Non-autoregressive Conditional Diffusion Models for Time Series Prediction

Lifeng Shen [1] **James T. Kwok** [2]

Abstract

Recently, denoising diffusion models have led to significant breakthroughs in the generation of images, audio and text. However, it is still an open question on how to adapt their strong modeling ability to model time series. In this paper, we propose TimeDiff, a non-autoregressive diffusion model that achieves high-quality time series prediction with the introduction of two novel conditioning mechanisms: future mixup and autoregressive initialization. Similar to teacher forcing, future mixup allows parts of the ground-truth future predictions for conditioning, while autoregressive initialization helps better initialize the model with basic time series patterns such as short-term trends. Extensive experiments are performed on nine real-world datasets. Results show that TimeDiff consistently outperforms existing time series diffusion models, and also achieves the best overall performance across a variety of the existing strong baselines (including transformers and FiLM).

1. Introduction

Time series prediction has been applied to many real-world applications such as economics (Henrique et al., 2019), transportation (Sapankevych & Sankar, 2009), and energy (Wang et al., 2011). As time series prediction can be regarded as a conditional generation task, it is natural to use diffusion models (Rasul et al., 2021). By generating data through iterative denoising, diffusion models present a strong ability for data generation and have led to breakthroughs in synthesizing images (Rombach et al., 2022), audio (Chen et al., 2020) and text (Gong et al., 2023; Li et al., 2022a). However, it is still an open question on how to build a strong diffusion model for time series prediction. A key challenge is that time series data usually involve complex dynamics, nonlinear patterns, and long temporal dependencies. Time series prediction is thus challenging, especially when the prediction horizon is long (Zhou et al., 2022a).

Existing time-series diffusion models can be roughly divided into two categories based on the decoding strategy. The first one is autoregressive (Rasul et al., 2021; Yang et al., 2022), in which future predictions are generated one by one over time. However, it has limited long-range prediction performance due to error accumulation and slow inference speed. To alleviate these problems, the second category of diffusion models is non-autoregressive, such as CSDI (Tashiro et al., 2021) and SSSD (Alcaraz & Strodthoff, 2022). They perform conditioning in the denoising networks' intermediate layers and introduce inductive bias into the denoising objective. However, as will be shown empirically in Section 4.2, their long-range prediction performance is still inferior to other time-series prediction models such as Fedformer (Zhou et al., 2022b) and NBeats (Oreshkin et al., 2019). This may be due to that their conditioning strategies are borrowed from image or text data, but not tailored for time series data. Only using the denoising objective to introduce inductive bias may not be sufficient to guide the conditioning network in capturing helpful information from the lookback window, leading to inaccurate predictions.

In this paper, we propose TimeDiff, a conditional non-autoregressive diffusion model that is effective for long time series prediction. Unlike CSDI and SSSD, it introduces additional inductive bias in the conditioning module that is tailor-made for time series. Specifically, TimeDiff introduces two conditioning mechanisms: (i) future mixup, which randomly reveals parts of the ground-truth future predictions during training, and (ii) autoregressive initialization, which better initializes the model with basic components in the time series. Experimental results on nine real-world datasets demonstrate superiority of TimeDiff over existing time series diffusion models and other recent strong baselines (such as time series transformers (Nie et al., 2023; Zhou et al., 2022b; Wu et al., 2021; Liu et al., 2021; Zhou et al., 2021), DLinear (Zeng et al., 2023) and FiLM (Zhou

[1]Division of Emerging Interdisciplinary Areas, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. [2]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. Correspondence to: Lifeng Shen <lshenae@connect.ust.hk>, James T. Kwok <jamesk@cse.ust.hk>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

et al., 2022a)).

2. Preliminaries

2.1. Diffusion Models

Diffusion models consist of a forward diffusion process and a backward denoising process. A well-known diffusion model is the denoising diffusion probabilistic model (DDPM) (Ho et al., 2020). By gradually adding noise, the forward diffusion process transforms an input $\mathbf{x}^0$ to a white Gaussian noise vector $\mathbf{x}^K$ in K diffusion steps. At step $k \in [1, K]$, the diffused sample $\mathbf{x}^k$ is obtained by scaling $\mathbf{x}^{k-1}$ with $\sqrt{1-\beta_k}$ and adding i.i.d. Gaussian noise, as:

$$q(\mathbf{x}^k|\mathbf{x}^{k-1}) = \mathcal{N}(\mathbf{x}^k; \sqrt{1-\beta_k}\mathbf{x}^{k-1}, \beta_k\mathbf{I}),$$

where $\beta_k \in [0,1]$ is the noise variance following a predefined schedule. It can be shown that

$$q(\mathbf{x}^k|\mathbf{x}^0) = \mathcal{N}(\mathbf{x}^k; \sqrt{\bar{\alpha}_k}\mathbf{x}^0, (1-\bar{\alpha}_k)\mathbf{I}), \tag{1}$$

where $\alpha_k := 1-\beta_k$ and $\bar{\alpha}_k := \Pi_{s=1}^k \alpha_s$. Thus, $\mathbf{x}^k$ can be directly obtained as

$$\mathbf{x}^k = \sqrt{\bar{\alpha}_k}\mathbf{x}^0 + \sqrt{1-\bar{\alpha}_k}\epsilon, \tag{2}$$

where ϵ is sampled from $\mathcal{N}(0, \mathbf{I})$. Note that (2) also allows $\mathbf{x}^0$ to be easily recovered from $\mathbf{x}^k$.

The backward denoising process is a Markovian process:

$$p_\theta(\mathbf{x}^{k-1}|\mathbf{x}^k) = \mathcal{N}(\mathbf{x}^{k-1}; \mu_\theta(\mathbf{x}^k, k), \Sigma_\theta(\mathbf{x}^k, k)). \tag{3}$$

In practice, $\Sigma_\theta(\mathbf{x}^k, k)$ is often fixed at $\sigma_k^2\mathbf{I}$, and $\mu_\theta(\mathbf{x}^k, k)$ is modeled by a neural network parameterized by θ.

To train the diffusion model, one uniformly samples k from $\{1, 2, \ldots, K\}$ and then minimizes the following KL-divergence

$$\mathcal{L}_k = D_{\text{KL}}\left(q(\mathbf{x}^{k-1}|\mathbf{x}^k)||p_\theta(\mathbf{x}^{k-1}|\mathbf{x}^k)\right). \tag{4}$$

For more stable training, $q(\mathbf{x}^{k-1}|\mathbf{x}^k)$ is often replaced by

$$q(\mathbf{x}^{k-1}|\mathbf{x}^k, \mathbf{x}^0) = \mathcal{N}(\mathbf{x}^{k-1}; \tilde{\mu}_k(\mathbf{x}^k, \mathbf{x}^0, k), \tilde{\beta}_k\mathbf{I}), \tag{5}$$

where $\tilde{\beta}_k = \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k$ and

$$\tilde{\mu}_k(\mathbf{x}^k, \mathbf{x}^0, k) = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\mathbf{x}^0 + \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\mathbf{x}^k.$$

The training objective in (4) is then transformed as

$$\mathcal{L}_k = \frac{1}{2\sigma_k^2}\|\tilde{\mu}_k(\mathbf{x}^k, \mathbf{x}^0, k) - \mu_\theta(\mathbf{x}^k, k)\|^2. \tag{6}$$

In (6), $\mu_\theta(\mathbf{x}^k, k)$ can be defined in two ways (Benny & Wolf, 2022): (i) $\mu_\epsilon(\epsilon_\theta)$ or (ii) $\mu_\mathbf{x}(\mathbf{x}_\theta)$. In the former case, $\mu_\epsilon(\epsilon_\theta)$ is computed from a noise prediction model $\epsilon_\theta(\mathbf{x}^k, k)$:

$$\mu_\epsilon(\epsilon_\theta) = \frac{1}{\sqrt{\alpha_k}}\mathbf{x}^k - \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}\sqrt{\alpha_k}}\epsilon_\theta(\mathbf{x}^k, k). \tag{7}$$

Ho et al. (2020) show that optimizing the following simplified training objective leads to better generation quality:

$$\mathcal{L}_\epsilon = \mathbb{E}_{k,\mathbf{x}^0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}^k, k)\|^2\right], \tag{8}$$

where ϵ is the noise used to obtain $\mathbf{x}^k$ from $\mathbf{x}^0$ in (2) at step k. Alternatively, $\mu_\mathbf{x}(\mathbf{x}_\theta)$ can be obtained from a data prediction model $\mathbf{x}_\theta(\mathbf{x}^k, k)$ as

$$\mu_\mathbf{x}(\mathbf{x}_\theta) = \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\mathbf{x}^k + \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\mathbf{x}_\theta(\mathbf{x}^k, k). \tag{9}$$

