FracTS: Hierarchical and Autoregressive Time Series Generation

Jiayu Li[2,†,*] **Umair Afzal**[1,*] **Zilong Zhao**[1,3,†]
Milad Abdollahzadeh[1] **Uzair Javaid**[1] **Biplab Sikdar**[3]
[1]Betterdata AI [2]University of Illinois Urbana-Champaign
[3]National University of Singapore

Abstract

Time series are ubiquitous across different domains, and the growing use of synthetic data for privacy-preserving sharing, data augmentation, and stress testing makes *time series generation* increasingly important. Early generative approaches are predominantly autoregressive, which can suffer from low sample diversity, error accumulation, and difficulty modeling long-range dependencies. In this paper, we introduce **FracTS**, a hierarchical, coarse-to-fine autoregressive model for multivariate time series generation. Inspired by fractal generative modeling developed for images, FracTS adapts the paradigm from 2D to 1D sequences by learning a multi-scale representation that captures global trajectory structure at coarse scales and local dynamics at fine scales while following the autoregressive nature of time series. In contrast to unstructured images, time series are structured, allowing easy automated or designated temporal feature processing optimized for the model, and global (static) feature extraction describing the instances that generate the corresponding time series in the collection. We incorporate them in the generation process to further improve the generation quality. The design naturally handles multivariate dependencies and improves long-horizon temporal coherence without requiring fixed-length inputs. Experiments on five datasets (two conditional, three unconditional) conducted against seven baselines show that FracTS consistently outperforms baselines across marginal distribution and predictive ML utility, and correlation preservation in the unconditional setting. In particular, the predictive ML utility is averagely improved by 100.0% and 75.0–81.0% for unconditional and conditional time series datasets compared to the best baseline (relative error computed on dataset-averaged raw values). Code is available at: https://github.com/BetterdataLabs/fracts.

1 Introduction

Time series data – sequences of observations over time – are ubiquitous across scientific, engineering, and social domains. Such data arise in diverse applications ranging from finance and economics to healthcare and manufacturing [1, 2]. This pervasiveness of time series has driven extensive research in tasks like forecasting, classification, and anomaly detection on temporal data. Generative modeling, in particular, of time series serves several critical purposes in data science and machine learning and widely utilized in real-life applications: simulating plausible trajectories for what-if and stress testing [3]; augmenting scarce or imbalanced datasets [4]; privacy-preserving data sharing and benchmarking [2]; and robust evaluation of downstream decision systems [5].

*Equal contributions. Work done while working at Betterdata AI
†Correspondence: jiayul11@illinois.edu, zilong@betterdata.ai

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

By the diversity of data sources and applications, time series generation can be posed under several settings. In this paper, we focus on the generation of ***new time series instances*** *(resp. new time series steps)* ***in a collection*** of time series instances (e.g., new users' transaction history, new companies' stocks). Thus, **forecasting** or **imputation**, which are also referred to as "time series generation" occasionally, are not discussed in this paper. Differentiated by the presence of explicitly known global features of the new instances, we classify the time series generation problems we cover into two major types: **unconditional** and **conditional** generation.

Early pioneers of time series generative models have mainly been autoregressive predictions of next steps [6–8], as autoregression aligns closely with the nature of time series, where new values are obtained or observed with dependency on the context of the past. However, autoregressive prediction typically suffers from small variance due to its deterministic nature [9], accumulates error over time steps [9, 10], and lacks understanding of long-range dependency and global information [11]. Consequently, many new time series generative models attempt to introduce auxiliary global components [9, 12, 13, 10, 14, 15]. That time series is inherently step-wise autoregressive, but requires auxiliary modules to complement a global understanding, naturally calls for hierarchical modeling approaches to reconcile local and global information in a coarse-to-fine manner. Some existing time series generative models introduce hierarchical components specifically for time series data [8, 13]. Nevertheless, generative models that are hierarchical by themselves [16–21] seem to be a more natural choice, but there is a lack of direct applications of these models on time series, except for limited exceptions [22].

In this paper, we introduce **FracTS** - a **Frac**tal Generative Model (FGM) [21] for **T**ime **S**eries. Time series data possesses the fractal property like images [21], as smaller chunks of time series are also time series. Inspired by this, we split long time series into shorter chunks, capturing the inter-and intra-chunk relations at two (full and chunked) fractal levels. This process continues until each chunk has length one, which is then modeled by a special final fractal level that learns across different variates, analogous to channels in images.

Unlike images where patches do not have specific orders, time series are naturally autoregressive. Thus, we adopt the autoregressive version of FGM [21] instead of the default choice of masked autoregressive version for images. By a backbone of autoregressive transformer, the model naturally inherits the property of vanilla autoregressive transformers in handling varying sequence lengths [23]. The structured nature of time series, specifically in comparison to the unstructured images, allows standard automated or domain-specific feature processing optimized for the generative models [24, 25, 13, 12], including tabular-formatted type-specific data processing of temporal features, and global static feature extraction. In this paper, we also introduce an effective data encoding for time series data in our FracTS model. Our main contributions can be summarized as follows:

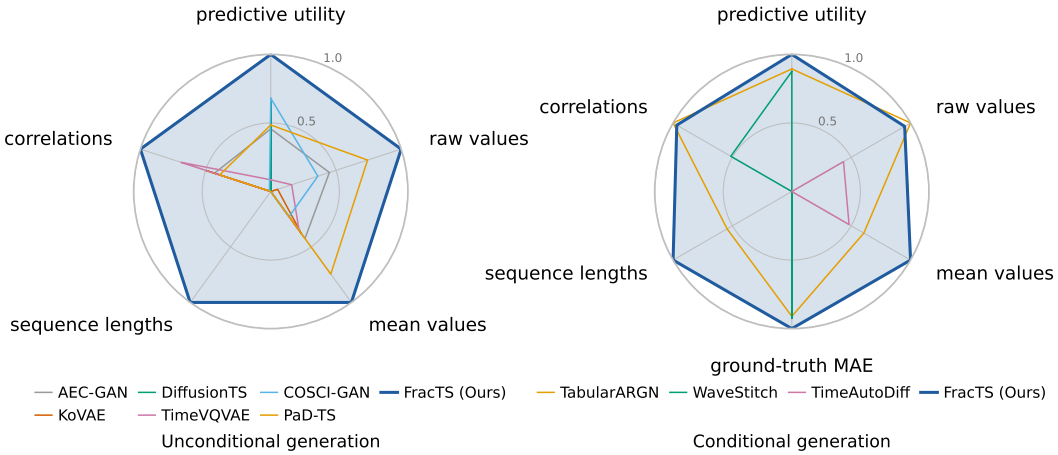


Figure 1: Summary comparison of FracTS (ours) and the baselines on dataset-averaged scores of Tables 3 and 4, rescaled per axis so that larger values indicate better performance. Clockwise from top: predictive ML utility at window $W=1000$, marginal distribution of raw values, of means and of lengths, and correlation preservation on raw values; conditional adds ground-truth MAE at the bottom (definitions in Section 5.1).

