**0.799**	**0.393**	2.428	0.826	**1.289**	**0.576**	0.886	0.440	**0.878**	**0.428**
	168	1.369	0.577	**1.161**	**0.500**	1.297	0.503	**1.071**	**0.411**	2.832	0.907	**1.344**	**0.575**	1.189	0.504	**1.150**	**0.498**
	336	2.541	0.852	**1.636**	**0.604**	2.124	0.712	**1.524**	**0.456**	3.148	0.960	**1.809**	**0.679**	1.612	0.607	**1.518**	**0.602**
NordPool	48	1.304	0.664	**1.156**	**0.636**	1.331	0.800	**1.233**	**0.769**	1.632	0.765	**1.478**	**0.732**	1.192	0.639	**1.150**	**0.632**
	96	1.771	0.799	**1.687**	**0.777**	1.851	0.936	**1.796**	**0.823**	2.187	0.889	**2.071**	**0.852**	1.644	0.764	**1.596**	**0.756**
	168	2.513	0.986	**2.009**	**0.865**	2.597	1.035	**2.584**	**0.908**	2.596	0.992	**2.430**	**0.957**	2.044	0.867	**1.918**	**0.836**
	336	2.946	1.040	**2.010**	**0.887**	3.022	0.976	**2.138**	**0.941**	2.993	1.236	**2.079**	**0.887**	2.557	0.953	**2.098**	**0.884**

effectiveness of our methods. Moreover, in certain situations, the improvements are even higher. Notably, with the lookback/horizon length as 168 on the Weather dataset, IN-Flow improves iTransformer by 32.5% (0.710 $\longrightarrow$ 0.479) and improves PatchTST by 33.2% (0.882 $\longrightarrow$ 0.589) on MSE. Another case on CAISO dataset is that the MSE of Autoformer with IN-Flow is reduced from 1.789 to 0.823, which achieves a very large (54.0%) improvement. The overall superior performance shows the great potential of IN-Flow in enhancing deep forecasting models on prediction.

5.2.2 Comparison with Other Non-stationary Forecasting Techniques. In this section, we further compare our performance with more recent normalization techniques, Dish-TS [15] and SAN [34] that handle distribution shifts in time series forecasting. Table 2 has shown the performance comparison in time series forecasting taking the PatchTST [37] as the backbone. From the results, we can observe that the existing Dish-TS can only improve the backbone in some shifted datasets. In some situations, Dish-TS might lead to worse performances, while SAN can perform better than it. Nevertheless, our IN-Flow can usually achieve the best performance. A potential explanation is that IN-Flow transforms data with multiple coupling and normalization layers with much expressiveness.

Besides, we also include the performance of IN-Flow in Table 3 compared with the SOTA architecture-specific method, Non-stationary Transformers [33], in which we consider three transformer-based models as backbones to compose Transformer, Informer, and Autoformer. We can easily observe that IN-Flow achieves a significant improvement over Non-stationary Transformers. The superiority signifies that even though IN-Flow is a model-agnostic method, it can have great adaptation for transformer-based models and even beat architecture-specific methods with tailor-designed mechanisms de-stationary attention in [33], which reveals the great potential of its compatibility to deep forecasting models.

5.3 Additional Experiments

5.3.1 Ablation Studies. In order to verify the effectiveness of our designs, we conduct adequate ablation studies by adjusting components of IN-Flow and considering some other transformation methods. Specifically, we consider four different variants for ablation tests: **RealNVP** [11] adopts the coupling layers [10] and the batch normalization [23]. In implementation, we mainly follow the reproduced implementation[1] and abandon some complicated modules such as multi-scale architectures [11, 26]. This method is shown in Figure 8(a) of Appendix. **RealNVP-c** [11] is a variant of RealNVP which only adopts the coupling layers. We use this method as comparison to verify the effectiveness of batch normalization [23] for transformation in forecasting. The method is shown in Figure 8(b) in Appendix. **IN-Flow-J** is a variant of IN-Flow (as shown in Figure 8(d) in Appendix), which jointly optimizes the

[1]https://github.com/kamenbliznashki/normalizing_flows

Table 2: Performance comparison of Dish-TS, SAN, and IN-Flow on six datasets under three seeds when taking the PatchTST as the backbone model.