The corresponding simplified loss is

$$\mathcal{L}_\mathbf{x} = \mathbb{E}_{k,\mathbf{x}^0,\epsilon}\left[\|\mathbf{x}^0 - \mathbf{x}_\theta(\mathbf{x}^k, k)\|^2\right]. \tag{10}$$

Note that the noise prediction model $\epsilon_\theta(\mathbf{x}^k, k)$ in (8) and the data prediction model $\mathbf{x}_\theta(\mathbf{x}^k, k)$ in (10) are both conditioned on the diffusion step k only. When an additional condition input $\mathbf{c}$ is available, this can be injected into the backward denoising step (3) as

$$p_\theta(\mathbf{x}^{k-1}|\mathbf{x}^k, \mathbf{c}) = \mathcal{N}(\mathbf{x}^{k-1}; \mu_\theta(\mathbf{x}^k, k|\mathbf{c}), \sigma_k^2\mathbf{I}). \tag{11}$$

We can then obtain conditional extensions of (7)-(10) by replacing $\epsilon_\theta(\mathbf{x}^k, k)$ (resp. $\mathbf{x}_\theta(\mathbf{x}^k, k)$) with $\epsilon_\theta(\mathbf{x}^k, k|\mathbf{c})$ (resp. $\mathbf{x}_\theta(\mathbf{x}^k, k|\mathbf{c})$). For example, using the data prediction model, we have

$$\mu_\mathbf{x}(\mathbf{x}_\theta) = \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\mathbf{x}^k + \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\mathbf{x}_\theta(\mathbf{x}^k, k|\mathbf{c}). \tag{12}$$

2.2. Conditional DDPMs for Time Series Prediction

In time series prediction, we aim to predict the future values $\mathbf{x}_{1:H}^0 \in \mathbb{R}^{d\times H}$ of a time series given its past observations $\mathbf{x}_{-L+1:0}^0 \in \mathbb{R}^{d\times L}$. Here, d is the number of variables in the possibly multivariate time series, H is the length of forecast window, and L is the length of lookback window.

Conditional DDPMs perform time series prediction by modeling the distribution

$$p_\theta(\mathbf{x}_{1:H}^{0:K}|\mathbf{c}) = p_\theta(\mathbf{x}_{1:H}^K)\prod_{k=1}^K p_\theta(\mathbf{x}_{1:H}^{k-1}|\mathbf{x}_{1:H}^k, \mathbf{c}),$$

where $\mathbf{x}_{1:H}^K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{c} = \mathcal{F}(\mathbf{x}_{-L+1:0}^0)$ is the output of the conditioning network $\mathcal{F}$ that takes the past observations $\mathbf{x}_{-L+1:0}^0$ as input, and

$$p_\theta(\mathbf{x}_{1:H}^{k-1}|\mathbf{x}_{1:H}^{k}, \mathbf{c}) = \mathcal{N}(\mathbf{x}_{1:H}^{k-1}; \mu_\theta(\mathbf{x}_{1:H}^{k}, k|\mathbf{c}), \sigma_k^2\mathbf{I}).$$

It is still an open question how to design an efficient denoising network μ_θ and conditioning network $\mathcal{F}$ in time series diffusion models.

TimeGrad (Rasul et al., 2021) is a recent denoising diffusion model for time series prediction. It generates future values in an autoregressive manner by modeling the joint distribution $p_\theta(\mathbf{x}_{1:H}^{0:K})$, where $\mathbf{x}_{1:H}^{0:K} = \{\mathbf{x}_{1:H}^{0}\} \bigcup \{\mathbf{x}_{1:H}^{k}\}_{k=1,\dots,K}$:

$$\begin{aligned} & p_\theta\left(\mathbf{x}_{1:H}^{0:K}|\mathbf{c} = \mathcal{F}(\mathbf{x}_{-L+1:0}^{0})\right) \\ &= \prod_{t=1}^{H} p_\theta\left(\mathbf{x}_{t}^{0:K}|\mathbf{c} = \mathcal{F}(\mathbf{x}_{-L+1:t-1}^{0})\right) \\ &= \prod_{t=1}^{H} p_\theta(\mathbf{x}_t^K) \prod_{k=1}^{K} p_\theta\left(\mathbf{x}_t^{k-1}|\mathbf{x}_t^k, \mathbf{c} = \mathcal{F}(\mathbf{x}_{-L+1:t-1}^{0})\right). \end{aligned}$$

In TimeGrad, $\mathcal{F}$ is a recurrent neural network that uses its hidden state $\mathbf{h}_t$ as $\mathbf{c}$. Similar to (8), its training objective is

$$\mathcal{L}_\epsilon = \mathbb{E}_{k,\mathbf{x}^0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_t^k, k|\mathbf{h}_t)\|^2\right].$$

TimeGrad has been successfully used for short-term time series prediction. However, with its use of autoregressive decoding, error can accumulate and inference is also slow. These are particularly problematic for long-range prediction.

Another denoising diffusion model for time series prediction is CSDI (Tashiro et al., 2021). Instead of training a shared diffusion model across time steps as in TimeGrad, CSDI avoids autoregressive inference on future values by diffusing and denoising the whole time series $\mathbf{x}_{-L+1:H}^{0}$. During training, the denoising model takes $\mathbf{x}_{-L+1:H}^{0}$ and a binary mask $\mathbf{m} \in \{0,1\}^{d\times(L+H)}$ as input, where $\mathbf{m}_{i,t} = 0$ if position i is observed at time t, and 1 otherwise. A self-supervised strategy is also introduced by further masking some input observations. The following loss is used during training:

$$\mathcal{L}_\epsilon = \mathbb{E}_{k,\mathbf{x}^0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\mathbf{x}_{\texttt{target}}^k, k|\mathbf{c} = \mathcal{F}(\mathbf{x}_{\texttt{observed}}^k))\|^2\right],$$

where $\mathbf{x}_{\texttt{target}}^k = \mathbf{m} \odot \mathbf{x}_{-L+1:H}^{0}$ is the masked part of the time series, and $\mathbf{x}_{\texttt{observed}}^k = (1-\mathbf{m}) \odot \mathbf{x}_{-L+1:H}^{0}$ is the observed part.

Although directly generating the future time series in this non-autoregressive manner avoids the error accumulation issue in TimeGrad, CSDI is still limited in two aspects: (i) CSDI's denoising network is based on two transformers, whose complexity is quadratic in the number of variables and length of time series. This can easily run out of memory when modeling long multivariate time series. (ii) Its conditioning is based on masking, similar to inpainting in computer vision. However, it is shown that this may cause disharmony at the boundaries between masked and observed regions (Lugmayr et al., 2022).

SSSD (Alcaraz & Strodthoff, 2022) replaces the transformers in CSDI by a structured state space model, thus avoiding the quadratic complexity issue. However, it still uses the same non-autoregressive strategy as CSDI, and can still have deteriorated performance due to boundary disharmony.

There are some recent attempts in the NLP community to develop sequence diffusion models with non-autoregressive decoding over time, e.g., DiffuSeq (Gong et al., 2023). Unlike natural language generation, time series prediction is more challenging as this requires modeling temporal dependencies on irregular, highly nonlinear, and noisy data.

3. Proposed Model

While conditional diffusion models have been widely used, they usually focus on capturing the semantic similarities across modalities (e.g., text and image) (Choi et al., 2021; Kim et al., 2022). However, to model real-world nonstationary time series, capturing the complex temporal dependencies maybe even more important. In this section, we propose TimeDiff, with novel conditioning mechanisms that are tailored for time series data. Figure 1 shows an overview of the proposed model.

3.1. Forward Diffusion Process

In TimeDiff, the forward diffusion process is straightforward. Based on (2), we obtain the diffused $\mathbf{x}_{1:H}^{k}$ by

$$\mathbf{x}_{1:H}^{k} = \sqrt{\bar{\alpha}_k}\mathbf{x}_{1:H}^{0} + \sqrt{1-\bar{\alpha}_k}\epsilon,$$

where ϵ is sampled from $\mathcal{N}(0, \mathbf{I})$ with the same size as $\mathbf{x}_{1:H}^{0}$.

The recent D^{3}VAE (Li et al., 2022b) also uses the same forward diffusion process on the lookback window $\mathbf{x}_{-L+1:0}^{0}$. However, strictly speaking, D^{3}VAE is not a diffusion model as the diffused $\mathbf{x}_{1:H}^{k}$ is produced by a deep variational autoencoder (VAE) (Kingma & Welling, 2014) with $\mathbf{x}_{-L+1:0}^{k}$ (instead of $\mathbf{x}_{1:H}^{k}$) as input and does not denoise from random noise. This makes the denoising process more difficult.