1. We propose FracTS model, an effective and intuitive hierarchically autoregressive generative model for conditional/unconditional generation of realistic time series data.
2. We introduce a robust time series data processing and compression pipeline optimized for transformer models.
3. Our proposed FracTS model applies on much more flexible data settings for time series data without performance degradation in terms of lengths (variable, or very long) and features (diverse data types and distributions) than most existing models.
4. Experiments demonstrate that FracTS consistently outperforms baselines across five datasets: compared to the best baseline, the predictive ML utility is improved by 100.0% on unconditional datasets and by 75.0–81.0% on conditional datasets, with consistent gains on the marginal distribution and correlation scores as well (Figure 1; per-column values in Tables 3 and 4).

2 Related Works

Time Series Generation. Early time series generative models have mainly been step-wise autoregressive predictions due to the autoregressive nature of time series. For example, teacher forcing [6] and professor forcing [7] use recurrent neural networks to generate sequences autoregressively, and WaveNet [8] autoregressively predicts the distribution of each step based on all previous steps with causal convolutions. However, autoregressive prediction tends to accumulate errors and lacks global understanding [9–11]. Therefore, recent models often introduce some global components, using modern generative methods such as generative adversarial networks (GAN) [26], variational autoencoders [27], diffusion models [28], and autoregressive transformers [29, 23]. For instance, TimeGAN [9] uses GAN to generate time series, which contains a global distribution objective on top of the next-step prediction objective, and DiffusionTS [12] uses a diffusion model to generate realistic time series data with both slow and rapidly-varying components for trend, seasonality, and error, respectively. Similarly, PaD-TS [30] introduces a population-aware training objective to diffusion models, explicitly preserving dataset-level statistics such as cross-dimension correlations alongside individual-level authenticity. Even for some models that stick to autoregressive prediction for generation, specific designs to counter the drawbacks of autoregressive methods are introduced, such as the masked variant of autoregression with vector quantization in TimeVQVAE [13] and the error correction module that is trained globally in AEC-GAN [10]. Note that the detailed problem settings of different models vary in terms of the lengths of time series, number of variants and their types, and presence and nature of static conditions. Most existing models are not immediately applicable to all settings, while this paper aims to handle a general case that covers all scenarios (more details in Section 3).

Hierarchical Understanding of Time Series and Hierarchical Generative Models. Hierarchical modeling is present in many time series generative models, explicitly or implicitly, such as the diluted causal convolution of WaveNet [8], and the separation of high and low frequency data in TimeVQVAE [13]. Hierarchical modeling is more prevalent in other time series tasks, such as forecasting and interpolation [31–33]. However, there are hierarchical variants of most modern deep generative models, such as hierarchical hidden Markov models [34], VAEs [16, 19], GANs [17, 18], diffusion models [20], and autoregressive transformers [21], mostly designed for other data modalities. There are applications of these hierarchical generative models to time series. For example, MR-HDMM [35] generates time series via multi-rate hierarchy using deep Markov models, and PSA-GAN [22] generates time series using a progressive self-attention GAN. Most recently, TimeMAR [36] proposes a multi-scale autoregressive framework using a dual-path VQ-VAE to disentangle trend and seasonal components for unconditional generation.[3] Nevertheless, to the best of our knowledge, there is no time series hierarchical generative model that generates data by fine-grained data units down to one variate of one time step, while natively supporting multivariate mixed-type features and complex static conditions.

Time Series Structured Data Processing. Data processing is crucial to tabular modalities. For instance, CTGAN [24] uses a simple GAN network, but the data processing of one-hot encoding for categorical data and variational Gaussian-mixture model decomposition of multimodal numeric data makes its performance outstanding. TabDiff [25] uses Gaussian and masked diffusion for numeric

[3]The code for TimeMAR is not publicly available, and thus it is not included as baseline.

and categorical features, while the transformer-based denoising network backbone remains relatively standard. Extending this capability to the time domain, TimeAutoDiff [37] employs a unified latent diffusion framework with a VAE backbone to natively model heterogeneous time series features, including categorical and binary variables. However, while time series data typically inherit the data properties of general tabular data, many time series generative models downstate this aspect, even overlooking the presence of categorical features [14, 13, 15, 12], which will be addressed in this paper. Additionally, some models leverage domain-specific processing, such as TimeVQVAE [13] and DiffusionTS [12], which utilize Fourier transforms to capture multi-frequency dynamics.

3 Problem Formulation

Table 1: Functionality and flexibility of representative time series generative models, compared to FracTS. ✗ indicates that a model is *limited to* the corresponding setting; for *limited length*, ✗ means the model contains no specific design for long time series generation. Blank entries indicate that the setting is supported. † marks models whose static conditioning is restricted to class labels.

Limited to	TimeGAN	WaveStitch	COSCI-GAN	TimeVQVAE	AEC-GAN	KoVAE	DiffusionTS	FracTS
i) fixed length	✗		✗	✗		✗	✗	
ii) single variate				✗				
iii) no static			✗	✗ †	✗	✗	✗ †	
iv) limited length	✗	✗	✗	✗		✗	✗	
v) mixed type			✗	✗	✗	✗	✗	

General Time Series Setting. Time series appear in several forms. For example, they can be *uni*-or *multi-variate*, where the latter is more complex due to the cross-variable and lagged dependencies. Sequences may be *irregularly sampled* (with explicit timestamps, usually *event-based*), have *variable length*, and their values may be *continuous* and *discrete/categorical*. Many time series are also accompanied with *static conditions* describing the sequence as a whole (e.g. demographics, asset metadata), which can also have diverse data types. In this paper, we use the most generic but complex setting — multivariate time series generation of various lengths and diverse data types.

Let $\mathcal{D} = \{(X_n, c_n)\}_{n=1}^{N}$ be a time series dataset comprising N sequences. Each sequence $X_n = (x_{n,1}, \ldots, x_{n,L_n})$ has L_n time steps, where L_n may vary across sequences. We set $L = \max_{n=1,\ldots,N} L_n$ as the maximum length. At each step t, the observation $x_{n,t} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_M$ is a multivariate measurement over M features, whose domains $\mathcal{X}_m$ can be numerical, categorical, or datetime-valued. Each sequence is accompanied by a static condition vector $c_n = (c_n^{(1)}, \ldots, c_n^{(M_c)})$ of M_c attributes (categorical, numeric, or datetime).