Method		PatchTST		+Dish-TS		+SAN		+IN-Flow	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	48	1.128	1.865	0.975	1.661	0.971	1.668	**0.936**	**1.635**
	96	1.320	2.109	1.100	1.933	1.093	1.839	**1.025**	**1.782**
	168	2.001	2.679	1.229	2.024	1.110	1.950	**1.077**	**1.888**
	336	1.784	2.613	1.352	2.172	1.251	2.137	**1.121**	**2.021**
Weather	48	1.003	3.291	0.852	2.288	0.466	1.753	**0.413**	**1.710**
	96	0.335	1.886	0.314	1.856	0.309	1.777	**0.279**	**1.541**
	168	0.338	2.138	0.219	1.975	0.217	1.622	**0.207**	**1.421**
	336	0.385	2.375	0.256	2.612	0.239	1.907	**0.192**	**1.475**
CAISO	48	0.986	0.499	0.838	0.456	0.748	0.425	**0.641**	**0.383**
	96	1.098	0.451	1.216	0.541	0.901	0.446	**0.828**	**0.422**
	168	1.369	0.577	1.396	0.558	1.216	0.527	**1.161**	**0.500**
	336	2.541	0.852	1.925	0.708	1.831	0.683	**1.636**	**0.604**
NordPool	48	1.304	0.664	1.293	0.654	1.200	0.646	**1.156**	**0.636**
	96	1.971	0.892	1.895	0.924	1.812	0.800	**1.687**	**0.777**
	168	2.513	0.986	2.321	0.928	2.047	0.885	**2.009**	**0.865**
	336	2.946	1.040	2.707	1.010	2.014	0.889	**2.010**	**0.887**

Table 3: Performance comparison with non-stationary transformers, where Transformer, Informer, and Autoformer are coupled into non-stationary versions to compare with IN-Flow-coupled versions.

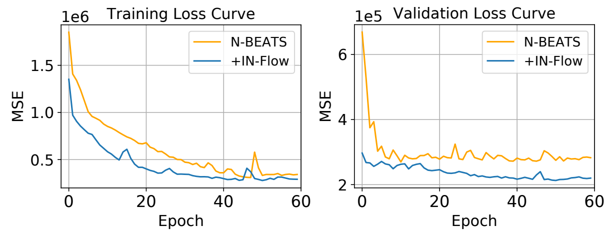
Datasets	ETTm2		Weather		CAISO	
Length	48	336	48	336	48	336
NS-Transformer	1.677	2.983	0.551	0.323	0.850	2.282
INF-Transformer	**1.352**	**2.053**	**0.360**	**0.210**	**0.757**	**1.548**
Improvement	19.4%	31.2%	34.6%	35.0%	10.9%	32.1%
NS-Informer	2.094	2.710	0.566	0.289	0.861	1.805
INF-Informer	**1.487**	**2.096**	**0.400**	**0.249**	**0.768**	**1.739**
Improvement	32.2%	22.7%	29.3%	13.8%	3.4%	3.7%
NS-Autoformer	1.553	2.117	0.580	0.375	0.844	1.923
INF-Autoformer	**1.427**	**2.068**	**0.421**	**0.261**	**0.794**	**1.535**
Improvement	8.1%	2.3%	27.7%	30.4%	5.9%	20.2%

transformation module and the forecasting module on the training data without bi-level optimization in the formulation. This method is to verify the effectiveness of bi-level optimization in IN-Flow. **IN-Flow-T** (post-norm) is another variant of IN-Flow, which changes the relative position of coupling layers and instance normalization and puts the instance normalization to the tail of flows, as shown in Figure 8(c) in Appendix. This method is to verify the architecture design of IN-Flow.

We conduct extensive ablation studies considering above four kinds of variants and taking N-BEATS and Autoformer as the backbone models respectively. We include the experimental ablation results on N-BEATS in Table 4. Based on the results, we can notice that RealNVP (with batch normalization) can always perform the worst in the comparison. This signifies the batch normalization is not proper for the transformation of time series. However, with pure coupling layers (RealNVP-c), it can perform well in some cases.

Table 4: Ablation studies taking N-BEATS as the backbone.

