# Training Transformers with Diffusion and Reinforcement Learning

Fall 2024

Arya N

May, 2025

# Diffusion Papers – Background

- **Deep Unsupervised Learning using Nonequilibrium Thermodynamics**, Jascha Sohl-Dickstein, et al. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015
- **Denoising Diffusion Probabilistic Models**. Jonathan Ho, Ajay Jain, Pieter Abbeel. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- **Diffusion Models for Time Series: A Survey**, Alejandro Espejo, et al. *arXiv preprint*, 2023. Available at: `arXiv:2306.12302`.
- **The Rise of Diffusion Models in Time-Series Forecasting**, Caspar Meijer and Lydia Y. Chen. *Delft University of Technology, Technical Report*, January 2023. Available at: `arXiv:2306.05043`.

# Diffusion Papers – Transformers

- **TimeDART: A Diffusion Autoregressive Transformer for Self-Supervised Time Series Representation**. Daoyu Wang, Mingyue Cheng, Zhiding Liu, Qi Liu, Enhong Chen. *arXiv preprint arXiv:2410.05711v4*, University of Science and Technology of China, 2025.
- **TimeDiT: General-Purpose Diffusion Transformers for Time Series Foundation Model**. Defu Cao, Wen Ye, Yizhou Zhang, Yan Liu. *arXiv preprint arXiv:2409.02322v2*, University of Southern California, 2025.
- **Scalable Diffusion Models with Transformers**. William Peebles, Saining Xie. *ICCV*, 2023. UC Berkeley, New York University.

# Diffusion Papers – Other

- **Predict, Refine, Synthesize: Self-Guiding Diffusion Models for Probabilistic Time Series Forecasting**, Marcel Kollovieh, et al. *NeurIPS 2023*. Available at: `arXiv:2307.11494`.
- **Non-autoregressive Conditional Diffusion Models for Time Series Prediction**, Lifeng Shen, James T. Kwok. *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. Available at: `arXiv:2306.12303`.
- **Multi-Resolution Diffusion Models for Time Series Forecasting**. Lifeng Shen, Weiyu Chen, James T. Kwok. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

# RL Papers – Background

# SoTA Training Algorithms – Motivation

- We have noisy power market data that are hard to forecast: load and/or real-time prices.

# SoTA Training Algorithms – Motivation

- We have noisy **[redacted]** data that are hard to forecast: load and/or real-time prices.
- Theoretically, we know that the covariates contribute information about future values of this data, but they have yet to be properly utilized by our models. Sometimes, they even worsen performance.

# SoTA Training Algorithms – Motivation

- We have noisy power market data that are hard to forecast: load and/or real-time prices.
- Theoretically, we know that the covariates contribute information about future values of this data, but they have yet to be properly utilized by our models. Sometimes, they even worsen performance.
- **Can we employ a training algorithm that forces our transformer model to cut through the noise present in the covariates and properly utilize their information?**

# Diffusion – Basic Idea

- A *diffusion model* is a latent variable model that
  - takes tractable, analytic distribution (e.g. standard normal),
  - learns how to transform this distribution back to the original (intractable) data distribution.

# Diffusion – Basic Idea

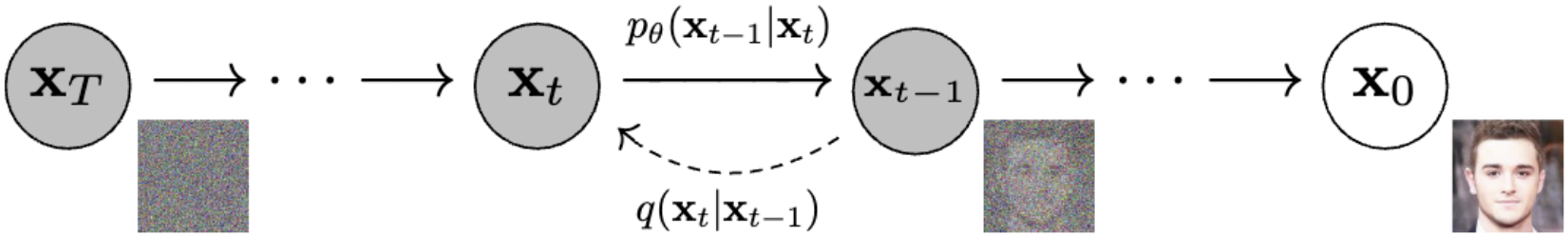
- A *diffusion model* is a latent variable model that
  - takes tractable, analytic distribution (e.g. standard normal),
  - learns how to transform this distribution back to the original (intractable) data distribution.
- **Forward Process:** Markov chain diffusing data distribution into a simpler distribution.
  - The forward process is not learned; it is imposed by us.

# Diffusion – Basic Idea

- A *diffusion model* is a latent variable model that
  - takes tractable, analytic distribution (e.g. standard normal),
  - learns how to transform this distribution back to the original (intractable) data distribution.
- **Forward Process:** Markov chain diffusing data distribution into a simpler distribution.
  - The forward process is not learned; it is imposed by us.
- **Reverse Process:** "Reverse" Markov chain over same states, reverses forward process back to the original data distribution.
  - These transition probabilities are learned by the model.
  - Implicitly learns the data distribution.

