NARRATIVE OF TIME ACROSS SCALES (NOTS)

Wenrui Ma[1,*], **Ran Liu**[2,*], **Ellen L Zippi**[2], **Christopher M Sandino**[2], **Juri Minxha**[2], **Behrooz Mahasseni**[2], **Erdrin Azemi**[2], **Ali Moin**[2], **Eva L Dyer**[1]
[1]University of Pennsylvania, [2]Apple, *Equal contribution

ABSTRACT

Time series inherently capture dynamic processes across multiple scales, yet standard self-supervised methods typically operate at a single scale by segmenting data into discrete patches. This disrupts temporal continuity and neglects the hierarchical interactions that define the signal, limiting adaptability across diverse tasks. To address this, we propose NoTS, a pre-training framework that learns cross-scale relationships by reconstructing fine-scale signals from progressively degraded coarse approximations. Theoretically, we show that this multi-scale sequence modeling enhances representational capacity compared to single-scale patch approaches. Empirically, NoTS achieves a 26% improvement in synthetic feature regression and outperforms existing methods by up to 6% across 22 real-world datasets in classification, imputation, and anomaly detection. Moreover, NoTS consistently boosts the performance of existing transformer backbones, establishing it as a theoretically grounded foundation for time series analysis.

Track: Research

1 INTRODUCTION

Time series naturally exhibit multi-scale patterns where fast fluctuations occur atop slower trends, necessitating representations that capture these cross-scale interactions. However, most self-supervised objectives for time series focus on a single scale, typically utilizing next-window prediction or masked reconstruction where inputs are segmented into fixed-length patches (Garza & Mergenthaler-Canseco, 2023; Dong et al., 2024). This patching introduces artificial boundaries that disrupt temporal continuity and weaken the connection between coarse context and local variation (Yu et al., 2025). Consequently, models often miss long-range structures like periodicity (Zhou et al., 2022) and suffer from implicit biases injected by the choice of patch length (Yu et al., 2025).

To address this, we propose NoTS, a pre-training framework that explicitly models cross-scale interactions. Inspired by resolution-based tasks in vision (Tian et al., 2024), we construct a resolution hierarchy via progressive smoothing degradation and train a transformer to reconstruct fine-scale signals from coarser approximations. From this process emerges a **N**arrative **o**f **T**ime across **S**cales (**NoTS**).

We support this approach with theoretical analysis in App. A.1.1, demonstrating that multi-scale sequence modeling increases representational capacity under the universal approximation framework (Yun et al., 2019). Empirically, NoTS demonstrates consistent gains across diverse tasks (classification, imputation, anomaly detection) and multiple model architectures. It improves performance by 26% on synthetic benchmarks and up to 6% across 22 real-world datasets compared to single-scale baselines, while consistently boosting existing backbones (Nie et al., 2022; Liu et al., 2023b).

2 METHODS

2.1 COARSE-TO-FINE NEXT-SCALE PREDICTION

Given $\mathbf{S} \in \mathbb{R}^{C \times T}$, we construct K ordered degraded views $\mathbf{S}_k = d_k(\mathbf{S})$, where the operators $\{d_k\}_{k=1}^{K}$ are arranged from coarse to fine. In particular, d_1 applies the strongest degradation and produces the coarsest view, while d_K applies the weakest degradation and produces the view closest to the raw signal. We then model this cross-scale sequence autoregressively:

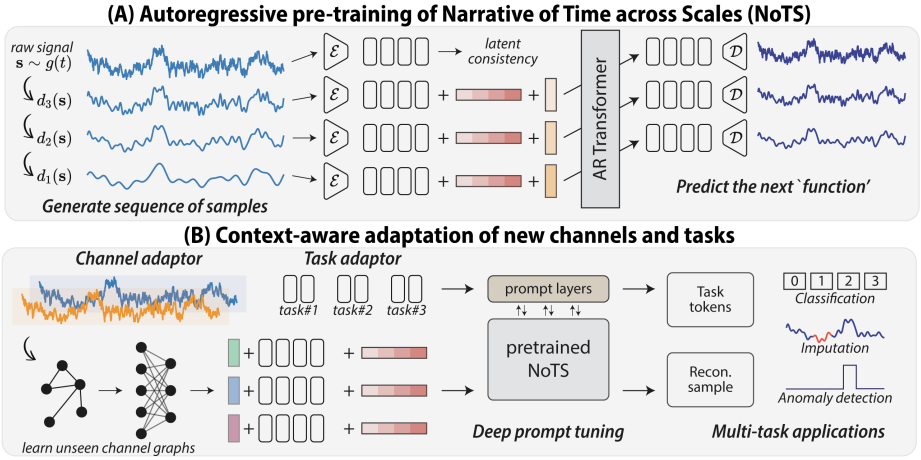

Figure 1: *Narrative of Time across Scales (NoTS).* (A) Pre-training: We generate a hierarchy of simplified signals which, combined with position and resolution embeddings, are fed into an autoregressive transformer that reconstructs the next finer resolution, regularized by a latent consistency loss. (B) Adaptation: For real-world dataset deployment, we use an MLP-based channel adaptor to handle unseen channels and a task adaptor utilizing prompt tokens (Jia et al., 2022), enabling the model to generalize across diverse downstream applications.

$$\mathbf{S}_k = d_k(\mathbf{S}), \quad k = 1, \ldots, K, \qquad p(\mathbf{S}_1, \ldots, \mathbf{S}_K) = p(\mathbf{S}_1) \prod_{k=1}^{K-1} p(\mathbf{S}_{k+1} \mid \mathbf{S}_1, \ldots, \mathbf{S}_k)\,. \tag{1}$$

This formulation encourages the model to learn cross-scale dependencies by progressively refining coarse temporal context into finer-scale detail. A theoretical perspective is deferred to Appendix A.1.1.

2.2 Degradation Operators

Each degradation operator $d_k(\cdot)$ is implemented as a 1D convolution along the temporal dimension,

$$\mathbf{S}_k = d_k(\mathbf{S}) = (\mathbf{S} * w_k)[n], \tag{2}$$

where w_k is a smoothing kernel whose width controls the degree of degradation. We consider two families of kernels: **Local smoothing (time-domain):** A uniform averaging kernel of length p_k, where $w_k[n] = 1/p_k$ for $-0.5p_k \le n \le 0.5p_k$ and 0 elsewhere; **Global smoothing (frequency-domain):** A low-pass filter implemented via a sinc kernel, $w_k[n] = \mathrm{sinc}(p_k n)$, where p_k controls the cutoff frequency. The parameters $\{p_k\}$ are chosen such that earlier views (smaller k) correspond to stronger smoothing. Mixing both kernel types produces a degradation sequence that spans both local smoothness and global bandwidth reduction.

2.3 Tokenizer and Autoregressive Transformer Across Scales

Next scale modeling of tokens Processing entire views directly incurs substantial computation and memory cost for long series. We therefore tokenize each scale $\mathbf{S}_k$ using an encoder–decoder pair, $\mathbf{R}_k = \mathcal{E}(\mathbf{S}_k)$ and $\hat{\mathbf{S}}_k = \mathcal{D}(\mathbf{R}'_k)$, where $\mathbf{R}_k$ is a group of latent tokens representing scale k, and $\mathbf{R}'_k$ denotes predicted tokens. Token groups are concatenated in scale order and processed by a transformer with a *group-wise causal mask* that ensures each finer-scale token group can attend only to previous coarser-scale groups:

$$[\mathbf{R}'_2, \ldots, \mathbf{R}'_K] = \mathrm{Transformer}([\mathbf{R}_1, \ldots, \mathbf{R}_{K-1}]), \qquad \mathrm{mask}[\Omega_k] = \begin{cases} 0, & \bigcup_{m=1}^{k} \Omega_m, \\ -\infty, & \text{otherwise}, \end{cases} \tag{3}$$

where Ω_k indexes token positions belonging to group $\mathbf{R}_k$.

NoTS-lw While $\mathcal{E}$ and $\mathcal{D}$ can be any encoder/decoder architectures, we implement a lightweight model NoTS-lw with a simple channel-independent 1D-ResNet encoder/decoder block to preserve the integrity in the token space. We also report results with different encoder/decoder architectures in Table 2.

Table 1: Feature approximation results on synthetic datasets. Results are averaged over three runs and scaled by 100 for readability. Lower is better.

	Fractional Brownian motion (fBm)			*Autocorrelated sinusoids*		
Regression (↓)	$\mathcal{H}$-index (1D)	SSC (32D)	WAMP (32D)	SSC (32D)	WAMP (32D)	b. power (96D)
VQVAE	3.78 ± 0.45	38.93 ± 0.70	65.77 ± 3.72	26.24 ± 0.61	29.13 ± 0.90	14.37 ± 0.03
MAE	2.01 ± 0.61	25.78 ± 0.11	26.34 ± 0.03	25.29 ± 0.31	28.81 ± 2.86	14.90 ± 0.02
FAMAE	1.99 ± 0.24	33.85 ± 0.53	45.76 ± 0.24	28.26 ± 0.57	24.82 ± 0.84	13.92 ± 0.02
Next-period pred.	1.75 ± 0.11	27.38 ± 0.12	26.66 ± 0.19	24.44 ± 0.11	28.97 ± 1.37	13.96 ± 0.04
NoTS (Ours)	1.27 ± 0.16	23.78 ± 0.34	20.04 ± 0.12	23.13 ± 0.19	24.58 ± 0.48	13.62 ± 0.05
Improvement	↑ **37.80**%	↑ **8.41**%	↑ **31.44**%	↑ **5.66**%	↑ **0.98**%	↑ **2.20**%

Positional embeddings To encode structural information, we employ three types of embeddings: (i) rotary positional embeddings within each token group (Su et al., 2024), (ii) a learnable degradation embedding that indicates the scale index k, and (iii) optional learnable channel embeddings when using channel-independent tokenizers.