Method		RealNVP		RealNVP-c		IN-Flow-J		IN-Flow-T		IN-Flow	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	48	1.039	4.973	0.533	2.665	0.495	1.840	0.460	1.731	**0.432**	**1.698**
	96	1.033	5.354	0.225	1.698	0.234	1.365	0.251	1.376	**0.223**	**1.328**
	168	0.771	4.724	0.183	1.494	0.186	1.303	0.191	1.243	**0.182**	**1.217**
	336	0.604	4.647	0.190	1.579	0.183	1.416	0.193	1.382	**0.179**	**1.311**
NordPool	48	3.137	1.147	1.166	0.643	1.167	0.643	1.188	0.639	**1.150**	**0.632**
	96	2.541	1.029	1.602	0.758	1.620	0.759	1.597	0.757	**1.596**	**0.756**
	168	2.008	0.910	2.006	0.870	1.980	0.845	1.972	0.849	**1.918**	**0.836**
	336	2.100	0.921	2.283	0.927	2.152	0.898	2.082	0.880	**2.098**	**0.884**



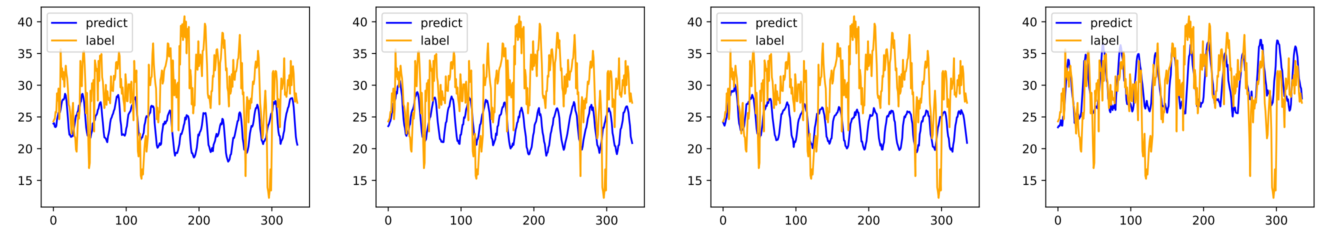
(a) Training Loss (b) Validation Loss

Figure 4: Learning loss curves with and without IN-Flow.

Both of these two observations motivate us to remove batch normalization for transformation and further propose the instance normalization instead. We can easily observe that, IN-Flow and its variants can always beat RealNVP series methods, which demonstrates the superiority of the instance normalization. Moreover, we notice that the joint optimization can sometime work well, especially in CAISO dataset. However, the overall performance of bi-level optimization becomes more stable and competitive. Thus, we finally adopt the bi-level optimization in our proposed methods.

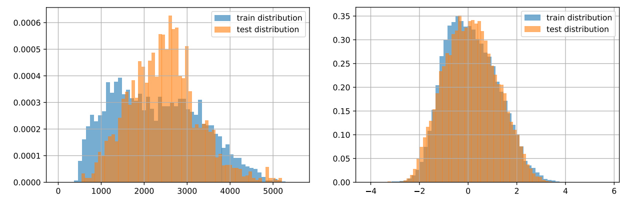
5.3.2 *Training Analysis.* To further study the learning process of IN-Flow, we visualize the learning loss curves of N-BEATS and IN-Flow. Figure 5(a) and Figure 5(d) demonstrate the training loss and validation loss curve on CAISO dataset respectively. We notice that with IN-Flow the training loss of N-BEATS has faster convergence which signifies the transformation module improves the learning of the forecasting module. After about 50 epochs, though the two training loss curves come close, the validation loss of IN-Flow is far lower than pure N-BEATS, which is attributed to the transformation module that relieves distribution shifts for fewer validation errors and better generalization in forecasting.

5.3.3 *Case Visualization.* We also include visualization case study towards forecasting. Figure 5 has shown a test sample predicted by RevIN, Dish-TS, SAN, and IN-Flow in the ETTm2 dataset when taking PatchTST as the backbone. Based on the results, we can find that all four methods can capture certain seasonal patterns. However, the amplitudes of the seasonal patterns are usually correctly learned. More importantly, we notice that in both RevIN, Dish-TS and SAN, the trend patterns are not correctly modeled, leading to a larger deviation towards the ground truth. This signifies that existing normalization techniques might abandon some useful information and thus have some worse cases. In contrast, our IN-Flow has adequate transformation ability for finishing forecasting by learning both trend and seasonality accurately.



(a) RevIN (b) Dish-TS (c) SAN (d) IN-Flow

Figure 5: Visualization of long-term 336 steps forecasting results of a test sample in ETTm2 dataset.