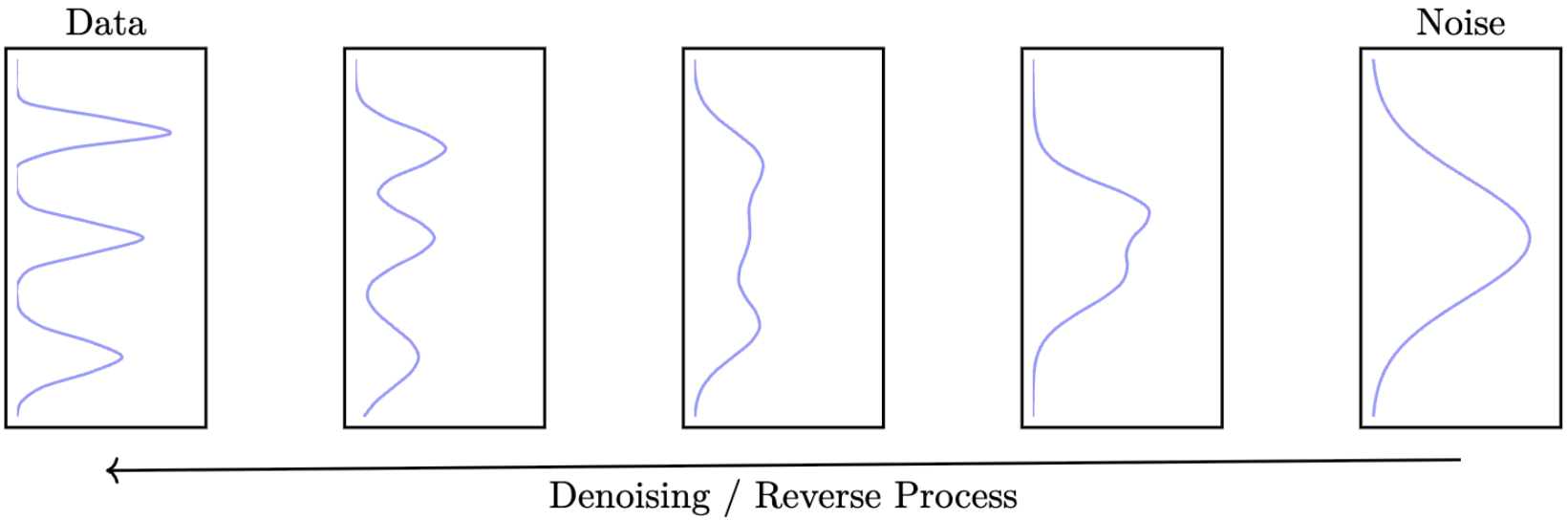
# Diffusion – Basic Idea

- **Originally**: applied ideas from non-equilibrium thermodynamics to generative image modeling.



# Diffusion – Basic Idea

- We will explore work applying this to time series.