3.2. Conditioning the Backward Denoising Process

At the kth denoising step, $\mathbf{x}_{1:H}^{k}$ is denoised to $\mathbf{x}_{1:H}^{k-1}$. In order to well predict the future time series segment $\mathbf{x}_{1:H}^{0}$, useful information needs to be extracted from the lookback window $\mathbf{x}_{-L+1:0}^{0}$ to guide the denoising of $\mathbf{x}_{1:H}^{k}$ to $\mathbf{x}_{1:H}^{0}$ (Figure 1, left).

The proposed inductive bias on the conditioning network is specific to time series prediction. Specifically, Section 3.2.1 proposes the use of mixup (Zhang et al., 2018) to combine the past and future time series information into $\mathbf{z}_{\texttt{mix}}$ (in (14)).

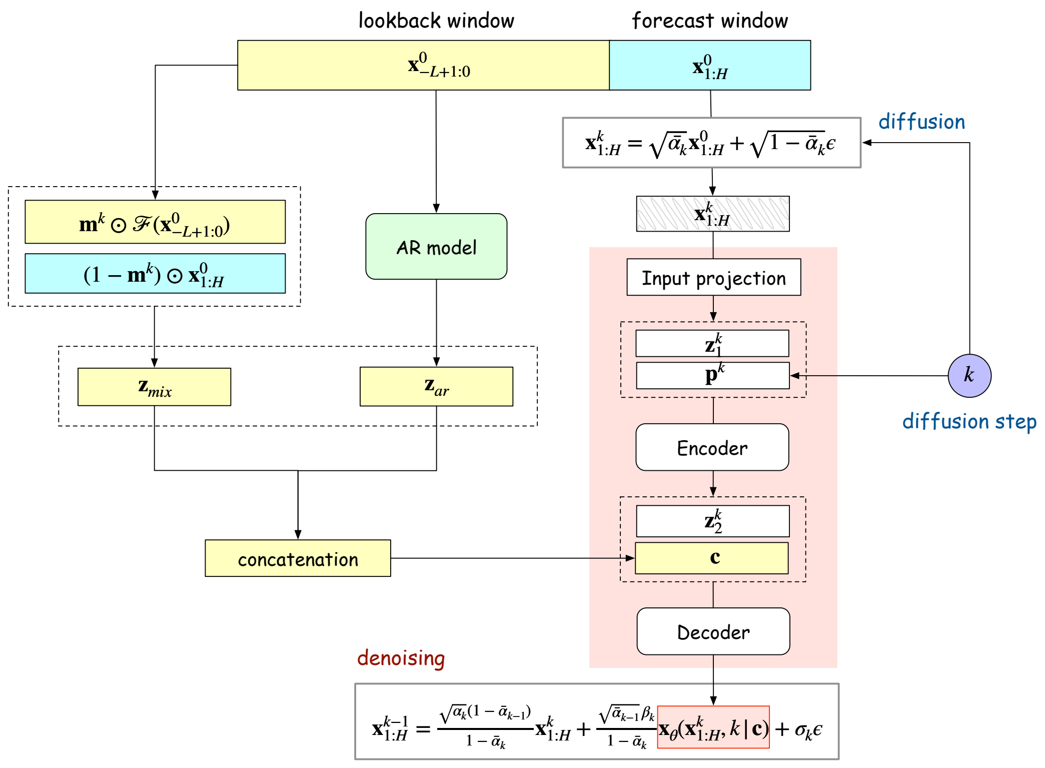

lookback window
forecast window
$\mathbf{x}^0_{-L+1:0}$
$\mathbf{x}^0_{1:H}$
$\mathbf{x}^k_{1:H} = \sqrt{\bar{\alpha}_k}\mathbf{x}^0_{1:H} + \sqrt{1-\bar{\alpha}_k}\epsilon$
diffusion
$\mathbf{m}^k \odot \mathcal{F}(\mathbf{x}^0_{-L+1:0})$
$(1-\mathbf{m}^k) \odot \mathbf{x}^0_{1:H}$
AR model
$\mathbf{x}^k_{1:H}$
Input projection
$\mathbf{z}^k_1$
$\mathbf{p}^k$
k
diffusion step
$\mathbf{z}_{mix}$
$\mathbf{z}_{ar}$
Encoder
$\mathbf{z}^k_2$
$\mathbf{c}$
concatenation
Decoder
denoising
$\mathbf{x}^{k-1}_{1:H} = \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\mathbf{x}^k_{1:H} + \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\mathbf{x}_\theta(\mathbf{x}^k_{1:H}, k \mid \mathbf{c}) + \sigma_k\epsilon$

Figure 1. An illustration of the proposed TimeDiff. $\mathbf{x}^0_{-L+1:0}$ contains the past observations and $\mathbf{x}^0_{1:H}$ contains the future ground-truth outputs.

Section 3.2.2 proposes an autoregressive model to produce a crude approximation $\mathbf{z}_{ar}$ (in (16)) of $\mathbf{x}^0_{1:H}$. Finally, these two are concatenated along the channel dimension to form the condition

$$\mathbf{c} = \text{concat}([\mathbf{z}_{\texttt{mix}}, \mathbf{z}_{ar}]) \in \mathbb{R}^{2d \times H}. \quad (13)$$

3.2.1. Future Mixup

Since the goal is to predict $\mathbf{x}^0_{1:H}$, the ideal condition to guide the denoising process is $\mathbf{x}^0_{1:H}$ itself. While obviously $\mathbf{x}^0_{1:H}$ cannot be accessed on inference, it is available during training. Inspired by mixup (Zhang et al., 2018), we propose a simple yet effective mechanism called *future mixup*. It combines the past information's mapping $\mathcal{F}(\mathbf{x}^0_{-L+1:0})$ and the future ground-truth $\mathbf{x}^0_{1:H}$. During training, at diffusion step k, it produces the conditioning signal

$$\mathbf{z}_{\texttt{mix}} = \mathbf{m}^k \odot \mathcal{F}(\mathbf{x}^0_{-L+1:0}) + (1-\mathbf{m}^k) \odot \mathbf{x}^0_{1:H}. \quad (14)$$

Here, $\mathbf{m}^k \in [0,1)^{d \times H}$ is a mixing matrix with each element randomly sampled from the uniform distribution on $[0,1)$, and $\odot$ is the Hadamard product. We use a convolution network as $\mathcal{F}$, which is commonly used for the modeling of local temporal patterns and long-range dependencies in time series (Wang et al., 2017; Li et al., 2019). On inference, $\mathbf{x}^0_{1:H}$ is no longer available, and we simply set

$$\mathbf{z}_{\texttt{mix}} = \mathcal{F}(\mathbf{x}^0_{-L+1:0}). \quad (15)$$

Future mixup is similar to teacher forcing (Williams & Zipser, 1989) and scheduled sampling (Bengio et al., 2015), which also introduce ground-truth observation as input during training but only use the model's prediction during inference. However, future mixup is for non-autoregressive conditional generation in time series diffusion models, while teacher forcing and scheduled sampling are for autoregressive decoding of recurrent networks. Moreover, future mixup mixes the past observations' embedding and future time series (which is also similar to the bidirectional LSTM (Graves et al., 2013)), while teacher forcing and scheduled sampling replace the model's prediction at the previous step by the ground-truth past observation.

3.2.2. Autoregressive Model

In image inpainting applications, non-autoregressive models often produce disharmony at the boundaries between masked and observed regions (Lugmayr et al., 2022). In the context of time series prediction, this translates to disharmony between the history and forecast segments, as will also be empirically demonstrated in Section 4.2.