The goal of generative modeling for time series is to learn a conditional distribution $p_\theta(X \mid c)$ that approximates the real data:

$$p_\theta(X \mid c) \approx p_{\text{data}}(X \mid c) \quad \text{for any } c \sim p_{\text{data}}(c). \tag{1}$$

At generation time, c may be specified by the user or sampled from a learned marginal $q_\phi(c) \approx p_{\text{data}}(c)$. Faithful approximation implies that generated sequences preserve key statistical properties of real time series, including:

i) the marginal distribution of each feature across all sequences and time steps: $p_\theta(X_{:,m}) \approx p_{\text{data}}(X_{:,m})$;

ii) the temporal dynamics within a sequence: $p_\theta(x_{n,t} \mid x_{n,1:t-1}) \approx p_{\text{data}}(x_{n,t} \mid x_{n,1:t-1})$;

iii) the instantaneous cross-feature dependencies: $p_\theta(x_{n,t}) \approx p_{\text{data}}(x_{n,t})$.

Challenges of General Time Series Generation Most existing works simplify the setup considerably relative to our general formulation (variable length, multivariate mixed-type features, static conditions). Common restrictions are: i) fixed length $L_n = L$, ii) univariate series ($M = 1$), iii) no static attributes ($M_c = 0$), iv) short sequences (e.g., $L < 1000$), and v) purely numeric features ($\mathcal{X}_m \subseteq \mathbb{R}$). Table 1 summarizes these simplifications; only a limited subset of prior works addresses the general case [38], and in this work we relax these restrictions to better mirror real-world scenarios.

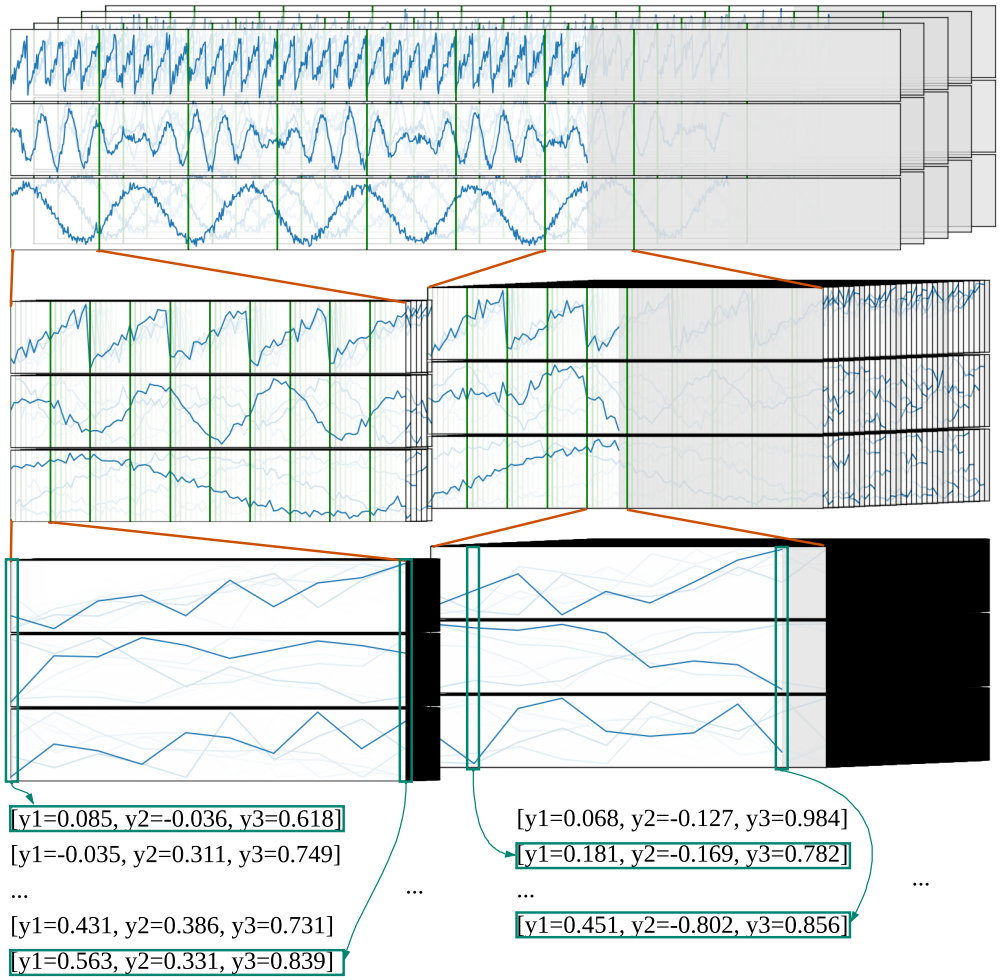


Figure 2: Fractal decomposition and model architecture for FracTS (with $K = 4$ fractal levels). At each level except for the last one, a long time series is decomposed into patches of shorter ones. The time series dataset can have variable lengths, where the gray spans indicate paddings. Completely empty patches are skipped for training. The last level $i = 3$ is not a fractal decomposition of different time steps, but a series of different variables in *one* time step.

4 FracTS Model

The underlying motivation of FGM is the existence of a fractal pattern in the data, namely, a recursive self-similar pattern, enabling divide-and-conquer generation. Time series data has such a fractal pattern, as long time series are composed of shorter time series. This hierarchical structure is also utilized in some existing time series models (recall Section 2). Nevertheless, time series and images are essentially different in many aspects, including i) time series is essentially 1D while images are 2D, ii) the shape of each data point for time series could vary due to variable time series lengths while it is fixed for most image datasets, iii) the distribution of each value unit is much more complex in time series as time series contain both discrete and continuous attributes with different distribution while the pixel values are integers approximately normally distributed, and same applies to the condition attributes of time series, in contrast to the only class labels for images in FGM. Therefore, FGM needs to be heavily customized to transfer from a visual model to a time series model.