# Seminal Paper – Sohl-Dickstein et al., 2015

**Deep Unsupervised Learning using Nonequilibrium Thermodynamics**

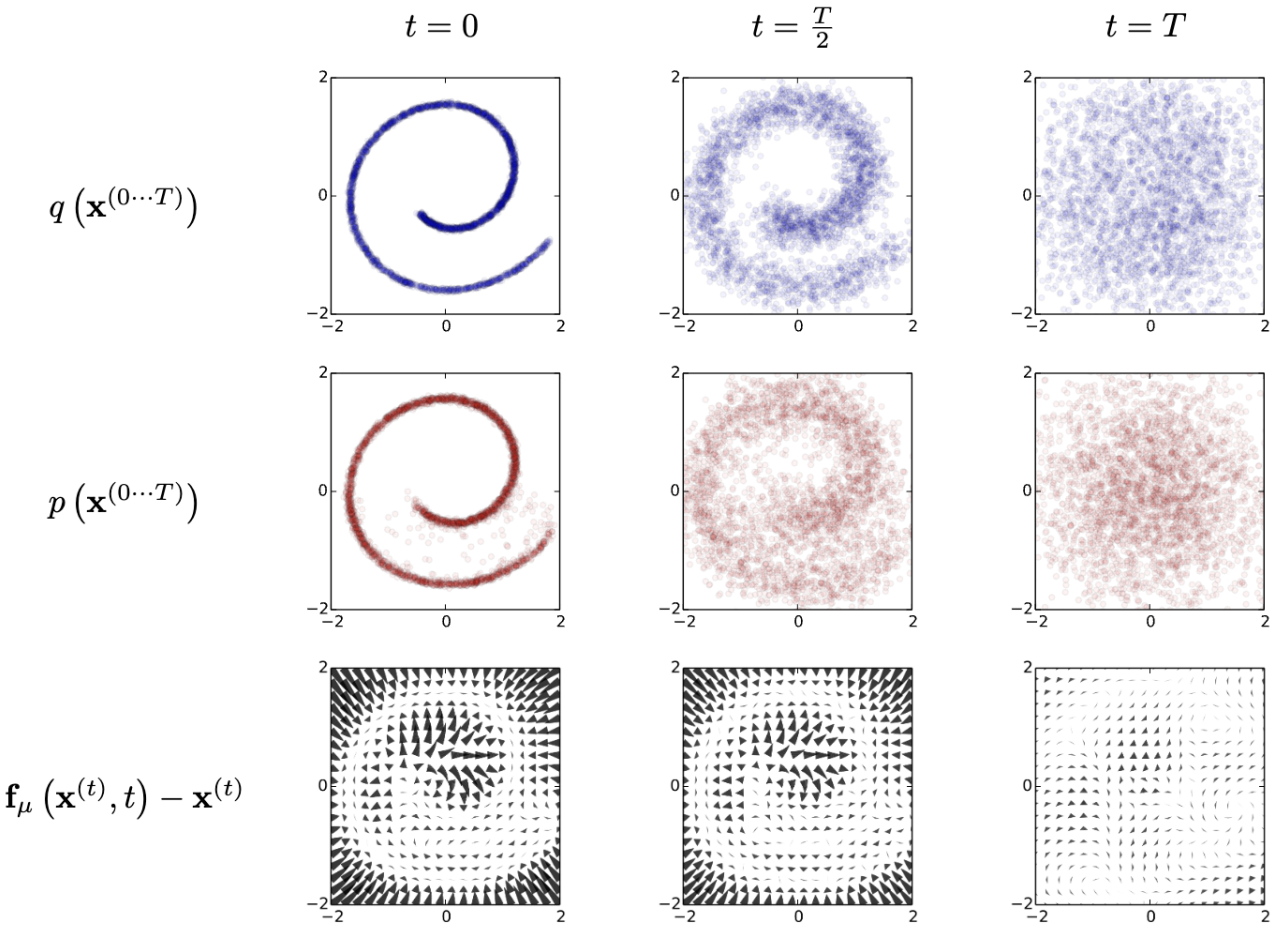

**Jascha Sohl-Dickstein** JASCHA@STANFORD.EDU
Stanford University

**Eric A. Weiss** EAWEISS@BERKELEY.EDU
University of California, Berkeley

**Niru Maheswaranathan** NIRUM@STANFORD.EDU
Stanford University

**Surya Ganguli** SGANGULI@STANFORD.EDU
Stanford University

# Seminal Paper – Sohl-Dickstein et al., 2015




- Authors introduce idea of diffusion process in deep learning.

# Diffusion – Theoretical (Forward Process)

- Suppose we have data conforming to distribution $q(x^{(0)})$.
- The **forward process** transforms this distribution into a simpler distribution $\pi(y)$.
- Sequence of Markov diffusion kernels $T_\pi(y \mid y'; \beta)$, where $\beta$ is the *diffusion rate*. I.e.,

$$\pi(y) = \int T_\pi(y \mid y'; \beta)\pi(y')dy'.$$

# Diffusion – Theoretical (Forward Process)

- Discretized into a $T-$state Markov chain with transition probabilities

$$q(x^{(t)} \mid x^{(t-1)}) = T_\pi(x^{(t)} \mid x^{(t-1)}; \beta_t), \quad 1 \leq t \leq T.$$

- Forward process distribution is

$$q(x^{(0 \cdots T)}) = q(x^{(0)}) \prod_{t=1}^{T} q(x^{(t)} \mid x^{(t-1)}).$$

- Note, this admits "true" *reverse* probabilities in the posteriors: $q(x^{(t-1)} \mid x^{(t)})$.

# Diffusion – Theoretical (Reverse Process)

- Need to construct **reverse process**, *learned* by the model.
- Label $p(x^{(T)}) = \pi(x^{(T)})$.
  - Distribution of forward processes' terminal state.
- Reverse process:

$$p(x^{(0\cdots T)}) = p(x^{(T)}) \prod_{t=1}^{T} p(x^{(t-1)} \mid x^{(t)})$$

for some distribution $p(x \mid x')$.

# Diffusion – Theoretical (Reverse Process)

- It can be shown

$$p(x^{(0)}) = \int p(x^{(0\cdots T)}) dx^{(1\cdots T)}$$

$$= \int dx^{(1\cdots T)} q(x^{(1\cdots T)} \mid x^{(0)}) \prod_{t=1}^{T} \left( \frac{p(x^{(t-1)} \mid x^{(t)})}{q(x^{(t)} \mid x^{(t-1)})} \right) p(x^{(T)}).$$

# Diffusion – Theoretical (Training Objective)

- Model task: learn reverse probabilities $p_\theta(x^{(t-1)} \mid x^{(t)})$ as close to "true" reverse probabilities $q(x^{(t-1)} \mid x^{(t)})$.
- We minimize

$$\mathscr{L} := \sum_{t=1}^{T} D_{KL}\left[q(x^{(t-1)} \mid x^{(t)}, x^{(0)}) \,||\, p_\theta(x^{(t-1)} \mid x^{(t)})\right].$$

# Denoising Diffusion Probabilistic Models

**Jonathan Ho**
UC Berkeley
jonathanho@berkeley.edu

**Ajay Jain**
UC Berkeley
ajayj@berkeley.edu

**Pieter Abbeel**
UC Berkeley
pabbeel@cs.berkeley.edu

# Image Gen – DDPM, Ho et al.

# Image Gen – DDPM, Ho et al.

- We transform the data $x_0$ into standard normal $\mathcal{N}(0, I)$.

# Image Gen – DDPM, Ho et al.

- We transform the data $x_0$ into standard normal $\mathcal{N}(0, I)$.
- The forward transition probability is

$$q(x_t \mid x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I),$$

where $\beta_1, \beta_2, \ldots, \beta_T \in (0,1)$ is the variance schedule.

# Image Gen – DDPM, Ho et al.

- We transform the data $x_0$ into standard normal $\mathcal{N}(0, I)$.
- The forward transition probability is

$$q(x_t \mid x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I),$$

  where $\beta_1, \beta_2, \ldots, \beta_T \in (0,1)$ is the variance schedule.
- Thus, the forward process in closed form is

$$x_t(x_0, \varepsilon) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \mathrm{I}),$$

  where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_1^t \alpha_s$.

# Image Gen – DDPM, Ho et al.

- **Goal:** learn $p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T}$.
  - $p_\theta(x_{0:T})$ is the product over all

$$p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

# Image Gen – DDPM, Ho et al.

- **Goal:** learn $p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T}$.
  - $p_\theta(x_{0:T})$ is the product over all

$$p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

- **How:** For each, $x_t$, have model learn the amount of *noise* added to $x_{t-1}$ to get $x_t$.

# Image Gen – DDPM, Ho et al.

- **Goal:** learn $p_\theta(x_0) = \int p_\theta(x_{0:T}) dx_{1:T}$.
  - $p_\theta(x_{0:T})$ is the product over all

$$p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

- **How:** For each, $x_t$, have model learn the amount of *noise* added to $x_{t-1}$ to get $x_t$.
- Denote noise model $\varepsilon_\theta(x_t, t)$. Properties of Gaussians allow us to approximate the KL-divergence loss with the simple loss function

$$\mathscr{L}_{simple} = \mathbb{E}_{t,x_0,\varepsilon} ||\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||^2.$$

# Image Gen – DDPM, Ho et al.

**Algorithm 1** Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

- The authors use a U-Net architecture for $\varepsilon_\theta$.