(a) Raw distribution (b) IN-Flow transformed distribution

Figure 6: The distribution visualization on NordPool dataset.

5.3.4 Distribution Visualization. In order to intuitively show the effectiveness of our IN-Flow, we have visualized the distribution of train and test sets before and after the transformation. Since the original time series is not i.i.d. and the window splitting decides the distribution, we visualize the distribution of the mean value of windows. Figure 6 shows the comparison of the distribution of train and test sets on the NordPool dataset. We can easily observe that in the original data, the train and test sets have obvious differences. After being transformed by IN-Flow, the distribution is much less distinguishable, showing the shift alleviation function of IN-Flow.

5.3.5 Statistic Measurements. We also include the statistical measurements of the distribution difference of train and test distribution before or after the transformation of IN-Flow. We select Wasserstein distance as our measurements. In order to intuitively show the effectiveness of our IN-Flow, we have visualized the distribution of train and test sets before and after the transformation. Since the original time series is not i.i.d. and the window splitting decides the distribution, we visualize the distribution of the mean value of windows of the data. The distribution difference between the train and test sets is shown in Table 5. From the results, we can observe that IN-Flow can largely reduce the distribution difference between train and test distribution. This provides large potential for the non-stationary forecasting.

Table 5: Wassertein distance of train and test set.

Dataset	Synthetic-2	ETTh1	Electricity
Train/Test Raw WD	0.2213	0.3270	0.3337
WD (with IN-Flow)	0.0783	0.0702	0.0547
Dataset	**Weather**	**CAISO**	**NordPool**
Train/Test Raw WD	0.1751	0.0944	0.1170
WD (with IN-Flow)	0.0775	0.0857	0.0471

6 Conclusion Remarks

In this paper, we propose a principled approach to address the time series distribution shift and contribute to non-stationary time series forecasting. Our core contribution lies in three aspects: a decoupled formulation to separate the removing-shift procedure from the forecasting and regard it as a special distribution transformation, a bi-level optimization problem to formalize the decoupled formulation and enable the joint learning of transformation and forecasting, and a novel invertible network, IN-Flow for time series distribution transformation. Moreover, distribution shift is a well-known and crucial topic but is still rarely studied in time series forecasting. We hope this principled approach, including the decoupled formulation, bi-level optimization, and IN-Flow can facilitate more research on distribution shift in time series and non-stationary forecasting.

7 Acknowledgments

Most of this work is done when the first author was a research assistant in State Key Laboratory of Internet of Things for Smart City at the University of Macau. This research is funded by the Science and Technology Development Fund (FDCT), Macau SAR (file no. 0123/2023/RIA2, 001/2024/SKL).

References

[1] Kumar Abhishek, MP Singh, Saswata Ghosh, and Abhishek Anand. 2012. Weather forecasting model using artificial neural network. *Procedia Technology* 4 (2012), 311–318.
[2] Diyar Akay and Mehmet Atak. 2007. Grey prediction with rolling mechanism for electricity demand forecasting of Turkey. *energy* 32, 9 (2007), 1670–1675.
[3] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. 2018. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730* (2018).
[4] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *CoRR* abs/1803.01271 (2018).

[5] Cristian Challu, Kin G. Olivares, Boris N. Oreshkin, Federico Garza, Max Mergenthaler, and Artur Dubrawski. 2022. N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting. *CoRR* abs/2201.12886 (2022).

[6] Can Chen, Xi Chen, Chen Ma, Zixuan Liu, and Xue Liu. 2022. Gradient-based bi-level optimization for deep learning: A survey. *arXiv preprint arXiv:2207.11719* (2022).

[7] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153, 1 (2007), 235–256.

[8] Jinliang Deng, Xiusi Chen, Renhe Jiang, Du Yin, Yi Yang, Xuan Song, and Ivor W Tsang. 2024. Disentangling Structured Components: Towards Adaptive, Interpretable and Scalable Time Series Forecasting. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[9] Jinliang Deng, Feiyang Ye, Du Yin, Xuan Song, Ivor Tsang, and Hui Xiong. 2024. Parsimony or Capability? Decomposition Delivers Both in Long-term Time Series Forecasting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=wiEHZSV15I

[10] Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516* (2014).

[11] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803* (2016).