4.1 Fractal Architecture for 1D Time Series Data

In FGM for images, each image is decomposed into patches, where each patch is a sub-image with smaller width and height. For time series, there are no widths and heights, but lengths, so we

decompose a time series into patches where each patch is a shorter time series. The last level of FGM for images is designed for exact values in each channel at one pixel, which we analogize to the exact values of different variables at one time step in time series, and this level has a length of 1 in terms of steps in the time series. Thus, the FracTS decomposition of time series is shown in Figure 2. Formally, in a fractal setting with K levels, let the length of time series handled at fractal level $i = 0, 1, \ldots, K-1$ be $L^{[i]}$ ($L^{[0]} = L$), then at each fractal level, FracTS decomposes a time series of length $L^{[i]}$ into shorter time series of length $L^{[i+1]}$, and the last level has $L^{[K-1]} = 1$. Following FGM for images, each fractal level is learned by a sequential model (i.e., transformer [29]). Let the sequential model's decoder length at fractal level i be $l^{[i]}$, then $l^{[i]} = \left\lceil L^{[i]}/L^{[i+1]} \right\rceil$ for $i \neq K-1$ and $l^{[K-1]} = M$. Let the number of data points at level i be $N^{[i]}$, then $N^{[i]} = \sum_{n=1}^{N} \left\lceil L_n/L^{[i]} \right\rceil \leq N \left\lceil L/L^{[i]} \right\rceil$. Therefore, we formally learn

$$\log p(X|c) = \mathbb{E}_n \Bigg[\overbrace{\sum_{i=0}^{K-1} \frac{1}{l^{[i]}} \sum_{j=1}^{l^{[i]}} \log p(X_{n,jL^{[i+1]}+1:(j+1)L^{[i+1]}} \mid X_{1:jL^{[i+1]}}, c)}^{\text{recursive fractal levels}} + \underbrace{\frac{1}{L_n} \sum_{t=1}^{L_n} \log p(X_{n,t,m}|X_{n,1:t-1}, X_{n,t,1:m-1}, c)}_{\text{final features level}} \Bigg] \tag{2}$$

Another unique feature of 1D time series compared to 2D images is the need for long-term dependencies. Although global dependency is implicitly captured by patch embeddings inferred from a prefix including conditions from previous fractal levels, we argue that global information gradually vanishes down the fractal levels, similarly to the problem with deep neural networks with too many layers. To counter this effect, we adopt a ResNet-like [39] skip connections to directly pass the global condition from the previous fractal levels down to the next levels with a simple projector for space alignment.

Theorems 4.1 and 4.2 (proofs in Appendix G) verify in theory that the hierarchical architecture reduces error accumulation across long sequences (error scale $\Omega(\delta l K)$ versus $\Omega(\delta L)$ for a flattened autoregressive process) and justify its smaller memory usage compared to a flattened transformer with large L.

Theorem 4.1. *For a sequence of total length L generated by a K-level hierarchical autoregressive architecture with branching factor l (where $K = \log_l L$), if the local parameter estimation error per step is δ, the total KL divergence between the ground-truth distribution $p(X_{\cdot,1:L})$ and the generated distribution $p(\hat{X}_{\cdot,1:L})$ satisfies:*

$$D_{KL}\left(p(X_{\cdot,L} \mid c) \| p(\hat{X}_{\cdot,L} \mid c)\right) = \Omega(\delta l K). \tag{3}$$

Theorem 4.2. *Let B be the batch size, L the sequence length, d the hidden dimension, and l the local patch size per level. For a hierarchical model like FracTS with $K = \log_l L$ levels, the total activation memory complexity is $\mathcal{O}(BL(l+d))$.*

4.2 Learning and Constructing Time Series of Variable Lengths

Unlike fixed-size image datasets, time series datasets often contain variable-length instances due to heterogeneous user behaviors and company histories. Similarly to the transformers for text [29, 23], we apply attention masks to avoid learning positions with no data. Moreover, models are not trained on completely empty patches, which saves much of training time when $L \gg L_n$ for most n.

To determine where the time series ends, we use a *length indicator predictor*. The idea is similar to the EOS token for natural languages. We used an additional simple MLP-based binary classifier to predict whether a position corresponds to the end-of-sequence. We refer to this Boolean signal as the *length indicator*. The process is visualized in Figure 3. Since termination signals in time series are often implicit, we additionally provide sequence length as a static feature to the context generator q_ϕ and the length predictor. Unlike simple truncation, this design enables termination to depend on local dynamics, such as abrupt changes before a stock leaves the market.

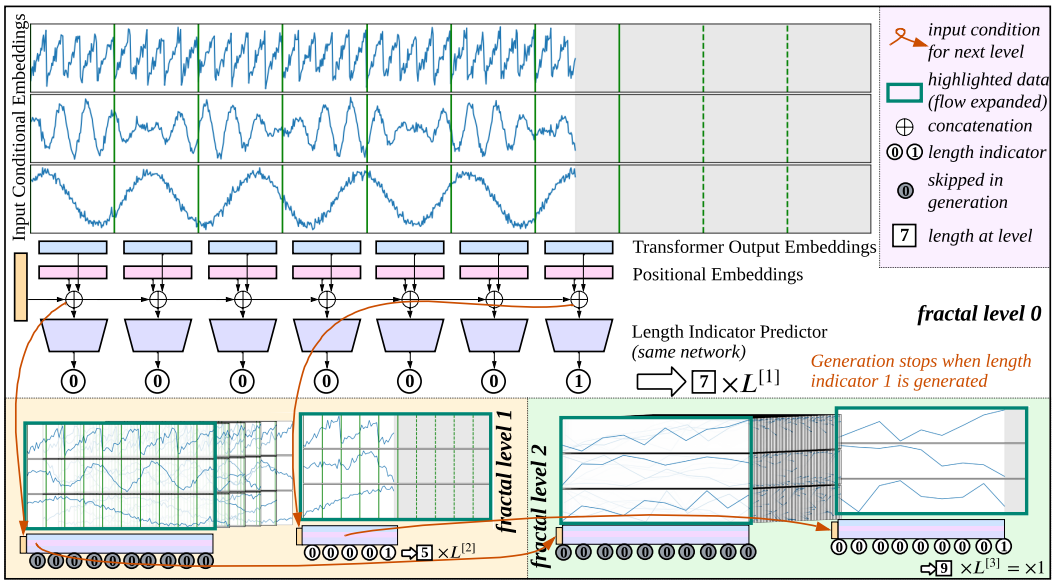


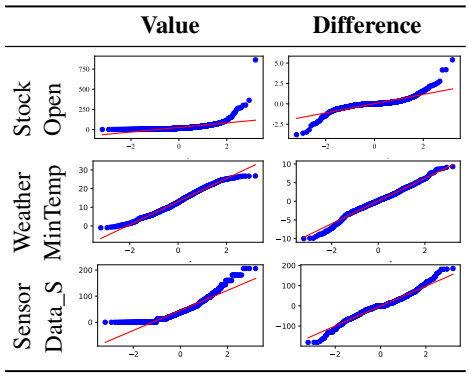
Figure 3: Length construction for variable-length time series data via length indicator predictors (purple trapezoids). At each fractal level, the length indicator is one network (drawn separately for separate calls). The length indicators are equivalent to is-[EOS] binary signals. Details at fractal levels 1-2 are omitted as they are the same as level 0, and the last level, unrelated to lengths, is hidden. The total length of the example in the figure is $L_n = (7-1)L^{[1]} + (5-1)L^{[2]} + 9L^{[3]} = 649$.