# Image Gen – DDPM, Ho et al.

**Algorithm 1** Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

- The authors use a U-Net architecture for $\varepsilon_\theta$.
- Can use **data-prediction** model, $\mu_\theta(x_t, t)$, instead.
  - Backprop on $\nabla_\theta ||\mu - \mu_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||^2$.

- **Sampling:** After $\varepsilon_\theta$ trained, apply transition kernels to $N(0, I)$:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}\varepsilon_\theta(x_t, t)} \right) + \sigma_t z, \qquad z \sim N(0, I).$$

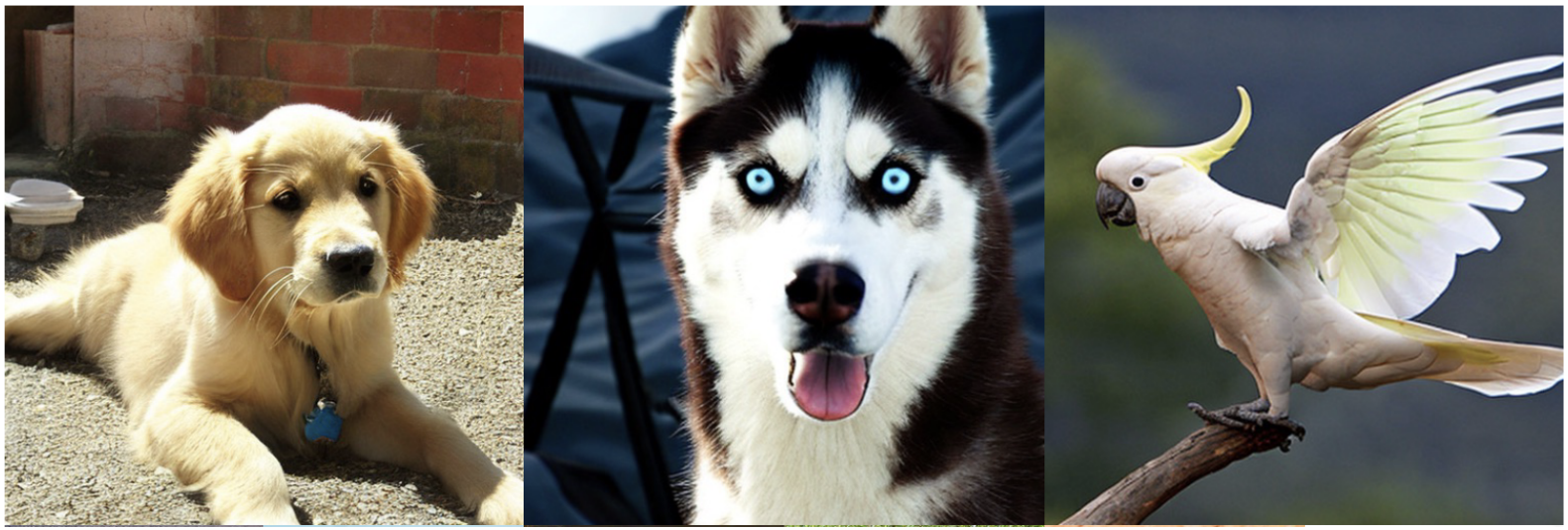
  - $z = 0$ on last step.

# Diffusion + Transformers – DiT, Peebles et al.

**Scalable Diffusion Models with Transformers**

William Peebles*
UC Berkeley

Saining Xie
New York University



# Diffusion + Transformers – DiT, Peebles et al.

- Default backbone for previous diffusion models is convolutional U-Net.

# Diffusion + Transformers – DiT, Peebles et al.

- Default backbone for previous diffusion models is convolutional U-Net.
- Peebles et al. utilize transformer models.

# Diffusion + Transformers – DiT, Peebles et al.

- Default backbone for previous diffusion models is convolutional U-Net.
- Peebles et al. utilize transformer models.
- They introduce a new class of models called Diffusion Transformers (DiTs).
  - Outperforms U-Nets in diffusion tasks.