Training Objective The model is trained using an autoregressive next-scale reconstruction loss combined with a latent consistency regularizer:

$$\mathcal{L} = \sum_{k=1}^{K-1} \mathcal{L}_{\text{recon}}(\hat{\mathbf{S}}_{k+1}, \mathbf{S}_{k+1}) + \lambda \, \mathcal{L}_{\text{recon}}(\mathcal{D}(\mathcal{E}(\mathbf{S}_K)), \mathbf{S}_K), \tag{4}$$

where $\mathcal{L}_{\text{recon}}$ is the mean absolute error (MAE) and λ is a weighting hyperparameter. The second term ensures that the tokenizer remains well-behaved for the finest scale $\mathbf{S}_K$, which is not directly fed into the autoregressive transformer during pretraining.

2.4 Downstream task Deployment via Context-Aware Adaptation

When adapting to downstream tasks, we use lightweight modules to handle potential distribution shifts in channel dimensions and temporal lengths: (i) **Channel adaptor:** a linear layer that transforms input channels, optionally supplemented with dataset-specific additive channel tokens; (ii) **Task adaptor:** learnable prompt tokens inserted into the transformer layers, following deep prompt tuning (Jia et al., 2022), together with a task-specific output head (e.g., linear classifier or regressor). This adaptation strategy introduces fewer than 1% new parameters, enabling efficient transfer while preserving the cross-scale representations learned during pretraining. We refer readers to Appendix C for more technical details.

3 Experimental Results

3.1 Synthetic experiments: Feature regression

Setup. We generate two datasets of length 1024 to probe long-range and multi-frequency structure: (i) 20k **fractional Brownian motion (fBm)** signals (Dieker & Mandjes, 2003) with varying Hurst index $\mathcal{H}$, and (ii) 50k **superimposed autocorrelated sinusoids** (Yoon et al., 2019). Details are provided in Appx. B.1. We assess representation quality by regressing four signal features from the learned embeddings: Slope Sign Change (SSC), Willison Amplitude (WAMP), the $\mathcal{H}$-index, and band power (details in App. B.1). Following Section 2.4, we train a VQVAE (Van Den Oord et al., 2017), masked autoencoder (MAE) (Dong et al., 2024), frequency-aware MAE (FAMAE) (Liu et al., 2023a), next-period prediction transformer (Garza & Mergenthaler-Canseco, 2023), and **NoTS-lw** on the synthetic datasets by appending them with a regression task adaptor to test our proposed method on the feature regression task.

Results. As shown in Table 1, across the board, **NoTS-lw** significantly outperforms all other pretraining methods given the same architecture and training pipeline. The relative improvement is especially pronounced on the fBm dataset, whose complex covariance structure is relevant to many real-world applications. In this setting, NoTS achieves a 26% improvement.

Table 2: Comparisons between NoTS and other pre-training methods on real-world datasets. We categorize the results based on (a) if adaptors are used, and (b) if the weights of the pre-trained models are frozen. We compute an average error rate (↓) to compare the final performance of different methods in each condition.

			Classification (↑)		*Anom. Det.* (↑)				*Imputation* (↓)				**Avg.** (↓)
Methods	(a)	(b)	UCR-9	UEA-5	SMD	MSL	SWaT	PSM	ETTm1	ETTm2	ETTh1	ETTh2	**error rate**
SimMTM	✓	✓	68.70	55.36	**84.06**	83.90	91.20	96.07	**0.164**	0.126	0.264	0.183	19.43
bioFAME	✓	✓	62.63	60.32	83.09	**84.28**	91.21	95.94	0.203	**0.122**	**0.258**	**0.178**	19.87
Next-pred	✓	✓	65.95	58.30	82.96	83.75	90.47	**96.54**	0.306	0.178	0.465	0.270	24.39
NoTS-lw (Ours)	✓	✓	**71.88**	**62.78**	83.63	**84.28**	**93.26**	96.27	**0.164**	0.126	0.286	0.196	**18.51**
SimMTM	✓	✗	81.65	61.23	83.48	84.11	91.35	96.36	0.123	**0.107**	**0.201**	0.166	16.14
bioFAME	✓	✗	81.53	63.57	83.59	83.98	**91.46**	**96.88**	0.129	**0.107**	0.202	0.178	16.05
Next-pred	✓	✗	80.62	62.76	83.00	84.09	91.00	96.87	0.130	0.119	0.228	0.188	16.82
NoTS-lw (Ours)	✓	✗	**88.08**	**66.38**	**84.19**	**84.15**	91.26	**96.88**	**0.122**	0.116	0.218	**0.163**	**15.10**
PatchTST	✗	✗	83.57	63.31	78.96	78.81	83.75	78.07	0.181	0.126	0.347	0.187	21.78
+NoTS (Ours)	✗	✗	**↑1.71**	**↑1.65**	**↑2.20**	**↑3.96**	**↑5.97**	**↑11.25**	**↓.003**	**↓.003**	**↓.064**	**↓.006**	**18.33**
iTransformer	✗	✗	82.67	67.62	85.18	83.04	91.88	97.07	0.162	0.111	0.240	0.168	16.07
+NoTS (Ours)	✗	✗	**↑1.26**	**↑0.65**	**↑0.17**	**↑0.11**	**↑0.05**	**↑0.01**	↑.005	**↓.002**	**↓.013**	**↓.004**	**15.70**

3.2 Real-world experiments

Benchmarks We follow the multi-task protocol of Wu et al. (2022): classification on selected UCR (Dau et al., 2019) and UEA (Bagnall et al., 2018) subsets, imputation on ETDataset (Zhou et al., 2021), and anomaly detection on MSL (Hundman et al., 2018), PSM (Abdulaal et al., 2021), SWaT (Mathur & Tippenhauer, 2016), and SMD (Su et al., 2019). We use the preprocessing and deployment in Wu et al. (2022), except evaluating a harder *channel-wise* imputation setting. We refer the readers to Appx. B.2.2, B.2.3 for more details.

Setups We evaluate NoTS in two complementary settings: (i) **Pre-training comparison:** pre-train NoTS-lw and baselines on the synthetic datasets, then adapt to downstream datasets with the same prompt-tuning pipeline (with or without freezing the backbone); (ii) **Plug-in benefit:** apply NoTS pre-training on top of PatchTST (Nie et al., 2022) and iTransformer (Liu et al., 2023b) and measure the performance benefits from adding NoTS.

Results Table 2 shows that NoTS-lw outperforms other pre-training methods across tasks and tuning regimes, improving the average performance by up to 6% under matched pipelines. When used as an add-on pre-training strategy, NoTS provides consistent gains on top of PatchTST and iTransformer, indicating that the learned cross-scale structure is complementary to backbone design. Notably, in the frozen-backbone regime (first 4 rows of Table 2), we train $<1\%$ parameters yet reach 82% average performance, highlighting parameter-efficient transfer ability of NoTS.

3.3 Ablation and Analysis

Var.	orig.	AR	conn.	noise	err.
(1)	×	✓	✓	×	1.75
(2)	✓	×	✓	×	1.48
(3)	✓	×	×	×	1.82
(4)	✓	✓	✓	✓	1.69
NoTS-lw	✓	✓	✓	×	**1.27**

Effective components. We ablate the key design choices of NoTS-lw on the synthetic feature regression task (H-index). Variant (1) removes the latent consistency term by excluding the original signal from training; variant (2) removes autoregressive masking, allowing the transformer to bridge degraded views without an ordered coarse-to-fine dependency; variant (3) further removes explicit connections among degraded views, treating them as independent augmentations; and variant (4) replaces our convolution-based degradation with stochastic Gaussian noise. Full NoTS-lw achieves the best performance overall. Removing latent consistency degrades performance substantially, consistent with the intuition that the tokenizer otherwise never receives a direct training signal on the finest-scale view. Removing the autoregressive structure or the explicit cross-view construction also hurts performance, showing that ordered next-scale prediction is more effective than treating degraded views as unordered augmentations.

Connection to diffusion-style degradation. The Gaussian-noise variant underperforms the smoothing-based degradation operators used in NoTS. A plausible explanation is that many real-world time series are already noisy, so adding random noise is less informative than removing

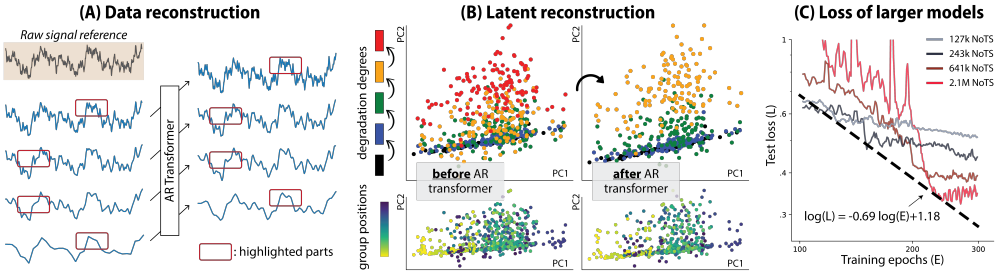

Figure 2: *Visualizations of AR performance and loss.* (A) We visualize the next scale inference process of NoTS on the synthetic dataset. From bottom to top, the signal variance is gradually recovered through the next scale transformer. (B) The token space is visualized through PCA, where tokens of the simplified signals gradually disperse to a larger region when colored in different degradation degrees. When colored with relative group positions, the distribution does not shift as much on the direction of another principal component. (C) A pilot study shows that training larger NoTS models leads to lower reconstruction loss on the test set, potentially following the power law behavior of AR models.

information in a structured coarse-to-fine manner. We therefore view diffusion-style degradation as an interesting but currently weaker alternative in our setting, and leave deeper connections to deterministic/cold diffusion and related generative formulations for future work.