Algorithm 1 Training.

```
Require: Number of diffusion steps $K$; pretrained AR model $\mathcal{M}_{ar}$.
 1: repeat
 2:     Sample $\mathbf{x}^0_{1:H}$ from the training set;
 3:     $k \sim \texttt{Uniform}(\{1, 2, \dots, K\})$;
 4:     $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;
 5:     generate diffused sample $\mathbf{x}^k_{1:H}$ (as in (2)) by
        $\mathbf{x}^k_{1:H} = \sqrt{\bar{\alpha}_k}\mathbf{x}^0_{1:H} + \sqrt{1-\bar{\alpha}_k}\epsilon$;
 6:     obtain diffusion step $k$'s embedding $\mathbf{p}^k$ using (17);
 7:     randomly generate a matrix $\mathbf{m}^k$ in (14);
 8:     obtain $\mathbf{z}_{\texttt{mix}}$ by future mixup using (14);
 9:     obtain $\mathbf{z}_{ar}$ by (16);
10:     obtain condition $\mathbf{c}$ based on $\mathbf{z}_{\texttt{mix}}$ and $\mathbf{z}_{ar}$ by (13);
11:     use the denoising network to generate denoised sample $\mathbf{x}^{k-1}_{1:H}$ by (18);
12:     calculate the loss $\mathcal{L}_k(\theta)$ in (19);
13:     take gradient descent step on $\nabla_\theta L_k(\theta)$;
14: until converged.
```

To alleviate this problem, we use a linear autoregressive (AR) model $\mathcal{M}_{ar}$ to provide an initial guess $\mathbf{z}_{ar} \in \mathbb{R}^{d\times H}$ for $\mathbf{x}^0_{1:H}$. Let $\mathbf{x}^0_i$ be the ith column of $\mathbf{x}^0_{-L+1:0}$. We define

$$\mathbf{z}_{ar} = \sum_{i=-L+1}^{0} \mathbf{W}_i \odot \mathbf{X}^0_i + \mathbf{B}, \tag{16}$$

where $\mathbf{X}^0_i \in \mathbb{R}^{d\times H}$ is a matrix containing H copies of $\mathbf{x}^0_i$, and $\mathbf{W}_i$'s $\in \mathbb{R}^{d\times H}$, $\mathbf{B} \in \mathbb{R}^{d\times H}$ are trainable parameters.

The AR model $\mathcal{M}_{ar}$ is pretrained on the training set by minimizing the ℓ_2-distance between $\mathbf{z}_{ar}$ and the ground-truth $\mathbf{x}^0_{1:H}$. The number of pretraining epochs can be few (in the experiments, it is set to 20).

While obviously this simple AR model cannot accurately approximate a complex nonlinear time series in general, it can still capture simple patterns, such as short-term trends (Lai et al., 2018). Moreover, note that though $\mathcal{M}_{ar}$ is an autoregressive model, it does not require autoregressive decoding. In other words, all columns of $\mathbf{z}_{ar}$ are obtained simultaneously, instead of obtaining them one by one sequentially. This avoids the error accumulation and slow inference problems in TimeGrad.

3.3. Denoising Network

The denoising network is shown in red in Figure 1. When denoising $\mathbf{x}^k_{1:H} \in \mathbb{R}^{d\times H}$, the diffusion-step embedding $\mathbf{p}^k$ is first combined with the diffused input $\mathbf{x}^k_{1:H}$'s embedding $\mathbf{z}^k_1 \in \mathbb{R}^{d'\times H}$, where $\mathbf{z}^k_1$ is obtained by an input projection block consisting of several convolution layers. As in (Rasul et al., 2021; Tashiro et al., 2021; Kong et al., 2020), we obtain the representation $\mathbf{p}^k \in \mathbb{R}^{d'}$ of diffusion step k using

Algorithm 2 Inference.

```
Require: Trained denoising network $\mathbf{x}_\theta$; trained conditioning network $\mathcal{F}$; pretrained AR model $\mathcal{M}_{ar}$.
 1: $\mathbf{x}^K_{1:H} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;
 2: for $k = K, \dots, 1$ do
 3:     $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, if $k > 1$, else $\epsilon = 0$;
 4:     obtain diffusion step $k$'s embedding $\mathbf{p}^k$ using (17);
 5:     obtain $\mathbf{z}_{\texttt{mix}}$ by (15);
 6:     obtain $\mathbf{z}_{ar}$ by (16);
 7:     obtain condition $\mathbf{c}$ based on $\mathbf{z}_{\texttt{mix}}$ and $\mathbf{z}_{ar}$ using (13);
 8:     sample $\hat{\mathbf{x}}^{k-1}_{1:H}$ by (18);
 9:     $\mathbf{x}^{k-1}_{1:H} = \hat{\mathbf{x}}^{k-1}_{1:H}$;
10: end for
11: return $\hat{\mathbf{x}}^0_{1:H}$.
```

the transformer's sinusoidal position embedding (Vaswani et al., 2017). Specifically, we first have

$$k_{\texttt{embedding}} = [\sin(10^{\frac{0\times 4}{w-1}}t), \dots, \sin(10^{\frac{w\times 4}{w-1}}t), \cos(10^{\frac{0\times 4}{w-1}}t), \dots, \cos(10^{\frac{w\times 4}{w-1}}t)],$$

where $w = \frac{d'}{2}$, and then use two fully-connected (FC) layers both with default hidden dimensions of 128 on $k_{\texttt{embedding}}$ to obtain

$$\mathbf{p}^k = \text{SiLU}(\text{FC}(\text{SiLU}(\text{FC}(k_{\texttt{embedding}})))) \in \mathbb{R}^{d'\times 1}, \tag{17}$$

where SiLU is the sigmoid-weighted linear unit (Elfwing et al., 2018).

For the concatenation, $\mathbf{p}^k$ is broadcasted over length to form $[\mathbf{p}^k, \dots, \mathbf{p}^k] \in \mathbb{R}^{d'\times H}$, and then concatenated with $\mathbf{z}^k_1$ along the channel dimension. The concatenated result is a tensor of size $2d' \times H$. A multilayer convolution-based encoder is then used to obtain the representation $\mathbf{z}^k_2 \in \mathbb{R}^{d''\times H}$.

A decoder is used to fuse $\mathbf{c}$ and $\mathbf{z}^k_2$. First, we concatenate $\mathbf{c}$ and $\mathbf{z}^k_2$ along the variable dimension to generate an input of size $(2d + d'') \times H$. The decoder consists of multiple convolution layers. Its output $\mathbf{x}_\theta(\mathbf{x}^k_{1:H}, k|\mathbf{c})$ is of size $d\times H$, the same as $\mathbf{x}^k_{1:H}$. Finally, we generate the denoised output $\hat{\mathbf{x}}^{k-1}_{1:H}$ as in (12):

$$\hat{\mathbf{x}}^{k-1}_{1:H} = \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\mathbf{x}^k_{1:H} + \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}\mathbf{x}_\theta(\mathbf{x}^k_{1:H}, k|\mathbf{c}) + \sigma_k\epsilon, \tag{18}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Note that we predict the data $\mathbf{x}_\theta(\mathbf{x}^k_{1:H}, k)$ for denoising, rather than predicting the noise $\epsilon_\theta(\mathbf{x}^k_{1:H}, k)$. As time series data usually contain highly irregular noisy components, estimating the diffusion noise ϵ can be more difficult.

Table 1. Summary of dataset statistics, including dimension, total observations, sampling frequency, and prediction length used in the experiments.

dataset	dim	#observations	freq.	H (steps)
NorPool	18	70,128	1 hour	1 month (720)
Caiso	10	74,472	1 hour	1 month (720)
Weather	21	52,696	10 mins	1 week (672)
ETTm1	7	69,680	15 mins	2 days (192)
Wind	7	48,673	15 mins	2 days (192)
Traffic	862	17,544	1 hour	1 week (168)
Electricity	321	26,304	1 hour	1 week (168)
ETTh1	7	17,420	1 hour	1 week (168)
Exchange	8	7,588	1 day	2 weeks (14)

3.4. Training

The training procedure is shown in Algorithm 1. For each $\mathbf{x}^0_{1:H}$, we first randomly sample a batch of diffusion steps k's, and then minimize a conditioned variant of (10): $\min_\theta \mathcal{L}(\theta) = \min_\theta \mathbb{E}_{\mathbf{x}^0_{1:H},\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I}),k}\mathcal{L}_k(\theta)$, where

$$\mathcal{L}_k(\theta) = \|\mathbf{x}^0_{1:H} - \mathbf{x}_\theta(\mathbf{x}^k_{1:H}, k|\mathbf{c})\|^2. \quad (19)$$

3.5. Inference

During inference (Algorithm 2), we first generate a noise vector $\mathbf{x}^K_{1:H} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ of size $d \times H$. By repeatedly running the denoising step (18) till k equals 1 (ϵ is set to zero when $k = 1$), we obtain the time series $\hat{\mathbf{x}}^0_{1:H}$ as final prediction.

4. Experiments

In this section, we perform extensive experiments to compare the proposed TimeDiff with a variety of time series prediction models on nine real-world datasets.