# Diffusion + Transformers – DiT, Peebles et al.

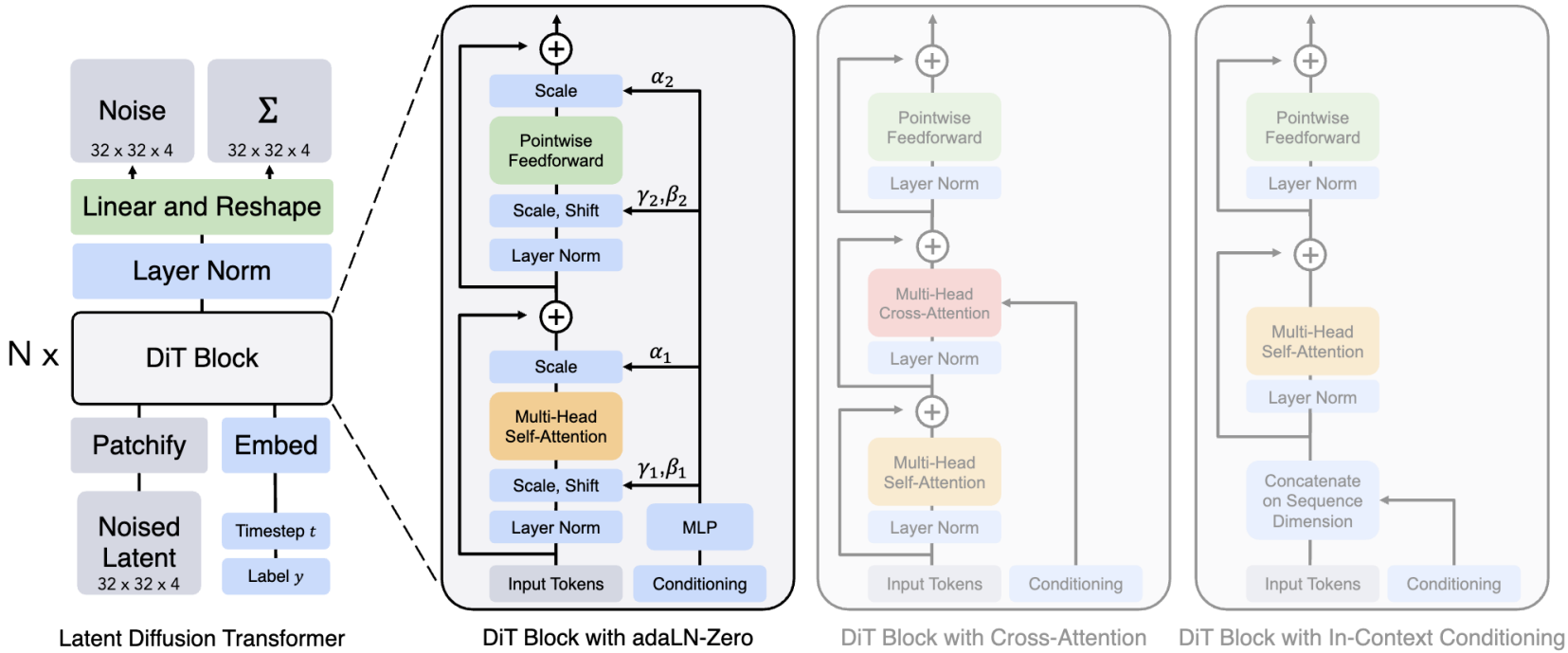



Figure 3. **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

**Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting**

**Kashif Rasul** [1] **Calvin Seward** [1] **Ingmar Schuster** [1] **Roland Vollgraf** [1]

# Diffusion + Time Series – TimeGrad, Rasul et al.

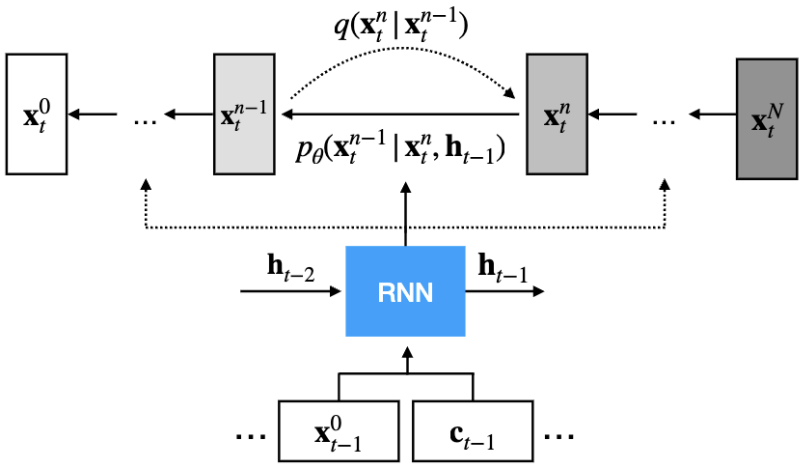
- Adapts Ho et al.'s DDPM model to time series with an autoregressive LSTM network.

# Diffusion + Time Series – TimeGrad, Rasul et al.

- Adapts Ho et al.'s DDPM model to time series with an autoregressive LSTM network.
- The diffusion setup is identical to DDPM's.

# Diffusion + Time Series – TimeGrad, Rasul et al.

- Adapts Ho et al.'s DDPM model to time series with an autoregressive LSTM network.
- The diffusion setup is identical to DDPM's.




**Algorithm 1** Training for each time series step $t \in [t_0, T]$

**Input:** data $\mathbf{x}_t^0 \sim q_{\mathcal{X}}(\mathbf{x}_t^0)$ and state $\mathbf{h}_{t-1}$

**repeat**

Initialize $n \sim \text{Uniform}(1, \dots, N)$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Take gradient step on

$$\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_n}\mathbf{x}_t^0 + \sqrt{1-\bar{\alpha}_n}\epsilon, \mathbf{h}_{t-1}, n)\|^2$$

**until** converged

- $h_t = RNN_\theta(concat(x_t^0, c_t), h_{t-1})$, $c_t$ denotes covariates.
  - Encodes time steps at current diffusion step.
- $\varepsilon_\theta()$ predicts the noise based on noised sample and timestep encoding.