Additional analysis. Figure 2 provides qualitative evidence for the learned coarse-to-fine dynamics: predicted signals progressively recover variance across scales, predicted latent groups follow the target latent organization, and larger models achieve lower reconstruction loss. Additional examples and extended visualizations are deferred to Fig. 3 and Fig. 4.

4 Conclusion

In this paper, we propose NoTS, a pre-training method based on next-scale reconstruction that enables hierarchical, cross-scale representation learning. Experiments across 2 synthetic and 22 real-world benchmarks show that NoTS consistently outperforms existing methods, establishing it as a viable foundation model for time series. Our work builds upon existing pre-training and multi-scale learning methods for time series as discussed in Appx. D. Future works may extend the existing results through scaling to larger datasets, more tasks like forecasting, and exploring theoretical connections to diffusion models (Dieleman, 2024).

References

Ahmed Abdulaal, Zhuanghua Liu, and Tomer Lancewicki. Practical approach to asynchronous multivariate time series anomaly detection and localization. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp. 2485–2494, 2021.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

Vassilis Assimakopoulos and Konstantinos Nikolopoulos. The theta model: a decomposition approach to forecasting. *International journal of forecasting*, 16(4):521–530, 2000.

Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.

Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. Nhits: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 6989–6997, 2023.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. *arXiv preprint arXiv:2310.10688*, 2023.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305, 2019.

Antonius Bernardus Dieker and Michael Mandjes. On spectral simulation of fractional brownian motion. *Probability in the Engineering and Informational Sciences*, 17(3):417–434, 2003.

Sander Dieleman. Diffusion is spectral autoregression, 2024. URL `https://sander.ai/2024/09/02/spectral-autoregression.html`.

Jiaxiang Dong, Haixu Wu, Haoran Zhang, Li Zhang, Jianmin Wang, and Mingsheng Long. Simmtm: A simple pre-training framework for masked time-series modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

Benjamin L Edelman, Surbhi Goel, Sham Kakade, and Cyril Zhang. Inductive biases and variable creation in self-attention mechanisms. In *International Conference on Machine Learning*, pp. 5793–5831. PMLR, 2022.

Cheng Gao, Yuan Cao, Zihao Li, Yihan He, Mengdi Wang, Han Liu, Jason M Klusowski, and Jianqing Fan. Global convergence in training large-scale transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

Azul Garza and Max Mergenthaler-Canseco. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023.

Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

Qihe Huang, Lei Shen, Ruixin Zhang, Shouhong Ding, Binwu Wang, Zhengyang Zhou, and Yang Wang. Crossgnn: Confronting noisy multivariate time series via cross interaction refinement. *Advances in Neural Information Processing Systems*, 36:46885–46902, 2023.

Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 387–395, 2018.

Rob J Hyndman and Yeasmin Khandakar. Automatic time series forecasting: the forecast package for r. *Journal of statistical software*, 27:1–22, 2008.

Vugar E Ismailov. A three layer neural network can represent any multivariate function. *Journal of Mathematical Analysis and Applications*, 523(1):127096, 2023.

Aysu Ismayilova and Vugar Ismailov. On the kolmogorov neural networks. *arXiv preprint arXiv:2311.00049*, 2023.

Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European Conference on Computer Vision*, pp. 709–727. Springer, 2022.

Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pp. 2306–2327. PMLR, 2020.

Ran Liu, Mehdi Azabou, Max Dabagia, Chi-Heng Lin, Mohammad Gheshlaghi Azar, Keith Hengen, Michal Valko, and Eva Dyer. Drop, swap, and generate: A self-supervised approach for generating neural activity. *Advances in neural information processing systems*, 34:10587–10599, 2021.

Ran Liu, Ellen L Zippi, Hadi Pouransari, Chris Sandino, Jingping Nie, Hanlin Goh, Erdrin Azemi, and Ali Moin. Frequency-aware masked autoencoders for multimodal pretraining on biosignals. *arXiv preprint arXiv:2309.05927*, 2023a.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023b.

Shengjie Luo, Shanda Li, Shuxin Zheng, Tie-Yan Liu, Liwei Wang, and Di He. Your transformer may not be as powerful as you expect. *Advances in Neural Information Processing Systems*, 35: 4301–4315, 2022.

Aditya P Mathur and Nils Ole Tippenhauer. Swat: A water treatment testbed for research and training on ics security. In *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*, pp. 31–36. IEEE, 2016.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv preprint arXiv:1905.10437*, 2019.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2828–2837, 2019.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *arXiv preprint arXiv:2404.02905*, 2024.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Y Zhang, and Jun Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. *arXiv preprint arXiv:2405.14616*, 2024.

Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Cost: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. *arXiv preprint arXiv:2202.01575*, 2022.

Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. Unified training of universal time series forecasting transformers. *arXiv preprint arXiv:2402.02592*, 2024.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The eleventh international conference on learning representations*, 2022.

Yongtao Wu, Fanghui Liu, Grigorios Chrysos, and Volkan Cevher. On the convergence of encoder-only shallow transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

Annan Yu, Danielle C Maddix, Boran Han, Xiyuan Zhang, Abdul Fatir Ansari, Oleksandr Shchur, Christos Faloutsos, Andrew Gordon Wilson, Michael W Mahoney, and Yuyang Wang. Understanding the implicit biases of design choices for time series foundation models. *arXiv preprint arXiv:2510.19236*, 2025.

Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 8980–8987, 2022.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

Xiang Zhang, Ziyuan Zhao, Theodoros Tsiligkaridis, and Marinka Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. *Advances in Neural Information Processing Systems*, 35:3988–4003, 2022.

Shubao Zhao, Ming Jin, Zhaoxiang Hou, Chengyi Yang, Zengxiang Li, Qingsong Wen, and Yi Wang. Himtm: Hierarchical multi-scale masked time series modeling for long-term forecasting. *arXiv preprint arXiv:2401.05012*, 2024.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International conference on machine learning*, pp. 27268–27286. PMLR, 2022.

APPENDIX

A ANALYTICAL RESULTS

A.1 PRELIMINARIES

Transformers We consider the standard transformer architecture as defined in (Luo et al., 2022). The transformer network is the stack of transformer blocks, each of them consists of a self-attention layer $\mathrm{Attn}(\cdot)$ and a feed forward layer $\mathrm{FF}(\cdot)$. Given an input $\mathbf{X} \in \mathbb{R}^{d\times T}$, they are written as:

$$\begin{aligned} \mathrm{Attn}(\boldsymbol{X}) &= \boldsymbol{X} + \sum_{i=1}^{h} \boldsymbol{W}_O^i \boldsymbol{W}_V^i \boldsymbol{X} \cdot \sigma\left[\left(\boldsymbol{W}_K^i \boldsymbol{X}\right)^T \boldsymbol{W}_Q^i \boldsymbol{X}\right] \\ \mathrm{FF}(\boldsymbol{X}) &= \mathrm{Attn}(\boldsymbol{X}) + \boldsymbol{W}_2 \cdot \mathrm{ReLU}\left(\boldsymbol{W}_1 \cdot \mathrm{Attn}(\boldsymbol{X})\right) \end{aligned} \tag{5}$$

where $\boldsymbol{W}_O^i \in \mathbb{R}^{d\times m}$, $\boldsymbol{W}_V^i, \boldsymbol{W}_K^i, \boldsymbol{W}_Q^i \in \mathbb{R}^{m\times d}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d\times r}$, and $\boldsymbol{W}_1 \in \mathbb{R}^{r\times d}$.

We denote $t^{h,m,r} : \mathbb{R}^{d\times T} \to \mathbb{R}^{d\times T}$ as a transformer block with an attention layer with h heads of size m, and a feed-forward layer with r hidden nodes. Thus, the transformer can be written as:

$$\mathcal{T}^{h,m,r} := \left\{ f : \mathbb{R}^{d\times T} \to \mathbb{R}^{d\times T} \mid f \text{ is a composition of transformer blocks } t^{h,m,r} \right\}. \tag{6}$$

Similarly, transformer with absolute positional embedding is:

$$\mathcal{T}_{\mathrm{P}}^{h,m,r} := \left\{ f_{\mathrm{P}}(\boldsymbol{X}) = f(\boldsymbol{X} + \boldsymbol{E}) \mid f \in \mathcal{T}^{h,m,r} \text{ and } \boldsymbol{E} \in \mathbb{R}^{d\times T} \right\} \tag{7}$$

Universal Approximator (UA) The universal approximation framework considers the feasibility or existence of a neural network that can approximate different types of functions with arbitrarily small error. Consider a transformer network f_1 and an arbitrary function f_2, where $f_1, f_2 : \mathbb{R}^{n\times T} \to \mathbb{R}^{n\times T}$ are both sequence-to-sequence functions. We define a distance between f_1 and f_2 as:

$$\mathrm{d}_p\left(f_1, f_2\right) := \left(\int \left\| f_1(\boldsymbol{X}) - f_2(\boldsymbol{X}) \right\|_p^p d\boldsymbol{X} \right)^{1/p} \tag{8}$$

being a UA means that for any given $f_2 \in \mathcal{F}$, let $1 \le p < \infty$ and $\epsilon > 0$, there exists a network f_1, such that $\mathrm{d}_p(f_1, f_2) \le \epsilon$. Several prior works have explored the concept of universal approximators (UAs) and whether transformers qualify as UAs. Below, we outline the key results from the literature that will be referenced in this paper:

Theorem 2 (informal, see Yun et al. (2019)). Given $1 \le p < \infty$ and $\epsilon > 0$, for any function $f \in \mathcal{F}_{\mathrm{PE}}$, where $\mathcal{F}_{\mathrm{PE}}$ consists of all continuous permutation equivariant functions with compact support, there exists a Transformer network $f \in \mathcal{T}^{2,1,4}$ where $\mathrm{d}_p(f, g) \le \epsilon$.