4.3 Handling Multivariate and Heterogeneous Mixed-Typed Features

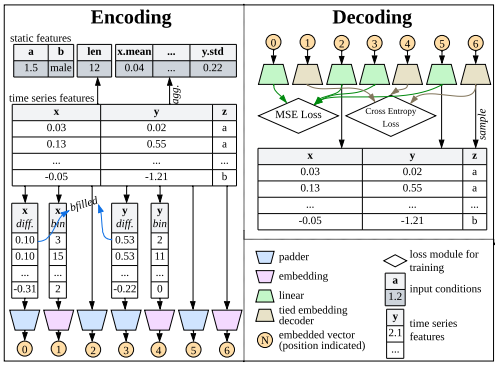
Time series data are typically stored in tabular form and can be viewed as tabular data with additional temporal dependencies. To preserve tabular properties, we encode each feature as separate positions for the last level's transformer. Categorical features are embedded by a vanilla embedding layer. Continuous features are decomposed into two positions: one K-Means binned category embedded similarly to categorical features, and one actual value padded to the embedding dimensions. To decode continuous values back, we take the second position only and pass it through a linear feed-forward layer. This decomposition of continuous value into bin and raw value is inspired by [40] for learning multimodal numeric distributions, but we use the raw numeric value with projections at the second position. Having the auxiliary bin values leading the raw values introduces randomness by multinomial sampling of the leading discrete component. This compensates for the lack of randomness of generation based on embeddings only in transformers [29]. This encoding method applies to both the input static features and the output time series features.

Unlike plain tabular data, time series contain temporal trends. For example, a $5 increase from yesterday may be as important as the current value itself. We analyzed the data of realistic time series datasets to compare the distributions of raw values and differences to previous time steps. The results are shown in Figure 4a, and more in Appendix A. The distributions of differences are closer to normal distributions. In cases where both distributions are far from normal distribution, differences typically avoid one-tailedness, making the result closer to normal distribution. In an extreme but simple case of linearly increasing values, the raw values are uniformly distributed, but differences are constants. The latter is obviously much simpler than the former. Therefore, to improve the model's capability in capturing the trend, we explicitly insert the difference to the previous time step for numeric features as an additional auxiliary leading numeric feature like the K-Means bins.

To provide more informative conditions, we compute simple summary statistics of each time series (e.g., mean and standard deviation) as additional static features. However, this produces many condition tokens that can overwhelm the context when directly prepended. To address this, we adopt a Perceiver-style condition compression module [41], which distills these features into a compact representation, avoiding context overload. The full pipeline is illustrated in Figure 4b.



(a) Q–Q plots comparing the distributions of raw values and differences of time series against the normal distribution. The closer the blue points lie along the red diagonal line, the more normally distributed the data.



(b) Data processing of time series data. The left and right halves show the data **encoding** and **decoding** processes, respectively. Processing of static features, including length and aggregated values, is omitted as they follow that of time series features.

5 Experiments

5.1 Experiment Setup

Datasets. We evaluate our models' unconditional generation ability on three real-world multivariate time series datasets from diverse domains, each posing distinct challenges due to their size, sequence length, and data characteristics: **Weather** [42], **Sensor** [43], and **Stock** [44]. For conditional generation, we use **Walmart** dataset [45]. We also construct a simulated dataset from mathematical functions to capture diverse temporal patterns, including linear, periodic, stepwise, convex, non-convex, modulated, and composite forms. Due to its mathematical design, this dataset is inherently conditional. Table 2 summarizes the five datasets; more details are provided in Appendix B, with overall statistics in Appendix B.1 and the generating formulas of the simulated dataset in Appendix B.3.

Table 2: Summary of the datasets used and their core statistics. Notations follow Section 3; $\overline{L_n}$ is the average sequence length.

Dataset	N	M_c	$\overline{L_n}$	L	M	Type
Weather	44	0	1798	2171	17	unconditional
Sensor	6939	0	51	139	17	unconditional
Stocks	1000	0	1278	1511	6	unconditional
Walmart	45	2	9368	10474	3	conditional
Simulated	100	4	15000	15000	15	conditional

FracTS Implementation. We implement FracTS using a hierarchical architecture with constant patch size $l^{[i]} = 20$ across all fractal levels $i \in \{1, \ldots, L\}$. Each fractal level employs a GPT2 small transformer [23] as its base model $\mathcal{M}^{[i]}$. For optimization, we use AdamW [46] with hyperparameters: learning rate $\eta = 2 \times 10^{-4}$, weight decay $\lambda = 0.01$, and gradient clipping threshold $g_{\text{clip}} = 1.0$. Training is performed for a fixed number of $T = 200$ epochs, and the best model is selected based on the lowest validation loss observed during training. We evaluate model performance on the validation set after each epoch to monitor convergence.Further implementation details are provided in Appendix C, including the datetime and aggregation processing in Appendices C.1 and C.1 and the full experimental configuration in Appendix C.2, while component ablations and hyperparameter sensitivity analyses are provided in Appendix I.

Baselines. For unconditional generation, we compare our model with recent baselines for synthetic time series generation. These include diffusion models: Diffusion-TS [12] and PaD-TS [30]; GANs: COSCI-GAN [14] and AEC-GAN [10]; and VAEs: TimeVQVAE [13] and Koopman VAE (Ko-VAE) [15]. Since most of these methods do not support conditional generation, we use three different baselines for the conditional setting. These include an older but influential model, WaveStitch [38], alongside two recent architectures: TabularARGN [47], an autoregressive network, and TimeAutoDiff [37], a latent diffusion model. A detailed introduction for each baseline is provided in Appendix D.

Table 3: Unconditional generation performance averaged over the three datasets (lower is better; best in **bold**). MD and CP are averaged across column pairs, and MAE uses standard-normalized values; PM is the $|\text{TSTR} - \text{TRTR}|$ MAE gap at prediction window W. The bottom row (Δbest baseline) reports, for each column, the relative error of FracTS against the best baseline, computed on these dataset-averaged raw values. Per-dataset results with standard deviations over three runs are reported in Appendix H.