4.1. Setup

数据集

Experiments are performed on nine real-world time series datasets (Table 1) (Zhou et al., 2021; Wu et al., 2021; Fan et al., 2022): (i) *NorPool*[1], which includes eight years of hourly energy production volume series in multiple European countries; (ii) *Caiso*[2], which contains eight years of hourly actual electricity load series in different zones of California; (iii) *Traffic*[3], which records the hourly road occupancy rates generated by sensors in the San Francisco Bay area freeways; (iv) *Electricity*[4], which includes the hourly electricity consumption of 321 clients over two years; (v) *Weather*[5], which records 21 meteorological indicators at 10-minute intervals from 2020 to 2021; (vi) *Exchange*(Lai et al., 2018), which describes the daily exchange rates of eight countries (Australia, British, Canada, Switzerland, China, Japan, New Zealand, and Singapore); (vii)-(viii) *ETTh1* and *ETTm1*, which contain two years of electricity transformer temperature data (Zhou et al., 2021) collected in China, at 1-hour and 15-minute intervals, respectively; (ix) *Wind*, which contains wind power records from 2020-2021 at 15-minute intervals (Li et al., 2022b).

For *NorPool*, following (Fan et al., 2022), the training set covers observations before April 1, 2020, the validation set is from April 1 to October 1, 2020, while the test set is for after October 1, 2020. For *Caiso*, following (Fan et al., 2022), the training set covers observations before January 1, 2020, the validation set is for January 1 to October 1, 2020, while the test set is for after October 1, 2020. For the other datasets, we follow (Wu et al., 2021; Zhou et al., 2022b) and split the whole data dataset into training, validation, and test sets in chronological order with the ratio of 6:2:2 for *ETTh1* and *ETTm1*, and 7:1:2 for *Weather*, *Wind*, *Traffic*, *Electricity*, and *Exchange*.

As can be seen in Table 1, all the datasets used are multivariate. Besides running the models directly on these multivariate datasets, we also convert them to univariate time series for performance comparison. For *NorPool* and *Caiso*, the univariate time series are extracted from all variables as in (Fan et al., 2022). For the other datasets, we follow (Wu et al., 2021; Zhou et al., 2022b) and extract the univariate time series by using the last variable only. Further details and example visualizations are in Appendix A.

A variety of baselines are compared, including: (i) Time series diffusion models, including TimeGrad (Rasul et al., 2021), CSDI (Tashiro et al., 2021), SSSD (Alcaraz & Strodthoff, 2022), and D^{3}VAE (Li et al., 2022b); (ii) SOTA time series prediction methods, including FiLM (Zhou et al., 2022a), Depts (Fan et al., 2022) and NBeats (Oreshkin et al., 2019); (iii) Time series transformers, including PatchTST [6] (Nie et al., 2023), Fedformer (Zhou et al., 2022b), Autoformer (Wu et al., 2021), Pyraformer (Liu et al., 2021), Informer (Zhou et al., 2021) and the standard Transformer (Vaswani et al., 2017); and (iv) DLinear (Zeng et al., 2023) and LSTMa (Bahdanau et al., 2015), an attention-based LSTM (Hochreiter & Schmidhuber, 1997). For all methods, the history length is selected from {96, 192, 720,

[1]`https://www.nordpoolgroup.com/Market-data1/Power-system-data`

[2]`http://www.energyonline.com/Data`

[3]`http://pems.dot.ca.gov`

[4]`https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`

[5]`https://www.bgc-jena.mpg.de/wetter/`

[6]PatchTST uses a channel-independence setup, in which each variate of the multivariate time series is predicted independently. However, this then cannot assess the model's ability to capture multivariate dependencies and may not be fair to the other models. Thus, we follow the traditional setup and does not use channel-independence in PatchTST. Additional experiments on using the channel-independence setup can be found in Appendix C.

Table 2. Testing MSE in the univariate setting. Number in brackets is the rank. The best is in bold, while the second best is underlined.

	NorPool	*Caiso*	*Weather*	*ETTm1*	*Wind*	*Traffic*	*Electricity*	*ETTh1*	*Exchange*	avg rank
TimeDiff	0.636 (2)	0.122 (3)	**0.002** (2)	0.040 (2)	2.407 (9)	**0.121** (1)	**0.232** (1)	**0.066** (1)	0.017 (3)	**2.7**
TimeGrad	1.129 (15)	0.325 (15)	**0.002** (2)	0.048 (6.5)	2.530 (12)	1.223 (16)	0.920 (16)	0.078 (8)	0.041 (13.5)	11.6
CSDI	0.967 (14)	0.192 (9)	**0.002** (2)	0.050 (10)	2.434 (10.5)	0.393 (13)	0.520 (12)	0.083 (11)	0.071 (16)	10.8
SSSD	1.145 (16)	0.176 (7)	0.004 (6)	0.049 (8.5)	3.149 (15)	0.151 (6)	0.370 (5)	0.097 (14)	0.023 (10.5)	9.8
D^{3}VAE	0.964 (13)	0.521 (16)	0.003 (4.5)	0.044 (4)	2.679 (13)	0.151 (6)	0.535 (13)	0.078 (8)	0.019 (7)	9.4
FiLM	0.707 (5)	0.185 (8)	0.007 (10)	**0.038** (1)	**2.143** (1)	0.198 (10)	0.260 (3)	0.070 (2.5)	0.018 (5.5)	5.1
Depts	0.668 (3)	**0.107** (1)	0.024 (13)	0.046 (5)	3.457 (16)	0.151 (6)	0.380 (8)	0.070 (2.5)	0.020 (8.5)	7.0
NBeats	0.768 (6)	0.125 (4)	0.137 (15)	0.048 (6.5)	2.434 (10.5)	0.142 (4)	0.378 (7)	0.095 (13)	**0.016** (1)	7.4
PatchTST	**0.595** (1)	0.193 (10)	0.026 (14)	0.052 (12)	2.698 (14)	0.177 (9)	0.450 (11)	0.106 (15)	0.020 (8.5)	10.5
FedFormer	0.891 (9)	0.164 (5)	0.005 (7.5)	0.065 (15)	2.351 (8)	0.173 (8)	0.376 (6)	0.076 (5)	0.050 (15)	8.7
Autoformer	0.946 (12)	0.248 (11)	0.003 (4.5)	0.051 (11)	2.349 (7)	0.473 (14)	0.659 (15)	0.081 (10)	0.041 (13.5)	10.9
Pyraformer	0.933 (11)	0.165 (6)	0.020 (12)	0.054 (13)	2.279 (3)	0.136 (2)	0.389 (9)	0.076 (5)	0.017 (3)	7.1
Informer	0.804 (7)	0.250 (12.5)	0.007 (10)	0.049 (8.5)	2.297 (4)	0.213 (11)	0.363 (4)	0.076 (5)	0.023 (10.5)	8.1
Transformer	0.928 (10)	0.250 (12.5)	0.007 (10)	0.058 (14)	2.306 (6)	0.238 (12)	0.430 (10)	0.092 (12)	0.018 (5.5)	10.2
DLinear	0.671 (4)	0.118 (2)	0.168 (16)	0.041 (3)	2.171 (2)	0.139 (3)	0.244 (2)	0.078 (8)	0.017 (3)	4.8
LSTMa	0.836 (8)	0.253 (14)	0.005 (7.5)	0.091 (16)	2.299 (5)	1.032 (15)	0.596 (14)	0.167 (16)	0.031 (12)	11.9

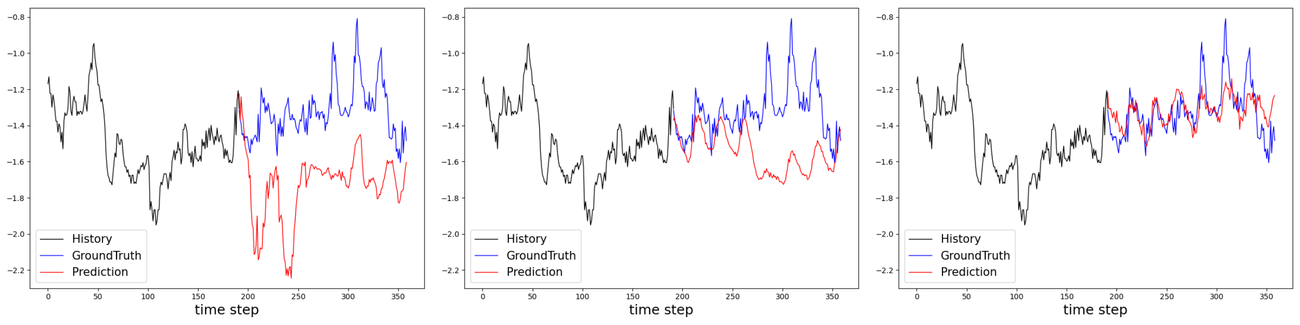
Table 3. Testing MSE in the multivariate setting. Number in brackets is the rank. The best is in bold, while the second best is underlined. CSDI runs out of memory on *Traffic* and *Electricity*.