Theorem 3 (informal, see Yun et al. (2019)). Given $1 \le p < \infty$ and $\epsilon > 0$, for any function $f \in \mathcal{F}_{\mathrm{CD}}$, where $\mathcal{F}_{\mathrm{CD}}$ consists of all continuous functions with compact support, there exists a Transformer network $f \in \mathcal{T}_{\mathrm{P}}^{2,1,4}$ where $\mathrm{d}_p(f, g) \le \epsilon$.

Theorem 2 discussed that transformers without positional embeddings are UAs for all continuous permutation equivariant functions; and Theorem 3 discussed that transformers with absolute positional embeddings (APE) are UAs for all continuous functions with compact support. Note that the latter results may be overruled by modifying the transformer architectures as follows:

Theorem 4 (informal, see Luo et al. (2022)). Consider relative positional encoding (RPE) that modifies the attention as $\mathrm{Attn}(\boldsymbol{X}) = \boldsymbol{X} + \sum_{i=1}^{h} \boldsymbol{W}_O^i \boldsymbol{W}_V^i \boldsymbol{X} \cdot \mathrm{softmax}\left[\left(\boldsymbol{W}_K^i \boldsymbol{X}\right)^T \boldsymbol{W}_Q^i \boldsymbol{X} + \mathbf{E}\right]$, where $\mathbf{E} \in \mathbb{R}^{T\times T}$ encodes the relative position within attention maps. Given $T > 2$, there always exists a continuous function $f_M : \mathcal{D} \to \mathbb{R}^{d\times T}$, such that $\sup_{\boldsymbol{X}\in\mathcal{D}} \left\| f_{\mathrm{RPE}}(\boldsymbol{X}) - f_M(\boldsymbol{X}) \right\|_2^2 \ge M$ holds for any modified RPE-based transformer.

While UA framework is typically used for understanding the approximation problem towards continuous functions, more recently, it is used to understand the approximation problem towards discontinuous functions. Specifically, in this work, we reference the results in Ismailov (2023) that

shows any discontinuous function may be implemented by a three-layer Kolmogorov type neural network:

Theorem 5 (informal, see Ismailov (2023)). Given $d \geq 2$ and any function $f : \mathbb{I}^d \to \mathbb{R}$, where $\mathbb{I}$ is a closed unit internal $[0, 1]$, then function f can be implemented exactly by a three-layer Kolmogorov neural network with d, $2d+1$, and 1 processing units in the first, second, and final layer, respectively.

As stated in Ismayilova & Ismailov (2023); Ismailov (2023), the expressiveness of simple neural networks can be extended by constructing more diverse activation functions, which helps us understand the approximation ability towards discontinuous functions which are more prevalent in real-world complex systems. Note that other works also discuss the approximation problems towards functions that may be discontinuous (Kidger & Lyons, 2020; Pinkus, 1999).

A.1.1 An intuitive example: Approximating functions

To justify the construction of functional sequences, we provide an intuitive example to investigate the expressive power of transformers under the context of time series domain. Following Yun et al. (2019); Luo et al. (2022), we consider the standard transformer architectures:

$$\mathcal{T}^{h,m,r} := \left\{ \begin{array}{l} f : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n} \\ \quad f \text{ consists of Transformer blocks } t^{h,m,r} \end{array} \right\} \tag{9}$$

where $t^{h,m,r}$ consists of one self-attention layer of h heads of size m and one feed-forward layer with r hidden dimensions (see definitions in Appx. Eq. 5). To combat permutation equivariance, we add absolute positional embedding[1] to the transformer that creates $\mathcal{T}_{\mathrm{P}}^{h,m,r} := \{f_{\mathrm{P}}(\boldsymbol{X}) = f(\boldsymbol{X} + \boldsymbol{E})\}$ where $f \in \mathcal{T}^{h,m,r}$ and $\boldsymbol{X}, \boldsymbol{E} \in \mathbb{R}^{d\times n}$. Detailed in Appendix A, our analysis is an extension of previous results in Yun et al. (2019); Ismailov (2023).

A.1.2 Time series in the function space

This work assumes time series data as functions in time $g(t)$, which forms function space $\mathcal{F}_g(\mathbb{R})$. An operator on the function space maps the original function $g(t)$ to a target function $h(t)$, creating a mapping across function spaces $A : \mathcal{F}_g(\mathbb{R}) \to \mathcal{F}_h(\mathbb{R})$. Assume that the signal is produced with a sampling plan $\{t_i\}_{i=1}^T$, the sampling operation on top of the functions discretizes the mapping into a set of output $\{A[g(t_i)]\}_{i=1}^T$, which forms a sequence-to-sequence function $f_{(A)} : \mathbb{R}^{d\times T} \to \mathbb{R}^{d\times T}$. When breaking time series into concatenations of time periods, $\mathbf{S}$ is directly treated as inputs to the transformer $\mathbf{X}$, where one aims to find a transformer network $f_P \in \mathcal{T}_P^{h,m,r}$ to approximate $f_{(A)}$.

Example: The differential operator It is intuitive that one can easily construct a linear but discontinuous mapping A, which is not necessarily approximable by transformers. See below:

Theorem 1. Given $T > 2$, and $\mathcal{D} \subseteq \mathbb{R}^{d\times T}$. Consider the differential operator A that forms a sequence-to-sequence function $f_{(A)}$ under sampling plans $\{t_i\}_{i=1}^T$ with its initial starting point $t_1 \in \mathbb{R}$ and fixed intervals. There exists a $\mathbf{X} \in \mathbb{R}^{d\times T}$, such that:

$$\sup_{\boldsymbol{X}\in\mathcal{D}} \left\| f_P(\boldsymbol{X}) - f_{(A)}(\boldsymbol{X}) \right\|_2^2 \geq T \tag{10}$$

holds for any transformer network $f_P \in \mathcal{T}_P^{h,m,r}$.

Proof. We construct a negative example with $d = 1$. Consider a set of input functions $g_M(t) = \sin(Mt)/M$, the target functions under the differential operator are $h_M(t) = \cos(Mt)$. As M increases, the input function converges uniformly to a constant zero function, which gives a sampled input matrix $\boldsymbol{X} \to \mathbf{0} \in \mathcal{D}$. At limit, the studied transformer network $f_P(\boldsymbol{X})$ converges to a fixed matrix $\mathbf{B}$ (see Appendix A.2). Thus, given a sampling plan of interval $t_{i+1} - t_i = \pi/M$ and two initial starting points $t_1^{(1)} = 0$ and $t_1^{(2)} = \pi$, we form $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ that give:

$$\lim_{M\to\infty} \Big(\left\| f_P(\boldsymbol{X}_1) - f_{(A,M)}(\boldsymbol{X}_1) \right\| + \left\| f_P(\boldsymbol{X}_2) - f_{(A,M)}(\boldsymbol{X}_2) \right\| \Big) \geq \sum_{i=1}^{T} 2 = 2T, \tag{11}$$

where $f_{(A,M)}$ denotes the function formed from $\{A[g_M(t_i)]\}_{i=1}^T$, which leads to Eq. 10.

[1] Refer to Luo et al. (2022) for a case study of relative positional embedding under the UA framework.

A.1.3 TWO SUFFICIENT CONDITIONS TO APPROXIMATE THE DIFFERENTIAL OPERATOR

When considering signals as functions in time, sampling from simple signal processing operators may create discontinuous sequence-to-sequence functions, causing approximation issues of transformer if one directly considers $\mathbf{S}$ as inputs $\mathbf{X}$ to the transformer. Instead, by constructing signals sequences of length T using $\mathbf{S}_k = d_k(\mathbf{S})$, and performing dimensionality reduction with an encoder $\mathcal{E}$, we create two sufficient conditions to address the approximation issue as follows:

Proposition 1. Given a signal $\mathbf{S} \in \mathbb{R}^{d\times T}$ and an encoder $\mathcal{E} : \mathbb{R}^{d\times T} \to \mathbb{R}^d$, there exists two sufficient conditions to approximate $\{A[g(t_i)]\}_{i=1}^T$ with the construction of $\mathbf{X} = [\mathcal{E}(\mathbf{S}_1), \mathcal{E}(\mathbf{S}_2), ..., \mathcal{E}(\mathbf{S}_T)]$:

- The constructed $\mathbf{S}_i$ is expressive such that there exists a continuous mapping between a fixed element of $\mathbf{S}_i$ and the i-th element of the target output $A[g(t_i)]$;
- Given any distinguishable $\mathbf{S}_i$, there exists an expressive tokenizer $\mathcal{E}$ that preprocess $\mathbf{S}_i$ to create a continuous mapping between $\mathcal{E}(\mathbf{S}_i)$ to the target.

Proof and examples. See Appx. A.3 for proof and an example solution for differential operator.