## Architecture of noise model $\varepsilon_\theta$:

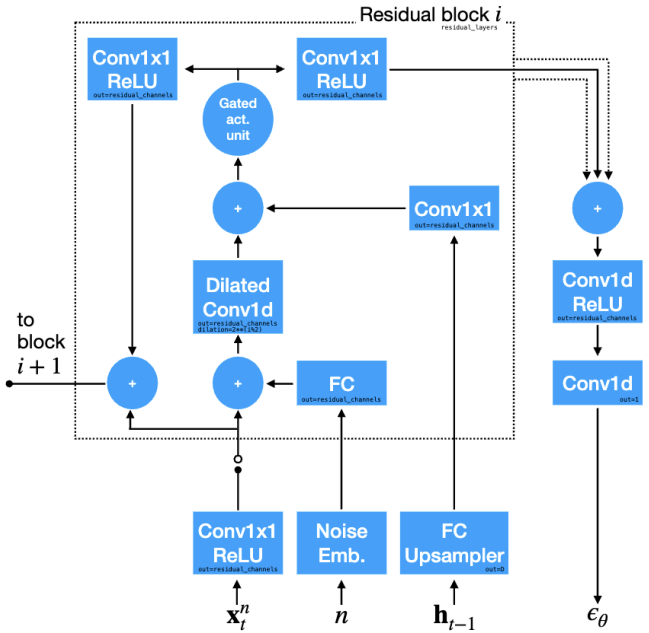



*Figure 2.* The network architecture of $\epsilon_\theta$ consisting of `residual_layers` $= 8$ conditional residual blocks with the Gated Activation Unit $\sigma(\cdot) \odot \tanh(\cdot)$ from (van den Oord et al., 2016b); whose skip-connection outputs are summed up to compute the final output. `Conv1x1` and `Conv1d` are 1D convolutional layers with filter size of 1 and 3, respectively, circular padding so that the spatial size remains $D$, and all but the last convolutional layer has output channels `residual_channels` $= 8$. `FC` are linear layers used to up/down-sample the input to the appropriate size for broadcasting.

**Algorithm 2** Sampling $\mathbf{x}_t^0$ via annealed Langevin dynamics

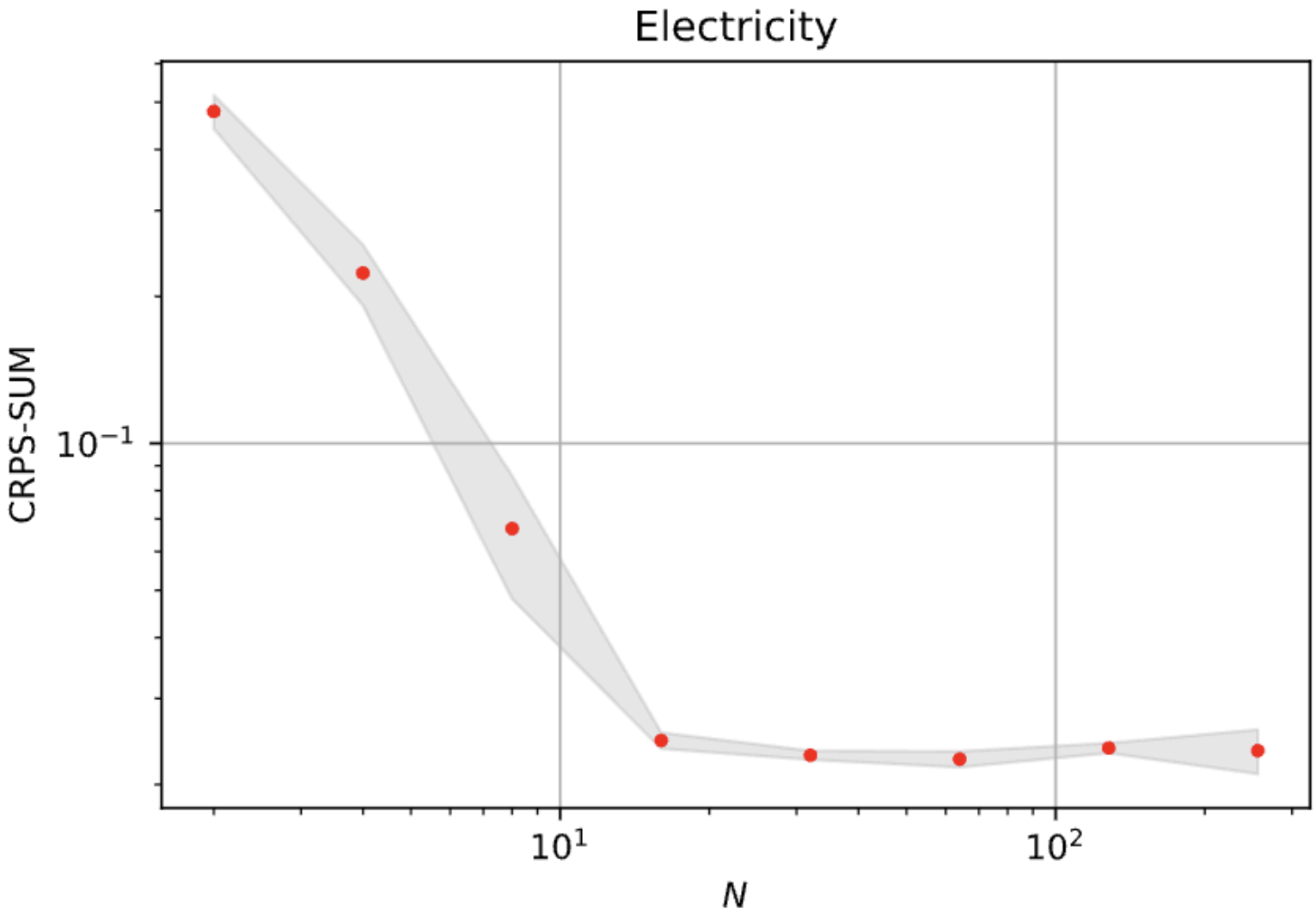
**Input:** noise $\mathbf{x}_t^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and state $\mathbf{h}_{t-1}$
**for** $n = N$ **to** $1$ **do**
  **if** $n > 1$ **then**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  **else**
    $\mathbf{z} = \mathbf{0}$
  **end if**
  $\mathbf{x}_t^{n-1} = \frac{1}{\sqrt{\alpha_n}}(\mathbf{x}_t^n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta(\mathbf{x}_t^n, \mathbf{h}_{t-1}, n)) + \sqrt{\Sigma_\theta}\mathbf{z}$
**end for**
**Return:** $\mathbf{x}_t^0$