[12] Yuntao Du, Jindong Wang, Wenjie Feng, Sinno Pan, Tao Qin, Renjun Xu, and Chongjun Wang. 2021. Adarnn: Adaptive learning and forecasting of time series. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. 402–411.

[13] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. 2016. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629* (2016).

[14] Wei Fan, Yanjie Fu, Shun Zheng, Jiang Bian, Yuanchun Zhou, and Hui Xiong. 2024. Dewp: Deep expansion learning for wind power forecasting. *ACM Transactions on Knowledge Discovery from Data* 18, 3 (2024), 1–21.

[15] Wei Fan, Pengyang Wang, Dongkun Wang, Dongjie Wang, Yuanchun Zhou, and Yanjie Fu. 2023. Dish-ts: a general paradigm for alleviating distribution shift in time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 7522–7529.

[16] Wei Fan, Kun Yi, Hangting Ye, Zhiyuan Ning, Qi Zhang, and Ning An. 2024. Deep frequency derivative learning for non-stationary time series forecasting. *arXiv preprint arXiv:2407.00502* (2024).

[17] Wei Fan, Shun Zheng, Xiaohan Yi, Wei Cao, Yanjie Fu, Jiang Bian, and Tie-Yan Liu. 2022. DEPTS: Deep Expansion Learning for Periodic Time Series Forecasting. In *International Conference on Learning Representations*.

[18] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. 2018. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*. PMLR, 1568–1577.

[19] Jindong Han, Hao Liu, Hengshu Zhu, Hui Xiong, and Dejing Dou. 2021. Joint Air Quality and Weather Prediction Based on Multi-Adversarial Spatiotemporal Networks. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

[20] Charles C Holt. 1957. Forecasting trends and seasonal by exponentially weighted moving averages. *ONR Memorandum* 52, 2 (1957).

[21] Charles C Holt. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20, 1 (2004), 5–10.

[22] Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*. 1501–1510.

[23] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, 448–456.

[24] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. 2022. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*.

[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems* 31 (2018).

[27] Ivan Kobyzev, Simon JD Prince, and Marcus A Brubaker. 2020. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence* 43, 11 (2020), 3964–3979.

[28] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In *SIGIR*. 95–104.

[29] Bryan Lim and Stefan Zohren. 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 379, 2194 (feb 2021), 20200209. doi:10.1098/rsta.2020.0209

[30] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

[31] Minhao Liu, Ailing Zeng, Muxi Chen, Zhijian Xu, Qiuxia Lai, Lingna Ma, and Qiang Xu. 2022. SCINet: time series modeling and forecasting with sample convolution and interaction. *Advances in Neural Information Processing Systems* 35 (2022), 5816–5828.

[32] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2023. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. *CoRR* abs/2310.06625 (2023).

[33] Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. 2022. Non-stationary Transformers: Exploring the Stationarity in Time Series Forecasting. In *Advances in Neural Information Processing Systems*.

[34] Zhiding Liu, Mingyue Cheng, Zhi Li, Zhenya Huang, Qi Liu, Yanhu Xie, and Enhong Chen. 2023. Adaptive Normalization for Non-stationary Time Series Forecasting: A Temporal Slice Perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[35] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. 2014. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems* 16, 2 (2014), 865–873.

[36] LIU Minhao, Ailing Zeng, Muxi Chen, Zhijian Xu, LAI Qiuxia, Lingna Ma, and Qiang Xu. 2022. SCINet: Time Series Modeling and Forecasting with Sample Convolution and Interaction. In *Advances in Neural Information Processing Systems*.

[37] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*.

[38] Eduardo Ogasawara, Leonardo C Martinez, Daniel De Oliveira, Geraldo Zimbrão, Gisele L Pappa, and Marta Mattoso. 2010. Adaptive normalization: A novel data normalization approach for non-stationary time series. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[39] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*.

[40] George Papamakarios, Eric T Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. Normalizing Flows for Probabilistic Modeling and Inference. *J. Mach. Learn. Res.* 22, 57 (2021), 1–64.

[41] George Papamakarios, Theo Pavlakou, and Iain Murray. 2017. Masked autoregressive flow for density estimation. *Advances in neural information processing systems* 30 (2017).

[42] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. 2019. Deep adaptive input normalization for time series forecasting. *IEEE transactions on neural networks and learning systems* 31, 9 (2019), 3760–3765.