A.2 ADDITIONAL PROOF OF THEOREM 1

We only study the convergence property of self attention layers as the convergence property of feed forward networks has been extensively studied in previous works. To prove convergence, we build an input sequence $\mathbf{X} + \Delta_n$, where Δ_n is defined as a bounded perturbation matrix $\Delta \in \mathcal{D}$ that is uniformly scaled by a positive value n. Given a self attention layer $\mathrm{Attn}(\mathbf{X})$, we show the following:

Lemma 1. Given $n \geq N$, $\mathbf{X} \in \mathcal{D}$, $\Delta_n = \Delta/n$, there exists an ϵ such that:

$$\sup_{\Delta\in\mathcal{D},\|\Delta\|_1\leq 1} \|\,\mathrm{Attn}(\mathbf{X}) - \mathrm{Attn}(\mathbf{X} + \Delta_n)\|_2 < \epsilon. \tag{12}$$

holds for any self attention layer parameterized by $\boldsymbol{W}_O^i \in \mathbb{R}^{d\times m}$, $\boldsymbol{W}_V^i, \boldsymbol{W}_K^i, \boldsymbol{W}_Q^i \in \mathbb{R}^{m\times d}$.

Proof. First, we re-write the activation component $\sigma\left[\left(\boldsymbol{W}_K^i\boldsymbol{X}\right)^\top \boldsymbol{W}_Q^i\boldsymbol{X}\right]$ in Eq. 5 as column-wise softmax operation $\mathrm{softmax}(\mathbf{X}^\top\mathbf{W}\mathbf{x}_j)$, where $\mathbf{x}_j$ is a random column of $\mathbf{X}$. We have:

$$\begin{aligned}
&\|\,\mathrm{softmax}(\mathbf{X}^\top\mathbf{W}\mathbf{x}_j) - \mathrm{softmax}((\mathbf{X}+\Delta_n)^\top\mathbf{W}(\mathbf{x}_j+\delta_{j,n}))\|_2 \\
&\leq \|\,\mathrm{softmax}(\mathbf{X}^\top\mathbf{W}\mathbf{x}_j) - \mathrm{softmax}((\mathbf{X}+\Delta_n)^\top\mathbf{W}(\mathbf{x}_j+\delta_{j,n}))\|_1 \\
&\leq 2\|\mathbf{X}^\top\mathbf{W}\mathbf{x}_j - (\mathbf{X}+\Delta_n)^\top\mathbf{W}(\mathbf{x}_j+\delta_{j,n})\|_\infty \\
&\quad \text{(Corollary A.7 in Edelman et al. (2022))} \\
&= 2\max_i\left(\frac{1}{n}\left(\mathbf{x}_i^\top\mathbf{W}\delta_j + \delta_i^\top\mathbf{W}\mathbf{x}_j\right) + \frac{1}{n^2}\delta_i^\top\mathbf{W}\delta_j\right) \\
&\leq 2\max_i\left(\frac{1}{n}\left(\mathbf{x}_i^\top\mathbf{W}\mathbb{1} + \mathbb{1}^\top\mathbf{W}\mathbf{x}_j\right) + \frac{1}{n^2}\mathbb{1}^\top\mathbf{W}\mathbb{1}\right) \\
&= \epsilon_h.
\end{aligned} \tag{13}$$

which shows that the attention map converges given deterministic $\mathbf{X}$ and $\mathbf{W} = (\mathbf{W}_K^i)^\top\mathbf{W}_Q^i$. Thus, Eq. 12 holds by considering the self attention operator $\mathrm{Attn}(\mathbf{X})$ as a convex combination of attention heads given deterministic $\mathbf{X}$, $\boldsymbol{W}_O^i \in \mathbb{R}^{d\times m}$, $\boldsymbol{W}_V^i \in \mathbb{R}^{m\times d}$.

Thus, given $\boldsymbol{X} \to \mathbf{0}$, the transformer network $f_P(\mathbf{X})$ converges to a deterministic matrix $\mathbf{B}$ that is dependent on the network parameters and the positional embedding $\mathbf{E}$. Note that, under the context of network optimization, a more generalized version of the convergence property of transformers has been proved in other previous works (Wu et al., 2024; Gao et al., 2024).

A.3 ADDITIONAL PROOF OF PROPOSITION 1

Based on the results in Theorem 3, it is known that as long as the constructed sequence $\mathbf{X} = [\mathcal{E}(\mathbf{S}_1), \mathcal{E}(\mathbf{S}_2), ..., \mathcal{E}(\mathbf{S}_T)]$ forms a continuous sequence-to-sequence function between input $\mathbf{X}$ and

the target $\{A[g(t_i)]\}_{i=1}^T$, it is guaranteed that there exists a transformer network $f_P \in \mathcal{T}_P^{2,1,4}$ that can approximate the constructed sequence-to-sequence function. Thus, we show how the presented two conditions are sufficient to meet the above requirement:

- When there exists a continuous mapping between a fixed element p of $\mathbf{S}_i$ and the i-th element of the target output $A[g(t_i)]$, one can construct a simple linear encoder $\mathcal{E}(\mathbf{S}) = \mathbf{S}\mathbf{v}$, where $\mathbf{v}[i] = 0$ when $i \neq p$ and $\mathbf{v}[p] = 1$, that creates the continuous sequence-to-sequence function.
- Based on the results in Theorem 5, if there exists an expressive tokenizer $\mathcal{E}$ (that may be a discontinuous function) that preprocess $\mathbf{S}_i$ to create a continuous mapping between $\mathcal{E}(\mathbf{S}_i)$ to the target, the existence of the transformer is guaranteed for a continuous sequence-to-sequence function.

Example solutions We provide example solutions to approximate the differential operator. Under the first condition, a trivial solution can be constructed by performing phase transition with degradation operator, creating $(d_i \circ g)(t) = \sin(Mt + \frac{i\pi}{2T})/M$. While the first condition requires data-specific degradation operators, the second condition provides more flexibility. In the case of differential operator, we rely on the result from Ismailov (2023) and use a Kolmogorov's mapping three-layer neural network to approximate an arbitrary (continuous or discontinuous) function $f : \mathbb{I}^T \to \mathbb{R}$, where $\mathbb{I}$ is a compact interval $[0, 1]$. Thus, one can construct a simple degradation operator of value shifts as: $(d_i \circ g)(t) = \sin(Mt)/M + (1 + i\delta/T)/M$, where $\delta \in (0, M - 2]$ is an arbitrary number that distinguishes $\mathbf{S}_i$. In this case, there exists an encoder $\mathcal{E}$ that can create a continuous sequence-to-sequence function to the desired target (e.g., $\mathcal{E}(\mathbf{S}_i) = Mt_i$), where the existence of a solution is guaranteed by previous results in Yun et al. (2019).

B Experimental details

B.1 Synthetic experiments

B.1.1 Synthetic datasets details

Fractional Brownian motion (fBm) Given a Hurst index $\mathcal{H}$ and two time steps i and j, a fBm process is a continuous-time Gaussian process $B_{\mathcal{H}}(t)$ with the following covariance structure:

$$E\left[B_{\mathcal{H}}(i)B_{\mathcal{H}}(j)\right] = \frac{1}{2}\left(|i|^{2\mathcal{H}} + |j|^{2\mathcal{H}} - |i-j|^{2\mathcal{H}}\right) \tag{14}$$

Define function $\gamma(i, \mathcal{H}) = 0.5(|i-1|^{2\mathcal{H}} + |i+1|^{2\mathcal{H}} - 2|i|^{2\mathcal{H}})$, a fBm process can be simulated through the Cholesky decomposition method detailed as follows:

Algorithm 1 Simulation of fBm processes using the Cholesky's method

Inputs: N as the length of sequence (time steps), $\mathcal{H} \in (0, 1)$ as the Hurst index
Initialize: $\mathbf{L} \in \mathbb{R}^{N\times N}$, $\mathbf{V} \in \mathbb{R}^N$ with each entry randomly sampled from $\mathcal{N}(0, 1)$
Define: $\mathbf{X} \in \mathbb{R}^N$ as the output vector Initial conditions for $\mathbf{L}$: $\mathbf{L}[0, 0] = 1$, $\mathbf{L}[1, 0] = 2^{2\mathcal{H}-1} - 1$, $\mathbf{L}[1, 1] = (1 - \mathbf{L}[1, 0]^2)^{1/2}$, Initial conditions for $\mathbf{X}$: $\mathbf{X}[0] = \mathbf{V}[0]$, $\mathbf{X}[1] = \mathbf{L}[1, 0]\mathbf{V}[0] + \mathbf{L}[1, 1]\mathbf{V}[1]$
for each time step i from 2 till $N - 1$ **do** $\mathbf{L}[i, 0] = \gamma(i, \mathcal{H})$
    for each time step j from 1 to $i - 1$ **do**

$$\mathbf{L}[i, j] = \frac{1}{\mathbf{L}[j, j]}\left(\gamma(i - j, \mathcal{H}) - \sum_{k=0}^{j-1} \mathbf{L}[i, k] \cdot \mathbf{L}[j, k]\right)$$

    end for Update $\mathbf{L}[i, i] = (1 - \sum_{k=0}^{i-1}(\mathbf{L}[i, k]^2))^{1/2}$, $X[i] = \sum_{k=0}^{i} \mathbf{L}[i, k]\mathbf{V}[k]$
end for
for each time step i from $N - 1$ till 0 **do** $\mathbf{X}[i] = (\sum_{k=0}^{i} \mathbf{X}[k]) \times N^{-\mathcal{H}}$
end for
Output: A simulated fBm process $\mathbf{X}$

Autocorrelated sinusoids The autocorrelated sinusoids dataset is generated with AR processes in the frequency space. Given an integer k, a randomly initialized set of weights $\{\phi_i\}_{i=1}^k$, an AR(k)

process defines the sequence of frequency values as follows:

$$f_t = \sum_{i=1}^{k} \phi_i f_{t-i} \tag{15}$$

The AR process ensures that the frequency components in the synthetic dataset are correlated, creating an autoregressive frequency structure that NoTS can effectively learn from.