	NorPool	*Caiso*	*Weather*	*ETTm1*	*Wind*	*Traffic*	*Electricity*	*ETTh1*	*Exchange*	avg rank
TimeDiff	0.665 (2)	0.136 (2)	**0.311**(1)	**0.336** (1)	**0.896** (1)	0.564 (3)	**0.193** (1)	**0.407** (1)	0.018 (3)	**1.7**
TimeGrad	1.152 (15)	0.258 (14)	0.392 (10)	0.874 (14)	1.209 (15)	1.745 (15)	0.736 (15)	0.993 (15)	0.079 (13)	13.9
CSDI	1.011 (14)	0.253 (13)	0.356 (5)	0.529 (11)	1.066 (5)	-	-	0.497 (4)	0.077 (12)	10.6
SSSD	0.872 (8)	0.195 (6)	0.349 (4)	0.464 (9)	1.188 (13)	0.642 (6)	0.255 (7)	0.726 (12)	0.061 (9)	8.1
D^{3}VAE	0.745 (5)	0.241 (12)	0.375 (7)	0.362 (4)	1.118 (11)	0.928 (12)	0.286 (10)	0.504 (5)	0.200 (15)	8.9
FiLM	0.723 (4)	0.179 (4)	0.327 (2)	0.347 (3)	0.984 (3)	0.628 (5)	0.210 (3)	0.426 (3)	**0.016** (1.5)	3.2
Depts	**0.662** (1)	**0.106** (1)	0.761 (14)	0.380 (6)	1.082 (8)	1.019 (14)	0.319 (12)	0.579 (9.5)	0.020 (4)	7.7
NBeats	0.832 (6)	0.141 (3)	1.344 (16)	0.391 (7)	1.069 (6)	**0.373** (1)	0.269 (8)	0.586 (11)	**0.016** (1.5)	6.6
PatchTST	0.851 (7)	0.193 (5)	0.782 (15)	0.372 (5)	1.070 (7)	0.831 (11)	0.225 (5)	0.526 (7)	0.047 (7)	7.7
FedFormer	0.873 (9)	0.205 (7)	0.342 (3)	0.426 (8)	1.113 (10)	0.591 (4)	0.238 (6)	0.541 (8)	0.133 (14)	7.6
Autoformer	0.940 (10)	0.226 (10)	0.360 (6)	0.565 (12)	1.083 (9)	0.688 (10)	0.201 (2)	0.516 (6)	0.056 (8)	9.0
Pyraformer	1.008 (13)	0.273 (15)	0.394 (11)	0.493 (10)	1.061 (4)	0.659 (7)	0.273 (9)	0.579 (9.5)	0.032 (6)	9.4
Informer	0.985 (11)	0.231 (11)	0.385 (8)	0.673 (13)	1.168 (12)	0.664 (8)	0.298 (11)	0.775 (14)	0.073 (11)	10.9
Transformer	1.005 (12)	0.206 (8)	0.388 (9)	0.992 (15)	1.201 (14)	0.671 (9)	0.328 (13)	0.759 (13)	0.062 (10)	11.3
DLinear	0.670 (3)	0.461 (16)	0.488 (12)	0.345 (2)	0.899 (2)	0.389 (2)	0.215 (4)	0.415 (2)	0.022 (5)	5.3
LSTMa	1.481 (16)	0.217 (9)	0.662 (13)	1.030 (16)	1.464 (16)	0.966 (13)	0.414 (14)	1.149 (16)	0.403 (16)	14.2

1440} by using the validation set.

For the proposed TimeDiff, the data is preprocessed with instance normalization (Kim et al., 2021) as in (Zhou et al., 2022a). Specifically, we subtract the mean value in the lookback window from each time series variable, and then divide it by the lookback window's standard deviation. On inference, the mean and standard deviation are added back to the final prediction. For the transformer baselines, series stationarization (Liu et al., 2022) is used as in (Liu et al., 2022).

We train the proposed model using Adam (Kingma & Ba, 2015) with a learning rate of 10^{-3}. The batch size is 64, and training with early stopping for a maximum of 100 epochs. $K = 100$ diffusion steps are used, with a cosine variance schedule (Rasul et al., 2021) starting from $\beta_1 = 10^{-4}$ to $\beta_K = 10^{-1}$. To accelerate the inference of diffusion models, many learning-free efficient samplers have been developed, such as DDIM (Song et al., 2021), Analytic-DPM (Bao et al., 2022), and DPM-Solver (Lu et al., 2022). In this work, we use the DPM-Solver. Empirically, the number of denoising steps can be reduced to below 20.



(a) CSDI. (b) SSSD. (c) TimeDiff.

Figure 2. Visualizations on *ETTh1* by CSDI, SSSD and the proposed TimeDiff.

The mean squared error (MSE) is used for performance evaluation. As in (Rasul et al., 2021; Tashiro et al., 2021), the time series diffusion models are evaluated by averaging over 10 predictions. All experiments are run on a Nvidia RTX A6000 GPU with 40GB memory. More implementation details can be found in Appendix B.

Table 4. MSEs by different variants of the conditioning network.

future mixup	AR	*Caiso*	*Electricity*	*Exchange*	*ETTh1*
✓	✓	**0.122**	**0.232**	**0.017**	**0.066**
✓	✗	0.149	0.328	0.020	0.086
✗	✓	0.160	0.295	0.022	0.162
✗	✗	0.170	0.340	0.024	0.182

4.2. Results

Tables 2 and 3 show the testing MSEs in the univariate and multivariate settings. As can be seen, the proposed method consistently outperforms existing time series diffusion models (TimeGrad, CSDI, SSSD, and D^{3}VAE). It also achieves the best overall performance across all the baselines. Note that CSDI can easily run out of memory on long multivariate time series (Table 3) due to its use of transformers.

Figure 2 compares the predictions obtained by the three non-autoregressive diffusion models CSDI, SSSD, and the proposed TimeDiff on a randomly selected (univariate) *ETTh1* sample. As can be seen, while CSDI and SSSD provide good prediction in the very short-term (from steps 192-200), their longer-term predictions differ significantly from the ground-truth. On the other hand, TimeDiff better captures both the trend and periodic patterns.

4.3. Ablation Studies

In this section, we study the effectiveness of the various proposed components. Four representative univariate datasets in Table 1 are used: *Caiso* and *Electricity*, which contain obvious periodic patterns; and *Exchange* and *ETTh1*, which are nonstationary (as discussed in Appendix A).

4.3.1. Conditioning mechanism

In this experiment, we study the effectiveness of future mixup (section 3.2.1) and autoregressive model (AR) (section 3.2.2). We consider the four combinations when each of these is used or not used in the conditioning network. When neither is used, we set the condition $\mathbf{c}$ to $\mathcal{F}(\mathbf{x}^0_{-L+1:0})$.

Table 4 shows the testing MSE. As can be seen, both future mixup and AR model lead to improved performance. In particular, the performance on *ETTh1* degrades significantly when future mixup is not used.

4.3.2. Mixup Strategies in Future Mixup

Recall that in future mixup, elements of $\mathbf{m}^k$ are randomly sampled from the uniform distribution on $[0, 1)$. In this experiment, we compare this strategy (which will be called soft-mixup) with two variants: (i) Hard-mixup: The sampled values in $\mathbf{m}^k$ are binarized by a threshold $\tau \in (0, 1)$ and (ii) Segment-mixup: The mask $\mathbf{m}^k$ is generated by the procedure in (Zerveas et al., 2021). Each masked segment has a length following the geometric distribution with a mean of 3. This is then followed by an unmasked segment with mean length $3(1-\tau)/\tau$.

Results are shown in Table 5. Note that soft-mixup does not require the extra hyper-parameter τ. As can be seen, soft-mixup has the best prediction performance. On the other hand, hard-mixup is sensitive to the setting of τ.

4.3.3. Predicting $\mathbf{x}_\theta$ vs Predicting ϵ_θ

In this experiment, we compare with the more common denoising strategy that is based on noise prediction $\mu_\epsilon(\epsilon_\theta)$ (Rasul et al., 2021). Here, the architecture of noise prediction network ϵ_θ is the same as that of the denoising network $\mathbf{x}_\theta$ in previous sections. The only difference is that they have

Table 5. MSEs of different mixing strategies in future mixup.