	MD ↓						CP ↓		PM ↓		
Model	raw	mean	quantile	skewness	autocorr.	length	raw	agg.	W=10	W=100	W=1000
AEC-GAN	0.524	0.683	0.697	0.617	0.609	–	0.288	0.315	4 052	4 171	8 233
KoVAE	0.675	0.708	0.673	0.784	0.701	0.956	0.261	0.489	7 461	7 680	15 156
DiffusionTS	0.695	0.932	0.915	0.686	0.775	0.956	0.466	0.542	2 524	2 598	5 129
TimeVQVAE	0.634	0.726	0.700	0.496	0.732	0.956	0.183	0.443	6 822	7 022	13 861
COSCI-GAN	0.557	0.799	0.748	0.839	0.858	0.956	0.462	0.473	2 392	2 462	4 858
PaD-TS	0.412	0.495	0.513	0.618	0.714	0.956	0.305	0.470	3 841	3 954	7 804
FracTS (Ours)	**0.314**	**0.344**	**0.360**	**0.270**	**0.356**	**0.442**	**0.054**	**0.222**	**0.006**	**0.007**	**0.010**
Δbest baseline	23.87%	30.39%	29.89%	45.50%	41.52%	53.75%	70.44%	29.42%	100.00%	100.00%	100.00%

Evaluation Metrics. We evaluate the quality of synthetic time series data in three aspects: (i) *marginal distributions (MD)*: evaluated by Wasserstein distance, normalized by the standard deviation of the real data, simple aggregation (mean, std., quantiles (5%, 50%, 95%), skewness, auto-corr. (lag 1)) of each column, and the lengths respectively, adapted but augmented from common feature-based measures [48]; (ii) *correlation preservation (CP)*: evaluated by MAE on the correlation matrix, commonly used for multivariate time series generation [14], but extended beyond raw values to aggregated results similarly to (i); (iii) *predictive ML utility (PM)*: evaluated on the performance downstream application of next step prediction following the *TSTR* (Train on Synthetic, Test on Real) framework [48, 12], for which we report MAE (mean absolute error) scores. We additionally report a windowed discriminative evaluation (discriminative accuracy minus 0.5, lower is better) in Appendix F.3.

All metrics for unconditional generation are also applied for conditional generation, except that static attributes are taken into consideration for CP and PM. For simulated dataset in particular, the groundtruth (GT) time series can be calculated based on the generated static conditions according to their mathematical formulas. Thus, we also report the MAE (mean-absolute error) and MSE (mean-squared error) to the GT on this dataset. All results reported are based on three runs. More details for all metrics are provided in Appendix E.

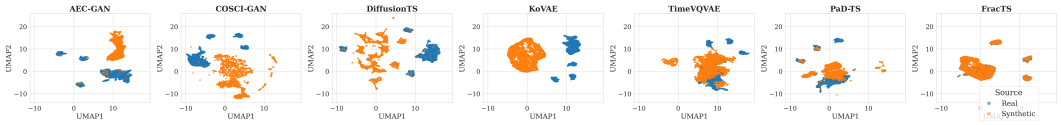


Figure 5: t-SNE of real compared to generated time series data of the Weather dataset.

5.2 Results on Unconditional Generation

As shown in Table 3, our model achieves the best scores on all metrics, with up to 53.8%, 70.4%, and 100.0% relative improvement from the best baseline in MD, CP, and PM, respectively (relative error computed on dataset-averaged raw values). Figure 5 visualizes the comparison of the outcomes from different models. The corresponding t-SNE visualization for the Sensors dataset is provided in Appendix F.1. This shows our model aligns well with the true distribution and works great for downstream tasks. Additional windowed t-SNE visualizations for all datasets are provided in Appendix F.1.

Notably, existing baselines do not support variable-length inputs; even with naïve zero-padding (Appendix D), they fail to learn meaningful length information and suffer from distribution distortion due to excessive padding. The gains on MD benefit from aggregated auxiliary features that better characterize data distributions, while improvements on CP stem from our fine-grained last level's learner and type-specific processing pipeline.

The superior predictive ML utility, across various window sizes show FracTS outstanding capability in learning long time series. While other models are still able to run, their performance degrades significantly compared to their performance on shorter time series. The exceptional performance of

Table 4: Conditional generation performance averaged over the two datasets (lower is better; best in **bold**). MD, CP, and GT are averaged across all column pairs, and MAE is computed on standard-normalized values; CP additionally accounts for the static conditions (agg.+cond.). PM is the $|\text{TSTR} - \text{TRTR}|$ MAE gap at prediction window W; GT reports the error against the ground truth of the simulated dataset. The bottom row (Δbest baseline) reports, for each column, the relative error of FracTS against the best baseline, computed on these dataset-averaged raw values. Per-dataset results with standard deviations over three runs are reported in Appendix H.

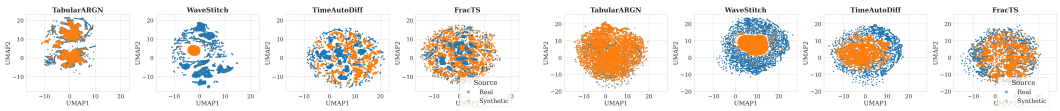
	MD ↓						CP ↓		PM ↓			GT ↓	
Model	raw	mean	quantile	skewness	autocorr.	length	raw	agg.+cond.	W=10	W=100	W=1000	MSE	MAE
TabularARGN	**0.154**	0.731	0.710	0.748	0.994	0.278	**0.069**	**0.345**	0.056	0.084	0.050	1.759	0.952
WaveStitch	0.506	0.916	0.890	0.842	0.996	0.445	0.146	0.500	0.059	0.030	0.057	1.233	0.884
TimeAutoDiff	0.353	0.769	0.782	0.797	0.898	0.445	0.228	0.421	0.406	0.420	0.392	144.260	5.103
FracTS (Ours)	0.172	**0.611**	**0.645**	**0.605**	**0.834**	**0.139**	0.074	0.409	**0.011**	**0.007**	**0.009**	**0.983**	**0.545**
Δbest baseline	−11.36%	16.42%	9.02%	19.17%	7.13%	50.00%	−7.30%	−18.72%	79.28%	75.00%	81.00%	20.28%	38.35%

FracTS on Stocks dataset further highlights that our model particularly shine when applied on long time series with abundant training data.

5.3 Results on Conditional Generation

FracTS also shows strong results for conditional generation, as shown in Table 4 and Figure 6. It outperforms the best baseline by up to 50.0%, 75.0–81.0% and 20.3–38.4% in MD, PM and GT, respectively (relative error computed on dataset-averaged raw values). For CP, FracTS and TabularARGN show competitive performance across different scores. Nevertheless, both methods consistently outperform the remaining baselines. On the three columns with a negative Δ (MD raw, CP raw and CP agg.+cond.), the two methods win on different datasets and their dataset-averaged scores differ only by 0.005 to 0.064, while FracTS is best or tied-best on all temporal and predictive columns on both datasets; Appendix H reports the per-dataset scores and standard deviations. We additionally provide windowed t-SNE visualizations for the conditional Walmart and Simulated datasets at window sizes $W \in \{10, 100, 1000\}$ in Appendix F.1.