[43] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[44] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. 2019. Meta-learning with implicit gradients. *Advances in neural information processing systems* 32 (2019).

[45] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep state space models for time series forecasting. *Advances in neural information processing systems* 31 (2018), 7785–7794.

[46] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.

[47] Sheng Shen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. 2020. Powernorm: Rethinking batch normalization in transformers. In *International Conference on Machine Learning*. PMLR, 8741–8751.

[48] Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. 2017. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 276–295.

[49] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016).

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[51] Peter Whittle. 1951. *Hypothesis testing in time series analysis*. Almqvist & Wiksells boktr.

[52] Peter Whittle. 1963. *Prediction and regulation by linear least-square methods*. English Universities Press.

[53] Peter R Winters. 1960. Forecasting sales by exponentially weighted moving averages. *Management science* 6, 3 (1960), 324–342.

[54] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. 2022. Deeptime: Deep time-index meta-learning for non-stationary time-series forecasting. *arXiv preprint arXiv:2207.06046* (2022).

[55] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems* 34 (2021), 22419–22430.

[56] Kun Yi, Jingru Fei, Qi Zhang, Hui He, Shufeng Hao, Defu Lian, and Wei Fan. 2024. FilterNet: Harnessing Frequency Filters for Time Series Forecasting. In

The Thirty-eighth Annual Conference on Neural Information Processing Systems. https://openreview.net/forum?id=ugL2D9idAD

[57] Kun Yi, Qi Zhang, Longbing Cao, Shoujin Wang, Guodong Long, Liang Hu, Hui He, Zhendong Niu, Wei Fan, and Hui Xiong. 2023. A Survey on Deep Learning based Time Series Analysis with Frequency Transformation. *CoRR* abs/2302.02173 (2023).

[58] Kun Yi, Qi Zhang, Wei Fan, Hui He, Liang Hu, Pengyang Wang, Ning An, Longbing Cao, and Zhendong Niu. 2024. FourierGNN: Rethinking multivariate time series forecasting from a pure graph perspective. *Advances in Neural Information Processing Systems* 36 (2024).

[59] Kun Yi, Qi Zhang, Wei Fan, Shoujin Wang, Pengyang Wang, Hui He, Ning An, Defu Lian, Longbing Cao, and Zhendong Niu. 2023. Frequency-domain MLPs are More Effective Learners in Time Series Forecasting. In *NeurIPS*.

[60] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2022. Are Transformers Effective for Time Series Forecasting? *arXiv preprint arXiv:2205.13504* (2022).

[61] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are Transformers Effective for Time Series Forecasting?. In *AAAI*. AAAI Press, 11121–11128.

[62] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11106–11115.

[63] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. 2022. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. *arXiv preprint arXiv:2201.12740* (2022).

A More Baseline Details

As aforementioned, our decoupled formulation with IN-Flow is model-agnostic such that it can be integrated with any deep time series forecasting models. To verify the effectiveness, we consider several state-of-the-art models as backbones. Mainly we reproduce the model by following their open source GitHub links. The detailed implementation details are as follows:

- **iTransformer** [32] inverts the structure of Transformer without modifying any existing modules by encoding individual series into variate tokens. These tokens are utilized by the attention mechanism to capture multivariate correlations and FFNs are adopted for each variate token to learn nonlinear representations. The official implementation is available at Github[2]. We follow the default parameters in the original paper in our experiments.
- **PatchTST** [37] divides time series data into subseries-level patches to extract local semantic information and adopts a channel-independence strategy where each channel shares the same embedding and Transformer weights across all the series. The official implementation is available at Github[3]. We follow the default parameters in the original paper.
- **Autoformer.** [55] considers the auto-correlation series and uses auto-correlation to replace the self-attention. We use the official open-source code of Autoformer[4] [55]. For hyperparameter, we follow the settings of the original paper including number of heads, moving average of Auto-correlation layer, dimension of Auto-Correlation layer, dimension of feed-forward layer, etc. In addition, we only adopt the positional embeddings and remove time embeddings.
- **N-BEATS.** We take the notable reproduced codes of N-BEATS[5]. Note that N-BEATS is an univariate time series forecasting model. To align with the input and output of multivariate settings (like Autoformer), we transform the multivariate lookback windows from dimension $B \times L \times D$ to $(B \times L) \times D$, where B is batch size, L is the lookback length and D is number of series. The horizon windows adopt the same strategy. In the mean time, for multivatiate forecasting on different datasets, we set the batch size B equal to $\lceil 1024/D \rceil$ to avoid very large batch size that influences training.