## Paper results:

| Method | Exchange | Solar | Electricity | Traffic | Taxi | Wikipedia |
|---|---|---|---|---|---|---|
| VES | **0.005**±0.000 | 0.9±0.003 | 0.88±0.0035 | 0.35±0.0023 | - | - |
| VAR | **0.005**±0.000 | 0.83±0.006 | 0.039±0.0005 | 0.29±0.005 | - | - |
| VAR-Lasso | 0.012±0.0002 | 0.51±0.006 | 0.025±0.0002 | 0.15±0.002 | - | 3.1±0.004 |
| GARCH | 0.023±0.000 | 0.88±0.002 | 0.19±0.001 | 0.37±0.0016 | - | - |
| KVAE | 0.014±0.002 | 0.34±0.025 | 0.051±0.019 | 0.1±0.005 | - | 0.095±0.012 |
| Vec-LSTM ind-scaling | 0.008±0.001 | 0.391±0.017 | 0.025±0.001 | 0.087±0.041 | 0.506±0.005 | 0.133±0.002 |
| Vec-LSTM lowrank-Copula | 0.007±0.000 | 0.319±0.011 | 0.064±0.008 | 0.103±0.006 | 0.326±0.007 | 0.241±0.033 |
| GP scaling | 0.009±0.000 | 0.368±0.012 | 0.022±0.000 | 0.079±0.000 | 0.183±0.395 | 1.483±1.034 |
| GP Copula | 0.007±0.000 | 0.337±0.024 | 0.0245±0.002 | 0.078±0.002 | 0.208±0.183 | 0.086±0.004 |
| Transformer MAF | **0.005**±0.003 | 0.301±0.014 | 0.0207±0.000 | 0.056±0.001 | 0.179±0.002 | 0.063±0.003 |
| **TimeGrad** | 0.006±0.001 | **0.287**±0.02 | **0.0206**±0.001 | **0.044**±0.006 | **0.114**±0.02 | **0.0485**±0.002 |

# Diffusion + Time Series – TimeGrad, Rasul et al.

**Increasing diffusion steps improves performance:**



**Non-autoregressive Conditional Diffusion Models for Time Series Prediction**

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

# Diffusion + Time Series – TimeDiff, Shen et al.

- Diffusion process identical to DDPM, but only applied to *forecast windows*, not inputs.

# Diffusion + Time Series – TimeDiff, Shen et al.

- Diffusion process identical to DDPM, but only applied to *forecast windows*, not inputs.
- The model input is a *conditioned signal*: $c = concat[z_{mix}, z_{AR}]$
  - $z_{mix} = m_k \odot F(x_{past}) + (1 - m_k) \odot x^0_{1:H}$, $m_k \sim Uniform(0,1)$, $F(x_{past})$ projects $x_{past}$ to same shape as $x^0_{1:H}$. ($m_k = 1$ during inference.)
  - $z_{AR} = \sum_{i=-L+1}^{0} W_i \odot X^0_i + B$.

# Diffusion + Time Series – TimeDiff, Shen et al.

- Diffusion process identical to DDPM, but only applied to *forecast windows*, not inputs.
- The model input is a *conditioned signal*: $c = concat[z_{mix}, z_{AR}]$
  - $z_{mix} = m_k \odot F(x_{past}) + (1 - m_k) \odot x^0_{1:H}$, $m_k \sim Uniform(0, 1)$, $F(x_{past})$ projects $x_{past}$ to same shape as $x^0_{1:H}$. ($m_k = 1$ during inference.)
  - $z_{AR} = \sum_{i=-L+1}^{0} W_i \odot X^0_i + B$.
- Uses a *data-prediction* model (rather than noise-prediction) $x_\theta(x^k_{1:H}, k \mid c)$.

# Diffusion + Time Series – TimeDiff, Shen et al.

**Algorithm 1** Training.

**Require:** Number of diffusion steps $K$; pretrained AR model $\mathcal{M}_{ar}$.

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

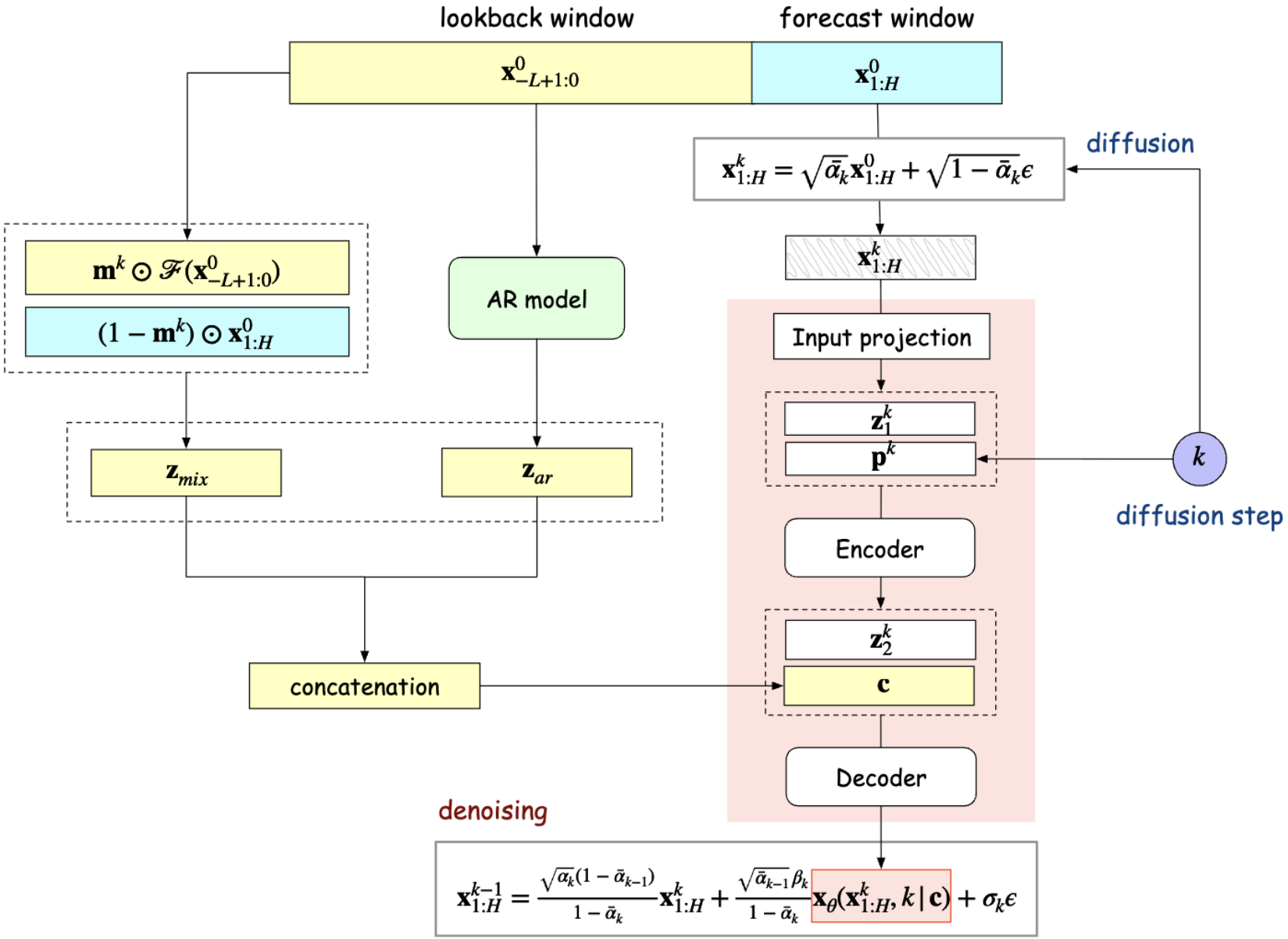
**Algorithm 2** Inference.