	τ	*Caiso*	*Electricity*	*Exchange*	*ETTh1*
soft-mixup	-	0.122	**0.232**	**0.017**	**0.066**
hard-mixup	0.1	0.146	0.289	0.031	0.161
	0.3	0.123	0.318	0.021	0.143
	0.5	**0.118**	0.260	0.018	0.081
	0.7	0.122	0.255	0.017	0.074
	0.9	0.124	0.264	0.018	0.074
segment-mixup	0.1	0.152	0.335	0.032	0.172
	0.3	0.121	0.323	0.020	0.132
	0.5	0.119	0.285	0.018	0.079
	0.7	0.124	0.254	0.018	0.072
	0.9	0.125	0.253	0.018	0.075

Table 6. MSEs of two denoising strategies: Predicting $\mathbf{x}_\theta$ vs predicting ϵ_θ.

denoising strategy	*Caiso*	*Electricity*	*Exchange*	*ETTh1*
$\mathbf{x}_\theta$	**0.122**	**0.232**	**0.017**	**0.66**
ϵ_θ	0.134	0.317	0.021	0.077

different objectives: predicting noise or predicting data.

Results are shown in Table 6. As can be seen, predicting the data $\mathbf{x}_\theta$ directly is more effective. This might be due to that real-world time series usually contains highly nonlinear noise, which can be easily confused with the noise generated from the diffusion process.

4.4. Integration into Existing Diffusion Models

The proposed future mixup and autoregressive initialization are general techniques and can be used with existing non-autoregressive time series diffusion models. In this section, we integrate them into two non-autoregressive diffusion models: CSDI and SSSD. Specifically, we concatenate the conditioning network's outputs with their conditioning masks along the channel dimension. Experiment is performed on the univariate *ETTh1* and *ETTm1*.

Table 7 shows the results. As can be seen, using future mixup and autoregressive initialization leads to improved performance. Note, however, that even with these enhancements, CSDI still falls short of the performance achieved by the proposed TimeDiff. Moreover, while the enhanced variant of SSSD achieves comparable performance as TimeDiff on *ETTm1*, it is still outperformed by TimeDiff on *ETTh1*.

4.5. Inference Efficiency

In this experiment, we compare the inference efficiency of the proposed TimeDiff with the other time series diffusion model baselines (TimeGrad, CSDI, SSSD). Table 8 shows the inference time on the univariate *ETTh1* with different

Table 7. MSEs of CSDI and SSSD with and without future mixup / autoregressive (AR) initialization.

	future mixup	AR	*ETTh1*	*ETTm1*
CSDI	✗	✗	0.083	0.050
	✓	✗	0.078	0.045
	✗	✓	0.088	0.054
	✓	✓	**0.075**	**0.043**
SSSD	✗	✗	0.097	0.049
	✓	✗	0.077	0.044
	✗	✓	0.084	0.052
	✓	✓	**0.071**	**0.040**

values of the prediction horizon H. As can be seen, the proposed TimeDiff is significantly faster than the others across all the H values. In particular, TimeGrad is the slowest due to its use of auto-regressive decoding.

Table 8. Inference time (ms) on the univariate *ETTh1* with different prediction horizon H.

	$H = 96$	$H = 168$	$H = 336$	$H = 720$
TimeDiff	**16.2**	**17.2**	**26.5**	**34.6**
TimeGrad	870.2	1579.2	3119.7	6724.1
CSDI	90.41	127.2	398.9	513.1
SSSD	418.6	595.0	1054.2	2516.9

5. Conclusion

In this paper, we propose TimeDiff, a novel diffusion model for time series prediction. By using two conditioning mechanisms (future mixup and autoregressive initialization), useful inductive bias is added to the conditioning network's outputs and helps the denoising process. Results on a number of real-world datasets show that the proposed method produces better prediction results than existing time series diffusion models. The proposed method also achieves the best overall performance across existing strong baselines. Besides, ablation studies demonstrate the effectiveness of each component in the proposed model.

One limitation of TimeDiff is that it has difficulties in learning the multivariate dependencies when the time series has a large number of variables (e.g., *Traffic*). To alleviate this problem, in the future we will consider capturing the dependencies by integrating with graph neural networks. 未来研究方向

Acknowledgements

This research was supported in part by the Research Grants Council of the Hong Kong Special Administrative Region (Grant 16200021).

References

Alcaraz, J. M. L. and Strodthoff, N. Diffusion-based time series imputation and forecasting with structured state space models. Technical report, arXiv, 2022.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Bao, F., Li, C., Zhu, J., and Zhang, B. Analytic-DPM: An analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *International Conference on Learning Representations*, 2022.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. *Neural Information Processing Systems*, 28, 2015.

Benny, Y. and Wolf, L. Dynamic dual-output diffusion models. In *Computer Vision and Pattern Recognition*, 2022.

Chen, N., Zhang, Y., Zen, H., Weiss, R. J., Norouzi, M., and Chan, W. WaveGrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2020.

Choi, J., Kim, S., Jeong, Y., Gwon, Y., and Yoon, S. ILVR: Conditioning method for denoising diffusion probabilistic models. In *International Conference on Computer Vision*, 2021.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Elliott, G., Rothenberg, T. J., and Stock, J. H. Efficient tests for an autoregressive unit root. *Econometrica*, pp. 813–836, 1996.

Fan, W., Zheng, S., Yi, X., Cao, W., Fu, Y., Bian, J., and Liu, T.-Y. DEPTS: Deep expansion learning for periodic time series forecasting. In *International Conference on Learning Representations*, 2022.

Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. DiffuSeq: Sequence to sequence text generation with diffusion models. In *International Conference on Learning Representations*, 2023.

Graves, A., Jaitly, N., and Mohamed, A.-R. Hybrid speech recognition with deep bidirectional LSTM. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 273–278, 2013.

Henrique, B. M., Sobreiro, V. A., and Kimura, H. Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*, 124: 226–251, 2019.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Neural Information Processing Systems*, 2020.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Kim, H., Kim, S., and Yoon, S. Guided-TTS: A diffusion model for text-to-speech via classifier guidance. In *International Conference on Machine Learning*, 2022.

Kim, T., Kim, J., Tae, Y., Park, C., Choi, J.-H., and Choo, J. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Kong, Z., Ping, W., Huang, J., Zhao, K., and Catanzaro, B. DiffWave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2020.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. Modeling long-and short-term temporal patterns with deep neural networks. In *SIGIR Conference on Research & Development in Information Retrieval*, 2018.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Neural Information Processing Systems*, 2019.

Li, X., Thickstun, J., Gulrajani, I., Liang, P. S., and Hashimoto, T. B. Diffusion-LM improves controllable text generation. In *Neural Information Processing Systems*, 2022a.

Li, Y., Lu, X., Wang, Y., and Dou, D. Generative time series forecasting with diffusion, denoise, and disentanglement. In *Neural Information Processing Systems*, 2022b.

Liu, S., Yu, H., Liao, C., Li, J., Lin, W., Liu, A. X., and Dustdar, S. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2021.

Liu, Y., Wu, H., Wang, J., and Long, M. Non-stationary transformers: Rethinking the stationarity in time series forecasting. In *Neural Information Processing Systems*, 2022.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. In *Neural Information Processing Systems*, 2022.

Lugmayr, A., Danelljan, M., Romero, A., Yu, F., Timofte, R., and Van Gool, L. Repaint: Inpainting using denoising diffusion probabilistic models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Nie, Y., Nguyen, N. H., Sinthong, P., and Kalagnanam, J. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

Oreshkin, B. N., Carpov, D., Chapados, N., and Bengio, Y. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2019.

Rasul, K., Seward, C., Schuster, I., and Vollgraf, R. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, 2021.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Sapankevych, N. I. and Sankar, R. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Tashiro, Y., Song, J., Song, Y., and Ermon, S. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. In *Neural Information Processing Systems*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Neural Information Processing Systems*, 30, 2017.

Wang, X., Guo, P., and Huang, X. A review of wind power forecasting models. *Energy procedia*, 12:770–778, 2011.

Wang, Z., Yan, W., and Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks*, 2017.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Neural Information Processing Systems*, 2021.

Yang, R., Srivastava, P., and Mandt, S. Diffusion probabilistic modeling for video generation. Technical report, arXiv, 2022.

Zeng, A., Chen, M., Zhang, L., and Xu, Q. Are transformers effective for time series forecasting? In *AAAI Conference on Artificial Intelligence*, 2023.

Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., and Eickhoff, C. A transformer-based framework for multivariate time series representation learning. In *SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

Zhang, Y. and Yan, J. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *International Conference on Learning Representations*, 2023.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI Conference on Artificial Intelligence*, 2021.

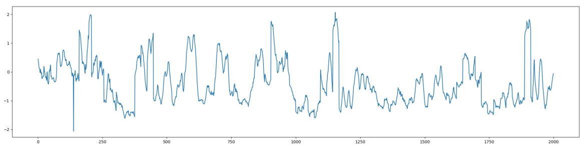
Zhou, T., Ma, Z., Wen, Q., Sun, L., Yao, T., Jin, R., et al. FiLM: Frequency improved Legendre memory model for long-term time series forecasting. In *Neural Information Processing Systems*, 2022a.