B.1.2 FEATURE REGRESSION TASK DETAILS

We detail the feature extraction methods as follows. We use them as the ground truth for the feature regression task. Define the indicator function $\mathbf{1}_A(x)$, where $\mathbf{1}_A(x) = 1$ if $x \in A$ and $\mathbf{1}_A(x) = 0$ otherwise. Given a single-channel signal $\mathbf{s} \in \mathbb{R}^T$ with v_i as the value on i-th timestamp, all features are extracted on each channel of the signal as follows:

Slope Sign Change (SSC) SSC measures directional slope changes in a signal, indicating the intensity of fluctuations. Given a threshold value δ as hyperparameter, a period of time series sequence $\mathbf{s}$, we extract SSC value with the following equation:

$$\begin{aligned}\mathrm{SSC}(\mathbf{s}) = \sum_{i=2}^{T-1} &\mathbf{1}_{(v_i - v_{i-1})(v_i - v_{i+1}) < 0}\left(v_i\right) \cdot \\ &\mathbf{1}_{\max(|v_i - v_{i+1}|, |v_i - v_{i-1}|) \geq \delta}\left(v_i\right).\end{aligned} \tag{16}$$

In practice, we extract the SSC values on top of segmented signals with a length of 32.

Willison Amplitude (WAMP) WAMP is a similar feature that focuses on counting significant amplitude changes between consecutive steps. Given a threshold value δ as hyperparameter, a period of time series sequence $\mathbf{s}$, WAMP is computed through $\mathrm{WAMP}(\mathbf{s}) = \sum_{i=1}^{T-1} \mathbf{1}_{|v_{i+1} - v_i| \geq \delta}\left(v_i\right)$. In practice, we also extract the WAMP values on top of segmented signals with a length of 32, creating a 32-dimensional feature for each studied synthetic data sample.

Band power (b. power) The band power quantifies the energy within a specific selected range of frequencies. It is computed by first performing the Fourier transform of $\mathbf{s}$, creating a frequency representation $\mathbf{s}(f)$. The band power within frequency range $[f_1, f_2]$ is later extracted as $\mathrm{BP}_{(\mathrm{f}_1, \mathrm{f}_2)}(\mathbf{s}) = \int_{f_1}^{f_2} |\mathbf{s}(f)|^2 \, df$. In this work, we consider 3 unique frequency range $\{[5, 10], [15, 30], [30, 80]\}$ as hyperparameters to extract a 96-dimensional feature for each studied synthetic data sample.

B.2 REAL-WORLD EXPERIMENTS

B.2.1 DATASET INFORMATION

Classification We selected 9 univariate datasets from the UCR archive (Dau et al., 2019), filtering out all datasets with less than 140 series length or less than 350 training samples. The dataset selection is performed to ensure each dataset has both sufficient samples and dynamics. The detailed information about the selected datasets is provided in Table 3.

We also selected 5 multivariate datasets from the UEA archive (Bagnall et al., 2018), excluding those with a series length below 100 and the training sample size below 200. The detailed information about the selected datasets is provided in Table 4.

Imputation For the imputation tasks, we use the ETDataset (Zhou et al., 2021), where ETTm1 and ETTm2 are sampled at minute intervals, and ETTh1 and ETTh2 are sampled at hourly intervals. The detailed information about the selected datasets is provided in Table 5.

Anomaly detection The detailed information about the selected datasets is provided in Table 6.

Dataset	Train	Test	Series Length	Classes
FordA	3601	1320	500	2
FordB	3636	810	500	2
ScreenType	375	375	720	3
ECG5000	500	4500	140	5
Wafer	1000	6164	152	2
StarLightCurves	1000	8236	1024	3
UWaveGestureLibraryAll	896	3582	945	8
HandOutlines	1000	370	2709	2
EthanolLevel	504	500	1751	4

Table 3: Detailed information about the selected datasets from the UCR archive.

Dataset	Channel	Train	Test	Series Length	Classes
EthanolConcentration	3	261	263	1751	4
Heartbeat	61	204	205	405	2
PEMS-SF	963	267	173	144	7
SelfRegulationSCP1	6	268	293	896	2
SelfRegulationSCP2	7	200	180	1152	2

Table 4: Detailed information about the selected datasets from the UEA archive.

Dataset	Channel	Series Length	Train	Validation	Test
ETTm1, ETTm2	7	96	34465	11521	11521
ETTh1, ETTh2	7	96	8545	2881	2881

Table 5: ETDataset for imputation tasks.

Dataset	Channel	Series Length	Train	Validation	Test
SMD	38	100	566724	141681	708420
MSL	55	100	44653	11664	73729
SWaT	51	100	396000	99000	449919
PSM	25	100	105984	26497	87841

Table 6: Detailed information about the selected datasets for the anomaly detection tasks.

B.2.2 Model training and architecture details

Training details For pre-training on synthetic datasets, we use a learning rate of 0.05, a MultiStepLR scheduler with a multiplicative factor $\gamma = 0.3$, and two milestones on epoch 30 and 150. We perform all pre-training for a total of 300 epochs on both of the synthetic datasets, where we set batch size as 1024 for the reconstruction task.

For pre-training on real-world datasets, we use a learning rate of 0.005. We perform all pre-training for either a total of 300 epochs, or a total of 6000 steps, whichever finishes the first. We set batch size as 32 for imputation and anomaly detection tasks, and batch size as 64 for classification tasks.

We apply the same set of hyperparameters for both parameter efficient fine-tuning and full-scale fine-tuning, where we perform hyperparameter selection on learning rate $\{0.005, 0.001, 0.05\}$ and batch size $\{32, 64, 128\}$. We perform the fine-tuning for 300 epochs on imputation, anomaly detection, and feature regression tasks, and perform the fine-tuning for 4000 steps on classification tasks.

The settings are applied consistently across all models to ensure a fair comparison. All models are optimized with an Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.99$, and a weight decay of 1×10^{-5}.

Model architectures For all pre-training methods including NoTS-lw, we use the same channel-independent 1D-ResNet encoder for fair comparison. The encoder has 3 ResNet layers of channel size $\{16, 32, 64\}$, each has 2 ResNet blocks. The first convolutional layer has a kernel size of 7, and the rest layers have a kernel size of 3. We append an additional convolutional layer after the ResNet blocks to alter the dimensionality d of the token embeddings, where model variant $d = 32$ is used for all experiments, and $d = 16, 64, 128$ is trained for the scalability pilot study. We use a 3-layer 4-head transformer with a token dimension of d, and $4\times$ size in the feed forward layer. The decoder is built to be symmetric to the encoder architecture.

In our experiments, iTransformer (Liu et al., 2023b) is implemented to transform inputs to an embedding dimension of 16, where we build the decoder to be symmetric and linear. We used a 3-layer 4-head transformer network with a token dimensionality of 128 and $4\times$ size in the feed forward layer. PatchTST (Nie et al., 2022) is implemented with a patch length of 16, stride of 8, and token dimension of 32. In cases where PatchTST becomes too computationally heavy (e.g., anomaly detection tasks), we adjust batch size to be 1 and increase patch length to be 32.

B.2.3 Complete experimental results

We show the complete classification results in Table 7 and Table 8 and the complete imputation results in Table 9. The averaged results are presented in Table 2.

Imputation task details We perform a channel-wise imputation task instead of the traditional random imputation task. Specifically, when performing the masking, instead of uniformly sampling random elements from $C \times T$ entries of $\mathbf{S} \in \mathbb{R}^{C \times T}$, we sample uniformly from T columns and cover inputs from all channels. This is to eliminate the overfitting issues from the data embeddings.

B.2.4 Visualizations and Scalability

Table 7: Complete classification results on the UCR datasets.

	Parameter efficient tuning				**Full-scale fine-tuning**			
Dataset	**NoTS-lw**	Next-pred	bioFAME	SimMTM	**NoTS-lw**	Next-pred	bioFAME	SimMTM
HandOutlines	71.62	64.32	64.05	88.92	93.51	72.16	91.62	89.73
EthanolLevel	28.60	26.60	25.20	29.00	91.40	48.60	41.20	38.00
StarLightCurves	91.66	87.34	85.15	88.59	97.21	97.49	97.56	97.39
UWave-GL-All	67.39	56.28	37.38	76.35	96.57	96.90	87.83	94.72
FordA	81.52	77.27	71.74	51.59	94.02	94.09	93.49	94.55
FordB	68.27	63.83	57.41	64.57	83.70	86.17	85.19	83.58
Wafer	98.78	89.21	89.21	89.23	99.81	99.87	99.64	99.85
ECG5000	91.31	89.00	92.96	87.38	94.13	88.47	94.29	93.82
ScreenType	47.73	39.73	40.53	42.67	42.40	41.87	42.93	43.20
Average	71.88	65.95	62.63	68.70	88.08	80.62	81.53	81.65

Dataset	Attaching NoTS to existing architectures			
	PatchTST	PatchTST + **NoTS**	iTransformer	iTransformer + **NoTS**
HandOutlines	91.89	93.51	92.16	92.16
EthanolLevel	57.80	68.00	86.20	86.40
StarLightCurves	97.46	97.56	93.94	93.52
UWave-GL-All	96.04	96.45	89.89	91.76
FordA	93.71	93.56	77.05	83.11
FordB	78.64	80.49	68.52	69.14
Wafer	99.59	99.63	99.72	99.77
ECG5000	94.09	94.33	94.42	94.49
ScreenType	42.93	44.00	42.13	45.07
Average	83.57	85.28	82.67	83.94

Table 8: Complete classification results on the UEA datasets.