The most important metric for simulated dataset is GT. We show the visualized samples of the simulated dataset in Appendix F.2. FracTS clearly learns these mathematical formulae better. This gain manifests the advantage of our data processing in both static and temporal features, and our model's outstanding ability in learning long temporal and static dependencies.



(a) Simulated **(b) Walmart**

Figure 6: t-SNE of real compared to generated time series data on conditional generation datasets.

6 Conclusion

In this paper, we introduced FracTS, a hierarchical and autoregressive generative model for time series. The model allows high-quality time series generation in a much less constrained setup than existing models in terms of the time series lengths and feature types and distributions. Building from a model originally designed for images, FracTS introduces significant adjustments for adaptation to time series domain, including a novel time series data processing pipeline optimized for the transformers backbone architecture, and correspondingly adjusted network structure. Extensive experiments across five datasets demonstrate that FracTS consistently outperforms seven baselines on distribution fidelity and downstream utility. We discuss the limitations of FracTS in Appendix J.

References

[1] Hugues Turbé, Mina Bjelogrlic, Christian Lovis, and Gianmarco Mengaldo. Evaluation of post-hoc interpretability methods in time-series classification. *Nat. Mac. Intell.*, 5(3): 250–260, 2023. doi: 10.1038/S42256-023-00620-W. URL https://doi.org/10.1038/s42256-023-00620-w.

[2] Michael Stenger, Robert Leppich, Ian T. Foster, Samuel Kounev, and André Bauer. Evaluation is key: A survey on evaluation measures for synthetic time series. *J. Big Data*, 11(1):66, 2024. doi: 10.1186/S40537-024-00924-7. URL https://doi.org/10.1186/s40537-024-00924-7.

[3] Jingyi Gu, Xuan Zhang, and Guiling Wang. Beyond the norm: A survey of synthetic data generation for rare events. *CoRR*, abs/2506.06380, 2025. doi: 10.48550/ARXIV.2506.06380. URL https://doi.org/10.48550/arXiv.2506.06380.

[4] Nadica Miljkovic, Nikola Milenic, Nenad B. Popovic, and Jaka Sodnik. Data augmentation for generating synthetic electrogastrogram time series. *Medical Biol. Eng. Comput.*, 62(9): 2879–2891, 2024. doi: 10.1007/S11517-024-03112-0. URL https://doi.org/10.1007/s11517-024-03112-0.

[5] Boris van Breugel, Nabeel Seedat, Fergus Imrie, and Mihaela van der Schaar. Can you rely on your model evaluation? Improving model evaluation with synthetic test data. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 1889–1904. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/05fb0f4e645cad23e0ab59d6b9901428-Paper-Conference.pdf.

[6] Alex Graves. Generating sequences with recurrent neural networks, 2014. URL https://arxiv.org/abs/1308.0850.

[7] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: a new algorithm for training recurrent networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, page 4608–4616, Red Hook, NY, USA, 2016. Curran Associates Inc. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/16026d60ff9b54410b3435b403afd226-Paper.pdf.

[8] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. In *9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, page 125, 2016.

[9] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[10] Lei Wang, Liang Zeng, and Jian Li. AEC-GAN: Adversarial error correction GANs for auto-regressive long time-series generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):10140–10148, Jun. 2023. doi: 10.1609/aaai.v37i8.26208. URL https://ojs.aaai.org/index.php/AAAI/article/view/26208.

[11] Chu-Cheng Lin, Aaron Jaech, Xin Li, Matthew R. Gormley, and Jason Eisner. Limitations of autoregressive models and their alternatives. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5147–5173, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.405. URL https://aclanthology.org/2021.naacl-main.405/.

[12] Xinyu Yuan and Yan Qiao. Diffusion-TS: Interpretable diffusion for general time series generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4h1apFjO99.

[13] Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 7665–7693. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/lee23d.html.

[14] Ali Seyfi, Jean-Francois Rajotte, and Raymond Ng. Generating multivariate time series with COmmon Source CoordInated GAN (COSCI-GAN). In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 32777–32788. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/d3408794e41dd23e34634344d662f5e9-Paper-Conference.pdf.

[15] Ilan Naiman, N. Benjamin Erichson, Pu Ren, Michael W. Mahoney, and Omri Azencot. Generative modeling of regular and irregular time series data via Koopman VAEs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=eY7sLb0dVF.

[16] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, page 3745–3753, Red Hook, NY, USA, 2016. Curran Associates Inc.

[17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hk99zCeAb.

[18] Zizhao Zhang, Yuanpu Xie, and Lin Yang. Photographic text-to-image synthesis with a hierarchically-nested adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[19] Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19667–19679. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e3b21256183cf7c2c7a66be163579d37-Paper.pdf.

[20] Seung Wook Kim, Bradley Brown, Kangxue Yin, Karsten Kreis, Katja Schwarz, Daiqing Li, Robin Rombach, Antonio Torralba, and Sanja Fidler. NeuralField-LDM: Scene generation with hierarchical latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8496–8506, June 2023.

[21] Tianhong Li, Qinyi Sun, Lijie Fan, and Kaiming He. Fractal generative models, 2025. URL https://arxiv.org/abs/2502.17437.

[22] Paul Jeha, Michael Bohlke-Schneider, Pedro Mercado, Shubham Kapoor, Rajbir Singh Nirwan, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. PSA-GAN: Progressive self attention GANs for synthetic time series. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Ix_mh42xq5w.

[23] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI Technical Report*, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[24] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/254ed7d2de3b23ab10936522dd547b78-Paper.pdf.

[25] Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. TabDiff: a mixed-type diffusion model for tabular data generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=swvURjrt8z.

[26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[27] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[30] Yang Li, Han Meng, Zhenyu Bi, Ingolv T. Urnes, and Haipeng Chen. Population aware diffusion for time series generation. In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press, 2025. ISBN 978-1-57735-897-8. doi: 10.1609/aaai.v39i17.34038. URL https://doi.org/10.1609/aaai.v39i17.34038.

[31] Cristian I. Challu, Peihong Jiang, Ying Nian Wu, and Laurent Callot. Deep generative model with hierarchical latent factors for time series anomaly detection. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 1643–1654. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/challu22a.html.

[32] Zhiding Liu, Jiqian Yang, Mingyue Cheng, Yucong Luo, and Zhi Li. Generative pretrained hierarchical transformer for time series forecasting. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 2003–2013, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671855. URL https://doi.org/10.1145/3637528.3671855.