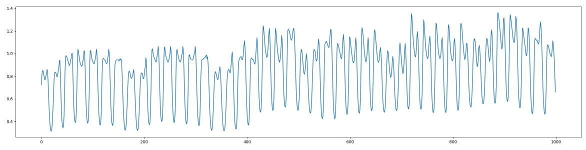
Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., and Jin, R. FEDformer: frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, 2022b.

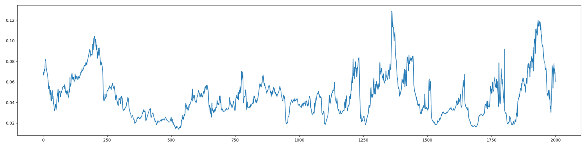
A. Time Series Datasets

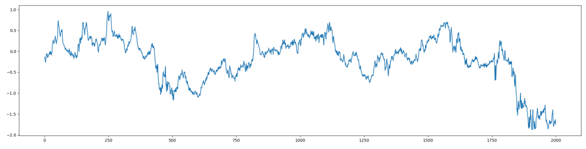
Since different datasets have different sampling interval lengths (see Table 1), using the same set of prediction horizons $\{96, 192, 336, 720\}$ as in (Zhou et al., 2021; Wu et al., 2021) for all datasets may not be appropriate. For example, the *Exchange* dataset contains daily exchange rates. A prediction horizon of 720 corresponds to predicting two years into the future. Instead, we set the prediction horizon H to 14, which corresponds to 2 weeks into the future, which is more reasonable. Similarly, we have $H = 168$ for *ETTh1* (corresponding to 1 week into the future), $H = 720$ for *NorPool* (corresponding to 1 month into the future), and so on as shown in Table 1. Note that some papers also choose the prediction length based on the dataset's sampling frequency. For example, Liu et al. (2021) and Zhang & Yan (2023) also use 168 (instead of 192) for *ETTh1*, *ECL*, and *Traffic*.

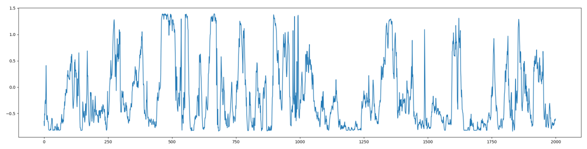
Figure 3 shows examples of the time series data used in the experiments. Since all of them are multivariate, we only show the last variate. As can be seen, these datasets have different temporal dynamics. Moreover, *Caiso*, *Traffic* and *Electricity* show abundant periodic behaviors.

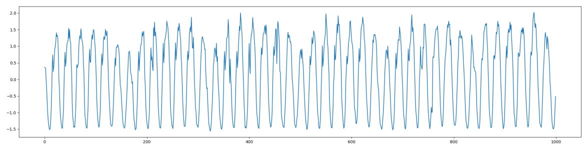

(a) *NorPool*.

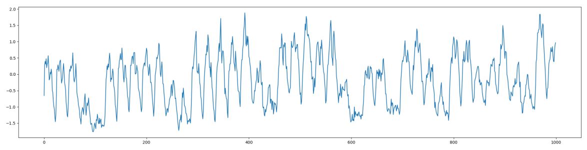

(b) *Caiso*.

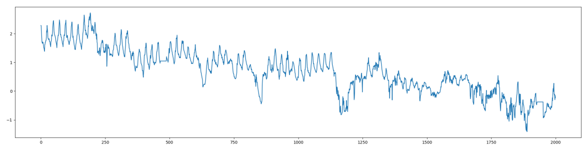

(c) *Weather*.


(d) *ETTm1*.


(e) *Wind*.


(f) *Traffic*.


(g) *Electricity*.


(h) *ETTh1*.

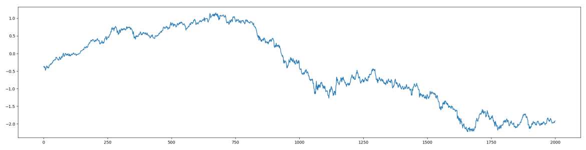

(i) *Exchange*.

Figure 3. Visualization of the time series datasets.

As in (Liu et al., 2022), we use the Augmented Dick-Fuller (ADF) test statistic (Elliott et al., 1996) to evaluate if they are non-stationary. The null hypothesis is that the time series is not stationary (has some time-dependent structure) and can be represented by a unit root. The test statistic results are shown in Table 9. As can be seen, with a threshold of 5%, *Caiso*, *ETTm1*, *ETTh1*, and *Exchange* are considered non-stationary.

Table 9. Evaluation of non-stationarity by the Augmented Dick-Fuller (ADF) test.

	NorPool	*Caiso*	*Weather*	*ETTm1*	*Wind*	*Traffic*	*Electricity*	*ETTh1*	*Exchange*
ADF statistic	-6.862	-2.218	-4.371	-1.734	-6.658	-2.801	-2.797	-2.571	-0.001
p-value	0	0.2	0	0.414	0	0.005	0.006	0.099	0.095

B. Implementation Details

B.1. Network Architecture

The proposed diffusion model has several subnetworks: the conditioning network $\mathcal{F}$, and the denoising network's encoder/decoder. Table 10 shows the subnetwork's input and output sizes, where d is the number of variables, and $d' = d'' = 256$. Each subnetwork is constructed by stacking a number of convolutional blocks. The configuration of each convolutional block is shown in Table 11.

Table 10. Input and output sizes of the subnetworks.

	input size	output size
$\mathcal{F}$	$d \times L$	$d \times H$
encoder	$2d' \times H$	$d'' \times H$
decoder	$(2d + d") \times L$	$d \times H$

Table 11. Configuration of the convolutional block.

layer	operator	default parameters
1	Conv1d	in channel=256, out channel=256, kernel size=3, stride=1, padding=1
2	BatchNorm1d	number of features=256
3	LeakyReLU	negative slope=0.1
4	Dropout	dropout rate=0.1

B.2. Baselines

Code for the baselines are downloaded from the following. (i) TimeGrad: `https://github.com/ForestsKing/TimeGrad`; (ii) CSDI: `https://github.com/ermongroup/CSDI`; (iii) SSSD: `https://github.com/AI4HealthUOL/SSSD`; (iv) D^{3}VAE: `https://github.com/ramber1836/d3vae`; (v) FiLM: `https://github.com/DAMO-DI-ML/NeurIPS2022-FiLM`; (vi) Depts: `https://github.com/weifantt/DEPTS`; (vii) NBeats: `https://github.com/ServiceNow/N-BEATS`; (viii) PatchTST: `https://github.com/yuqinie98/PatchTST/tree/main/PatchTST_self_supervised`; (ix) Fedformer: `https://github.com/DAMO-DI-ML/ICML2022-FEDformer`; (x) Autoformer: `https://github.com/thuml/Autoformer`; (xi) Pyraformer: `https://github.com/ant-research/Pyraformer`; (xii) Informer: `https://github.com/zhouhaoyi/Informer2020`; (xiii) Transformer: `https://github.com/thuml/Autoformer/blob/main/models/Transformer.py`; (xiv) DLinear: `https://github.com/ioannislivieris/DLinear`; (xv) LSTMa: `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`.

C. Using Channel-Independence on Multivariate Time Series Datasets

Recall from Section 4.1 that channel-independence is not used on PatchTST for the experiments in Section 4. In this section, we compare the proposed TimeDiff with PatchTST, FedFormer, Autoformer and Informer under the channel-independence setup. In other words, each variate of the multivariate time series is predicted independently. Table 12 shows the testing MSEs. As can be seen, TimeDiff still outperforms the other baselines most of the time under this setup.

Table 12. Testing MSEs under the channel-independence setup. Results on PatchTST, FedFormer, Autoformer and Informer are from Table 15 in (Nie et al., 2023). The best is in bold, while the second best is underlined.

	ETTh1				*ETTm1*			
	$H = 96$	$H = 192$	$H = 336$	$H = 720$	$H = 96$	$H = 192$	$H = 336$	$H = 720$
TimeDiff	**0.371**	**0.405**	**0.430**	**0.437**	**0.287**	**0.327**	<u>0.368</u>	**0.414**
PatchTST	<u>0.375</u>	<u>0.414</u>	<u>0.431</u>	<u>0.449</u>	<u>0.290</u>	<u>0.332</u>	**0.366**	<u>0.420</u>
FedFormer	0.387	0.439	0.479	0.485	0.408	0.445	0.476	0.533
Autoformer	0.414	0.453	0.496	0.662	0.455	0.598	0.566	0.680
Informer	0.590	0.677	0.710	0.777	0.383	0.42	0.465	0.529