Dataset	Parameter efficient tuning				Full-scale fine-tuning			
	NoTS-lw	Next-pred	bioFAME	SimMTM	**NoTS-lw**	Next-pred	bioFAME	SimMTM
EthanolConcentration	28.14	25.48	28.14	25.48	30.04	29.28	27.76	28.90
Heartbeat	73.66	73.17	72.68	72.20	74.63	73.66	73.17	73.17
PEMS-SF	75.72	58.38	77.46	64.16	80.35	67.63	75.15	72.25
SelfRegulationSCP1	79.18	77.82	69.97	59.39	89.08	86.01	85.67	74.06
SelfRegulationSCP2	57.22	56.67	53.33	55.56	57.78	57.22	56.11	57.78
Average	62.78	58.30	60.32	55.36	66.38	62.76	63.57	61.23

Dataset	Attaching NoTS to existing architectures			
	PatchTST	PatchTST + **NoTS**	iTransformer	iTransformer + **NoTS**
EthanolConcentration	25.10	25.48	30.42	30.04
Heartbeat	73.17	74.63	73.17	74.15
PEMS-SF	88.44	90.75	89.02	90.75
SelfRegulationSCP1	78.16	82.25	87.71	88.06
SelfRegulationSCP2	51.67	51.67	57.78	58.33
Average	63.31	64.96	67.62	68.27

Table 9: Complete imputation results with masking ratio 12.5% and 25%.

Dataset	Parameter efficient tuning				Full-scale fine-tuning			
	NoTS-lw	Next-pred	bioFAME	SimMTM	**NoTS-lw**	Next-pred	bioFAME	SimMTM
12.5% masking ratio								
ETTm1	0.1556	0.2832	0.1957	0.1573	0.1194	0.1219	0.1251	0.1207
ETTm2	0.1232	0.1774	0.1183	0.1243	0.1110	0.1164	0.1038	0.1041
ETTh1	0.2764	0.4569	0.2471	0.2545	0.2091	0.2126	0.1966	0.1947
ETTh2	0.1917	0.2692	0.1746	0.1796	0.1615	0.1886	0.1751	0.1632
25% masking ratio								
ETTm1	0.1730	0.3280	0.2103	0.1697	0.1246	0.1377	0.1325	0.1244
ETTm2	0.1294	0.1789	0.1257	0.1269	0.1205	0.1218	0.1095	0.1093
ETTh1	0.2957	0.4738	0.2695	0.2734	0.2266	0.2440	0.2068	0.2078
ETTh2	0.1994	0.2707	0.1824	0.1861	0.1653	0.1872	0.1806	0.1681

Dataset	Attaching NoTS to existing architectures			
	PatchTST	PatchTST + **NoTS**	iTransformer	iTransformer + **NoTS**
12.5% masking ratio				
ETTm1	0.1791	0.1657	0.1539	0.1662
ETTm2	0.1233	0.1193	0.1082	0.1071
ETTh1	0.3277	0.2705	0.2325	0.2227
ETTh2	0.1817	0.1797	0.1639	0.1609
25% masking ratio				
ETTm1	0.1837	0.1903	0.1698	0.1665
ETTm2	0.1295	0.1268	0.1140	0.1117
ETTh1	0.3668	0.2952	0.2483	0.2318
ETTh2	0.1926	0.1827	0.1725	0.1678

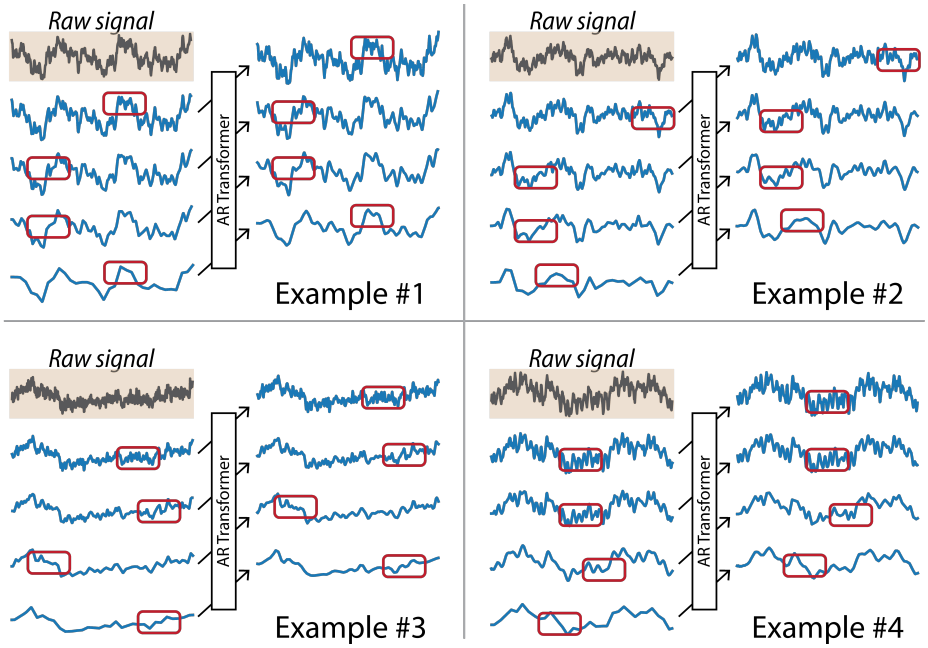

Figure 3: Additional data space visualizations.

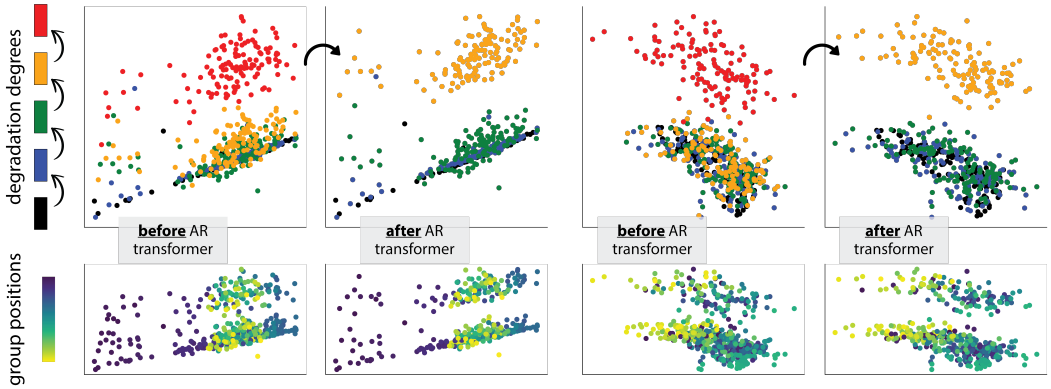

Figure 4: Additional token space visualizations.

C Detailed Model Deployment Pipeline

C.1 Overall Pre-training, Fine-tuning, and Testing Pipelines

C.1.1 Overall Model Deployment Flow

The proposed NoTS method, as a pre-training strategy, considers the following model training and deployment flow. It considers a pre-training dataset D_{PT} and a downstream dataset D_{FT} with samples shaped $C_{\text{PT}} \times T_{\text{PT}}$ and $C_{\text{FT}} \times T_{\text{FT}}$, respectively, allowing for differing channel and temporal dimensions between these two phases. The model is first pre-trained by performing the autoregressive reconstruction task, guided by the training objective as detailed in Equation 4. It is later fine-tuned on the training split of the downstream dataset, and is finally evaluated on the testing split of the downstream dataset. Depending on the differences between pre-training and downstream datasets, there are two schemes as presented in Table 10.

During pre-training, the input time series are sequentially processed by the encoding layers, the autoregressive transformer, and the decoding layers for the reconstruction task. In fine-tuning, ran-

Stage	Training	Tuning	Testing
Cross-domain	Synthetic	Real-world (train split)	Real-world (test split)
In-domain	Real-world (train split)	Real-world (train split)	Real-world (test split)

Table 10: Data splits for training, tuning, and testing.

domly initialized channel adapters and prompt task adapters are added, either trained independently or jointly with the pre-trained encoder, decoder, and transformer during the prompt tuning and full-scale fine-tuning stages, respectively. Finally, the tuned model, with all parameters frozen, is used for testing. The process is detailed as in Table 11.

Stage	Training	Tuning	Testing
Prompt tuning	$\mathcal{E}$, $\mathcal{D}$, Transformer	Adaptors	NA
Full-scale tuning	$\mathcal{E}$, $\mathcal{D}$, Transformer	$\mathcal{E}$, $\mathcal{D}$, Transformer, Adaptors	NA

Table 11: Parameters updated during training, tuning, and testing.

C.1.2 Domain shifts across pre-training and fine-tuning

Domain shift issues between pre-training and downstream datasets are discussed in many recent works on pre-training methods for time series data (Dong et al., 2024; Liu et al., 2023a). These challenges primarily stem from differences between D_{PT} and D_{FT}, leading to the following two cross-domain and within-domain schemes:

Cross-domain pre-training and fine-tuning. All NoTS-lw models are trained and evaluated using the cross-domain scheme (first 8 rows of Table 2). Specifically, the models are pre-trained on the synthetic datasets detailed in Section 3.1 to learn and extract the universal dynamic features, then tuned on real-world datasets to demonstrate the generalization capability of the pre-training approach.