**Require:** Trained denoising network $\mathbf{x}_\theta$; trained conditioning network $\mathcal{F}$; pretrained AR model $\mathcal{M}_{ar}$.

$\mathbf{x}^K_{1:H} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$;
**for** $k = K, \ldots, 1$ **do**
  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, if $k > 1$, else $\epsilon = 0$;
  obtain diffusion step $k$'s embedding $\mathbf{p}^k$ using (17);
  obtain $\mathbf{z}_{\texttt{mix}}$ by (15);
  obtain $\mathbf{z}_{ar}$ by (16);
  obtain condition $\mathbf{c}$ based on $\mathbf{z}_{\texttt{mix}}$ and $\mathbf{z}_{ar}$ using (13);
  sample $\hat{\mathbf{x}}^{k-1}_{1:H}$ by (18);
  $\mathbf{x}^{k-1}_{1:H} = \hat{\mathbf{x}}^{k-1}_{1:H}$;
**end for**
**return** $\hat{\mathbf{x}}^0_{1:H}$.

- **Training Obj:** $\mathscr{L}_k(\theta) = ||x^0_{1:H} - x_\theta(x^k_{1:H}, k \mid c)||^2$.
- **Inference samples:** $\hat{x}^{k-1}_{1:H} = \frac{\alpha_k(1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k} x^k_{1:H} + \frac{\bar{\alpha}_{k-1}\beta_k}{1-\bar{\alpha}_k} x_\theta(\cdot) + \sigma_k \varepsilon$

# Diffusion + Time Series – TimeDiff, Shen et al.



# Diffusion + Time Series – TimeDiff, Shen et al.

*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 |

# Diffusion + Time Series – TimeDiff, Shen et al.

*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 |

# Diffusion + Transformers + Time Series – TDSTF, Chang et al. (2024)

A Transformer-based Diffusion Probabilistic Model for Heart Rate and Blood Pressure Forecasting in Intensive Care Unit

Ping Chang[1], Huayu Li[1], Stuart F. Quan[2,3], Shuyang Lu [4,5], Shu-Fen Wung [6,7], Janet Roveda[1,6,8], Ao Li[1,6]

# Diffusion + Transformers + Time Series – TDSTF, Chang et al.

# Diffusion + Transformers + Time Series – TimeDART, Wang et al. (2025)

**TimeDART: A Diffusion Autoregressive Transformer for Self-Supervised Time Series Representation**

**Daoyu Wang** [1] **Mingyue Cheng** [1] **Zhiding Liu** [1] **Qi Liu** [1] **Enhong Chen** [1]

# Mathematical Properties – I-MMSE Relations, Kong et al.

# Mathematical Properties – Information Decomposition, Kong et al.

# Areas for Contribution

- Incorporate known properties of load/real-time prices into conditional diffusion.
- Combining methods from these papers + non-autoregressive transformer.
- **My idea:** develop diffusion pre-training method for non-autoregressive transformers, combine with *dynamic ensemble learning*.

# Evaluations – Encoder-Only Noise Prediction

The following results are from pre-training our encoder-only transformer with diffusion as a *data-prediction* model. We employ the *Min-SNR-$\gamma$-weighted* MSE loss (Hang et al.):

$$\mathscr{L} := \mathbb{E}_{t\sim\mathscr{U}(1,T),x_0\sim p_{data},\varepsilon\sim\mathscr{N}(0,I)}\left[w(t)||x_0 - \hat{x}_{0,\theta}(x_t,t)||_2^2\right],$$

where

$$w(t) = \frac{\min\{SNR(t),\gamma\}}{SNR(t)}, \quad SNR(t) = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t}, \quad \bar{\alpha}_t = \prod_{1\le s\le t}\alpha_s.$$

The rest of the setting are as in Ho et al. The model is then fine-tuned on the same task as the other benchmark models.