To overcome the distribution shift in time series forecasting, we take state-of-the-art model-agnostic normalization techniques, RevIN [24], Dish-TS [15] and SAN [34] as the main baselines. We adopt the official implementation[6] of RevIN, the official implementation[7] of Dish-TS, the official implementation[8] of SAN. We run our models in three different seeds and report the average performances.

We also consider another recent method that's designed for the non-stationarity of transformer-based forecasting models, namely Nonstationary Transformers [33] for comparison. To couple with Nonstationary Transformers, we take three model as backbones, including vanilla Transformer [50], Informer [62] and Autoformer [55]. We adopt the official implementation[9] of Nonstationary Transformers to set the parameters of nonstationary layers, including nonstationary hidden layers as 2 and hidden size as 128. Specifically,

- **Nonstationary Transformer**. Based on the implementation of vanilla Transformer [50], Nonstationary Transformer applies series stationarization layer and revise the raw self attention into the de-stationary self attention [33]. Other hyperparameters are set the same as the vanilla Transformer.
- **Nonstationary Informer**. Based on the original implementation[10] of Informer [62], Nonstationary Informer also adds series stationarization layer and changes the PropSparse attention [62] into the de-stationary PropSparse attention version [33]. Other hyperparameters are the same as the Informer.
- **Nonstationary Autoformer**. Since Autoformer replaces classic self attention layers with auto-correlation layers [55], Nonstationary Autoformer also adds series stationarization layers and adopts the revised de-stationary auto-correlation layers [33] for learning. And other hyperparameters are the same as the Autoformer.

Notably, when we integrate the backbones with our IN-Flow, we set all the other hyperparameters the same as the backbones under the same experimental settings (including the random seeds) in order to conduct a fair comparison.

B More Details on Synthetic Data

We make two synthetic datasets, namely Synthetic-1 and Synthetic-2, We mainly follow the simple rules mentioned in Section 5.1, and accordingly evaluate the performance on the synthetic datasets.

Now we illustrate the specific details of the choice of $\mathcal{P}_u$ and τ for the three datasets. Specifically, in the u-th segment, $s_t = A_u \cos(2\pi \frac{1}{T_u} t + B_u) + C_u$ where the distribution $\mathcal{P}_u$ of u-th segment is controlled by the amplitude parameters A_u, the period parameters T_u, the phase parameters B_u and the level parameters C_u. In order to make time series always shifted, we randomly sample the values

[2]https://github.com/thuml/iTransformer
[3]https://github.com/yuqinie98/PatchTST
[4]https://github.com/thuml/Autoformer
[5]https://github.com/ElementAI/N-BEATS
[6]https://github.com/ts-kim/RevIN
[7]https://github.com/weifantt/Dish-TS
[8]https://github.com/icantnamemyself/SAN
[9]https://github.com/thuml/Nonstationary_Transformers/
[10]https://github.com/zhouhaoyi/Informer2020

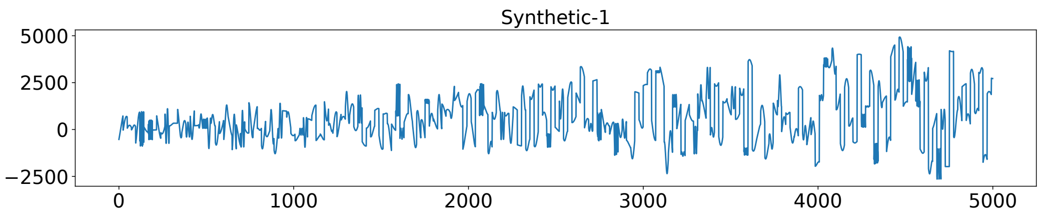


Figure 7: An example series of Synthetic-1 dataset.