Within-domain pre-training and fine-tuning. All NoTS models are trained and evaluated using the within-domain scheme (Table 1 and last 4 rows of Table 2). This setting assesses the model's ability to learn dataset-specific patterns by pre-training and fine-tuning on the same train split of a dataset. The final evaluation of the downstream task in fine-tuning is evaluated on the test split of the corresponding dataset.

C.1.3 Prompt tuning and full-scale fine-tuning

We design the prompt tuning method to adapt pre-trained knowledge to new tasks while introducing minimal additional parameters. In this tuning paradigm, the pre-trained transformer (along with the encoder and decoder) remains frozen. Only the adaptor parameters (channel and task adaptors) and, for discriminative tasks, a final linear classification head are trained.

In contrast, the full-scale fine-tuning method unfreezes the pre-trained weights, allowing the entire model, including the encoder, transformer layers, the decoder, the adaptors, and the final linear classification head, to be fine-tuned. This approach enables comprehensive adaptation to new data and tasks but comes with higher computational demands.

C.1.4 Testing Schemes

The model is tested on the testing split of the downstream dataset. The specific testing scheme varies across different downstream tasks, with major differences across generative (anomaly detection and imputation tasks) and discriminative tasks (classification and regression tasks). More details are included in the downstream tasks section.

C.2 Mathematical formulation of our prompt-tuning strategy

C.2.1 Task adaptors inspired by Visual Prompt Tuning (VPT)

Consider the input sequence to transformer layer $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{N \times D}$, where N is the sequence length and D is the transformer's hidden dimension. Our prompt tuning pipeline deploys the deep visual prompt tuning strategy as follows:

We randomly initialize a set of k learnable prompt tokens $\{p_1, \ldots, p_k\}$, where each $p_i \in \mathbb{R}^D$. For each transformer layer d, suppose we have H attention heads, and we initialize three separate linear projection layers for the prompt tokens:

$$L_{q_p}^{(d)}, \quad L_{k_p}^{(d)}, \quad L_{v_p}^{(d)} \in \mathbb{R}^{D \times (H \times D)},$$

each without bias. These layers project the prompt tokens into query, key, and value representations of shape $\mathbb{R}^{H \times D}$ as follows:

$$p_{q_i}^{(d)} = p_i L_{q_p}^{(d)}, \quad p_{k_i}^{(d)} = p_i L_{k_p}^{(d)}, \quad p_{v_i}^{(d)} = p_i L_{v_p}^{(d)}, \quad \text{for } i = 1, 2, \ldots, k.$$

While the input tokens $\mathbf{x}_j \in \mathbb{R}^D$, where $j = 1, 2, \ldots, N$ have separate linear projection layers that are inherited from the pre-trained transformer attention blocks, giving $\mathbf{x}_{q_j}^{(d)}$, $\mathbf{x}_{k_j}^{(d)}$, and $\mathbf{x}_{v_j}^{(d)}$ for each j in each transformer layer d. The projected prompt tokens and input tokens are then concatenated separately to form the augmented query, key, and value sequences:

$$\begin{aligned}
\mathbf{Q}_{\text{aug}}^{(d)} &= [p_{q_1}^{(d)}, p_{q_2}^{(d)}, \ldots, p_{q_k}^{(d)}, \mathbf{x}_{q_1}^{(d)}, \mathbf{x}_{q_2}^{(d)}, \ldots, \mathbf{x}_{q_N}^{(d)}], \\
\mathbf{K}_{\text{aug}}^{(d)} &= [p_{k_1}^{(d)}, p_{k_2}^{(d)}, \ldots, p_{k_k}^{(d)}, \mathbf{x}_{k_1}^{(d)}, \mathbf{x}_{k_2}^{(d)}, \ldots, \mathbf{x}_{k_N}^{(d)}], \\
\mathbf{V}_{\text{aug}}^{(d)} &= [p_{v_1}^{(d)}, p_{v_2}^{(d)}, \ldots, p_{v_k}^{(d)}, \mathbf{x}_{v_1}^{(d)}, \mathbf{x}_{v_2}^{(d)}, \ldots, \mathbf{x}_{v_N}^{(d)}].
\end{aligned}$$

These augmented query, key, and value sequences are then processed by the multi-head attention mechanism with H heads in transformer layer d, allowing the model to integrate information from both the prompt tokens and the input time series tokens.

C.2.2 Channel adaptors and channel embedding

The channel embedding layer is applied before feeding data into the encoding layers, transforming the channel dimension from C to C' as follows:

$$L(\mathbf{S}) = W_{\text{embed}}\mathbf{X} + \mathbf{b}_{\text{embed}},$$

where $W_{\text{embed}} \in \mathbb{R}^{C' \times C}$ and $\mathbf{b}_{\text{embed}} \in \mathbb{R}^{C'}$ are learnable parameters.

Additionally, the learnable channel tokens $\mathbf{T} \in \mathbb{R}^{C \times D}$ are used as additive embeddings before the input sequence enters the transformer layers. These tokens recalibrate channel-specific representations, ensuring consistent adjustments across all tokens. This method enhances robustness and flexibility across diverse domains and channel configurations without requiring significant architectural changes.

C.3 Additional experimental details for downstream tasks

We follow the multitask evaluation pipeline in (Wu et al., 2022) and provide the necessary details as follows.

Classification We consider the sequence-level classification task, where each time series sample is mapped into one label. The classification is performed by injecting prompt tokens into the task adapter, where the transformer processes the appended token sequence $[p_1, p_2, \ldots, p_k, x_1, x_2, \ldots, x_N] \in \mathbb{R}^{(N+k) \times D}$, where k is the number of prompt tokens, N is the number of input tokens, and D is the transformer hidden dimension. We selected $k = 5$ for all classification tasks, where the prompt tokens are averaged and passed into a linear projection head to obtain classification logits.

Anomaly detection We consider the unsupervised time series anomaly detection task. During the training stage, we minimize the reconstruction error of the normal data using MAE loss. In testing, anomalies are detected by comparing reconstruction errors to a threshold derived from the error distribution of the entire dataset. We employ event-based anomaly detection and compute the detection accuracy for model evaluation. For all experiments, we sweep the anomaly rates across the data distribution $\{0.5\%, 0.75\%, 1.0\%, 1.25\%, 1.5\%\}$ to compute the final rate for each dataset.

Imputation We randomly generate and apply masks to the input to simulate and reconstruct the missing values. We evaluate imputation masking rates of 12.5% and 25%, with the mask ratio set to 1.5 times higher during training. The training objective minimizes the Mean Squared Error (MSE) between the ground truth and the model's imputed outputs, focusing on both masked points and the entire series.

Regression The regression task is performed exclusively on the synthetic dataset in a within-domain manner using a regression task adaptor during fine-tuning. The procedure is similar to the classification task, with the key difference being the loss function: classification uses cross-entropy loss, while regression employs MAE loss for fine-tuning.

D Related works

D.1 Pre-training methods for time series

Pre-training on large-scale datasets has shown promise in learning generalizable patterns, especially for time series where downstream dataset is often small(Liu et al., 2021; Zhang et al., 2022; Woo et al., 2022). Reconstruction-based pre-training in transformers typically follows two approaches: masked modeling and next-period prediction. Masked modeling, as in SimMTM (Dong et al., 2024) and bioFAME (Liu et al., 2023a), predicts masked elements using neighboring points or frequency-based information. Next-period prediction, as in Time-GPT1 (Garza & Mergenthaler-Canseco, 2023), uses simple next-period prediction, while Chronos (Ansari et al., 2024) tokenizes time series via scaling and quantization to model categorical distributions. Building on these, our work mitigates the loss of nonlocal information of patches by learning cross-scale interplay among degraded functional components.

D.2 Multi-scale methods for time series

Most multi-scale time series methods focus on supervised learning for single tasks, such as TimesNet(Wu et al., 2022), which folds 1D series into 2D shape using periodicity, and CrossGNN(Huang et al., 2023), which models cross-scale correlation with GNNs. (Wang et al., 2024) is another notable work which proposes multiscale-mixing to disentangle temporal patterns via decomposition and cross-scale mixing in an MLP-based framework. For contrastive learning methods, TS2Vec(Yue et al., 2022) introduces hierarchical contrastive learning to generate universal timestamp-level representations for time series that can be used for various downstream tasks. (Zhou et al., 2021) Informer uses multiscale ProbSparse attention and pyramidal distillation in its encoder, combined with a generative decoder, for efficient long-sequence forecasting (Zhang et al., 2022).

In the context of masked-based SSL for time series pre-training, some approaches utilize varying patch sizes to hard-code multi-scale information directly into the input data(Das et al., 2023; Woo et al., 2024). One notable work, HiMTM (Zhao et al., 2024) models hierarchical scales by merging latent tokens from fixed-size neighboring patches, as the transformer layers deepen. In contrast, our work introduces a fundamentally different approach to hierarchical resolution modeling: we generate a coarse-to-fine sequence by progressively degrading the input temporal function, pre-training the transformer to reconstruct the original signal.

D.3 Learning time series from the functional view

Traditional time series methods often adopt a functional perspective, employing statistical techniques (Holt, 2004), the Theta method (Assimakopoulos & Nikolopoulos, 2000), or ARIMA models (Hyndman & Khandakar, 2008). These have evolved into deep learning approaches like N-BEATS (Oreshkin et al., 2019), which uses fully connected layers for hierarchical decomposition via neural

bases, and N-HiTS (Challu et al., 2023), which enhances N-BEATS with subsampling for multi-frequency data. Our work differs from prior approaches by focusing on the advantages of building sequences with functional aware relationships through the transformer architecture.