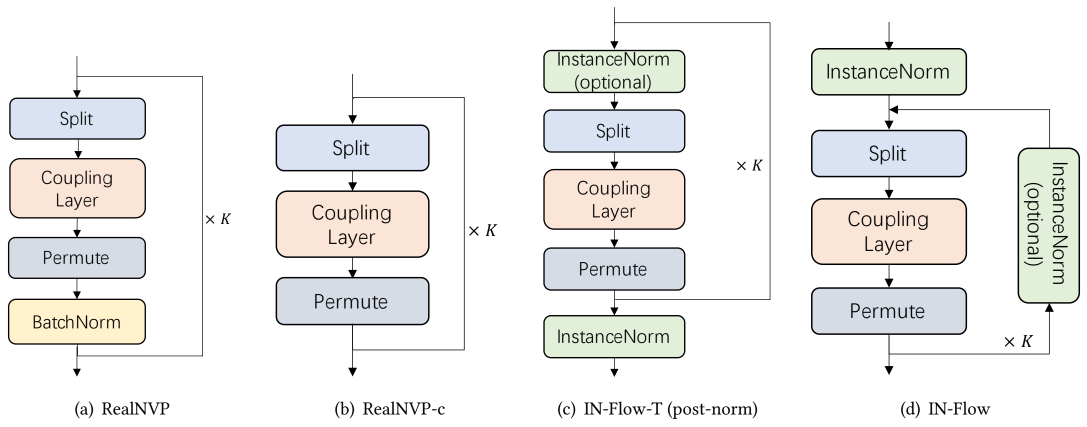


Figure 8: Four kinds of variants in ablation studies, where K stands for the number of blocks in flows; "split" and "permute" stand for two basic operations for coupling layers to assure reversibility [10, 11].

of A_u, T_u, B_u, C_u to make every segment shifted constantly. We let the amplitude A_u randomly sampled from $(-1000, 1000)$, the period T_u randomly sampled from $(0, 100)$, the phase B_u randomly sampled from $(0, 100)$ and the level C_u randomly sampled from $(-\lceil t/100 \rceil * 50, -\lceil t/100 \rceil * 100)$, where t is the timestamp. Then, we can control the distribution changing frequency through adjusting τ. We set $T = 10,000$ to synthetic 10,000 points for each series, in order to make enough data for training and evaluation. We take the first 6000 points for training, and 2000 points for validation and another 2000 points for test. We make Synthetic-1 dataset by setting $\tau = 24$ and we make the Synthetic-2 dataset by setting $\tau = 12$ in order to model different distribution shift situations. We let each dataset include 5 distinct series, each of which is generated by setting different seeds. Figure 7 demonstrates an example series from the Synthetic-1 dataset respectively, where we show the first five thousand points for visualization.

C More Details of Evaluation on Real-world Data

C.1 More Implementation Details

For a stable evaluation, we rerun all the models under four different seeds and report the average MSE/MAE on the testset of each dataset. In the main experiments of time series forecasting, we let the lookback window and the horizon window have the same length, where we gradually prolong the length as $\{48, 96, 168, 336\}$ to accommodate short-term/long-term forecasting settings. We train all the models using L2 loss and Adam [25] optimizer with learning rate of [1e-4, 1e-3]. The batchsize is set as 1024 for N-BEATS and 128 for others. For every model, we use early stopping with the patience as five steps. For IN-Flow, we traverse the hyperparameters in the validation set: with the number of blocks in coupling layers as $\{2, 8, 16\}$, hidden size as 128. The learning rate for the transformation module is set as 1e-4. All the experiments are implemented with PyTorch [43] on single NVIDIA A100 40GB GPU.

C.2 More Evaluation Metric Details

To directly reflects the shift in time series, all experiments and evaluations are conducted on original data without any data normalization or data scaling, which is different from experimental settings of [55, 62] which use z-score normalization to process data before training and evaluation.

In evaluation, we evaluate the time series forecasting performances on the mean squared error (MSE) and mean absolute error (MAE). Note that our experiments and evaluations are on original data; thus the reported metrics on real-world data are scaled for readability. Specifically, for MSE, the report metrics are scaled by $1e^{-1}$ on *ETTh1* dataset and *ETTm2* dataset, by $1e^{-4}$ on *Weather* and *Electricity* dataset, by $1e^{-6}$ on *CAISO* dataset and by $1e^{-6}$ on *NordPool* dataset. For MAE, the reported metrics are not scaled for ETT series datasets. In contrast, the reported MAE is scaled by $1e^{-1}$ on *Weather* dataset, by $1e^{-3}$ on *CAISO* dataset and by $1e^{-2}$ on *NordPool* dataset.