[33] Shubao Zhao, Ming Jin, Zhaoxiang Hou, Chengyi Yang, Zengxiang Li, Qingsong Wen, and Yi Wang. HiMTM: Hierarchical multi-scale masked time series modeling with self-distillation for long-term forecasting. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 3352–3362, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704369. doi: 10.1145/3627673.3679741. URL https://doi.org/10.1145/3627673.3679741.

[34] Hung H. Bui, Dinh Q. Phung, and Svetha Venkatesh. Hierarchical hidden markov models with general state hierarchy. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, page 324–329. AAAI Press, 2004. ISBN 0262511835.

[35] Zhengping Che, Sanjay Purushotham, Guangyu Li, Bo Jiang, and Yan Liu. Hierarchical deep generative models for multi-rate multivariate time series. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 784–793. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/che18a.html.

[36] Xiangyu Xu, Qingsong Zhong, and Jilin Hu. Timemar: Multi-scale autoregressive modeling for unconditional time series generation. In *Proceedings of the ACM Web Conference 2026*, pages 5132–5143, 2026.

[37] Namjoon Suh, Yuning Yang, Din-Yin Hsieh, Qitong Luan, Shirong Xu, Shixiang Zhu, and Guang Cheng. TimeAutoDiff: A unified framework for generation, imputation, forecasting, and time-varying metadata conditioning of heterogeneous time series tabular data. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=bkUd1Dg46c. J2C Certification.

[38] Aditya Shankar, Lydia Y. Chen, Arie van Deursen, and Rihan Hai. Wavestitch: Flexible and fast conditional time series generation with diffusion models, 2025. URL https://arxiv.org/abs/2503.06231.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[40] Jiayu Li, Bingyin Zhao, Zilong Zhao, Uzair Javaid, Kevin Yee, and Biplab Sikdar. TabTreeFormer: Tabular data generation using hybrid tree-transformer, 2025. URL https://arxiv.org/abs/2501.01216.

[41] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4651–4664. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/jaegle21a.html.

[42] Arunava Kr. Chakraborty. Australia weather data. https://www.kaggle.com/datasets/arunavakrchakraborty/australia-weather-data, 2023. Data originally sourced from the Australian Bureau of Meteorology (http://www.bom.gov.au/climate/data/). Accessed: 2025-07-03.

[43] Iman Almomani, Bassam Al-Kasasbeh, and Mousa Al-Akhras. WSN-DS: A dataset for intrusion detection systems in wireless sensor networks. *Journal of Sensors*, 2016, 2016. Dataset available at: https://www.kaggle.com/datasets/bassamkasasbeh1/wsnds.

[44] Boris Marjanovic. Huge stock market dataset. https://www.kaggle.com/datasets/borismarjanovic/price-volume-data-for-all-us-stocks-etfs?select=Stocks, 2018. Accessed: 2025-07-03.

[45] Aslan Ahmedov. Walmart sales forecast. https://www.kaggle.com/datasets/aslanahmedov/walmart-sales-forecast?select=stores.csv, 2021. Accessed: 2025-10-15.

[46] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[47] Paul Tiwald, Ivona Krchova, Andrey Sidorenko, Mariana Vargas Vieyra, Mario Scriminaci, and Michael Platzer. TabularARGN: A flexible and efficient auto-regressive framework for generating high-fidelity synthetic data, 2025. URL https://arxiv.org/abs/2501.12012.

[48] Yihao Ang, Qiang Huang, Yifan Bao, Anthony K. H. Tung, and Zhiyong Huang. TSGBench: Time series generation benchmark. *Proc. VLDB Endow.*, 17(3):305–318, November 2023. ISSN 2150-8097. doi: 10.14778/3632093.3632097. URL https://doi.org/10.14778/3632093.3632097.

[49] J. R. Klauder, A. C. Price, S. Darlington, and W. J. Albersheim. The theory and design of chirp radars. *The Bell System Technical Journal*, 39(4):745–808, 1960. doi: 10.1002/j.1538-7305.1960.tb03942.x.

[50] Reginald A. Fessenden. Wireless telephony. *Transactions of the American Institute of Electrical Engineers*, XXVII(Part 1):553–629, June 29 1908.

[51] D. Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93:429–441, 1946. doi: 10.1049/ji-3-2.1946.0074. URL https://digital-library.theiet.org/doi/abs/10.1049/ji-3-2.1946.0074.

[52] Hengrui Zhang, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4Ay23yeuz0.

[53] Kyle Nickerson, Terrence Tricco, Antonina Kolokolova, Farzaneh Shoeleh, Charles Robertson, John Hawkin, and Ting Hu. Banksformer: A deep generative model for synthetic transaction sequences. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part VI*, page 121–136, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-26421-4. doi: 10.1007/978-3-031-26422-1_8. URL https://doi.org/10.1007/978-3-031-26422-1_8.

A More Results of Step-wise Differences versus Raw Values

In Figure 4a, we show the Q-Q plots for the normal distribution of one feature from three datasets each, and for all three features, the distribution of differences is closer to a normal distribution than raw values. These are not individual cases, but a common phenomenon in many datasets. In this section, we show the Q-Q plots of more features from more datasets. Details of datasets will be introduced with experiments, or found in Appendix B.

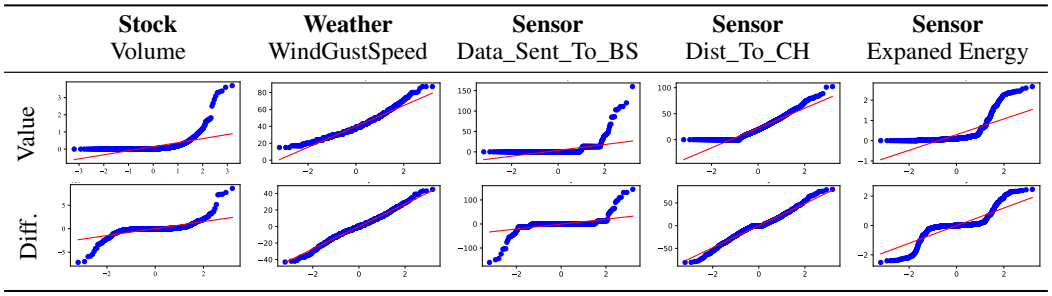


Figure 7: Comparison to normal distribution of the distribution of raw values and differences of consecutive timesteps in Q-Q plots of more features from real-life time series datasets.

In Figure 7, we show more features from the realistic datasets we experimented with. Highly synchronized features are omitted, such as close and high for stocks, as open is already presented. It is clearly shown that the distributions of differences are closer to a normal distribution than the raw values. Indeed, there are cases where neither differences nor raw values are close to normal distribution, computing differences at least transforms the one-sided tailed distributions to two-sided tailed distributions, which is closer to the nature of normal distributions. The distribution of differences is at least not farther from a normal distribution than the distribution of raw values.

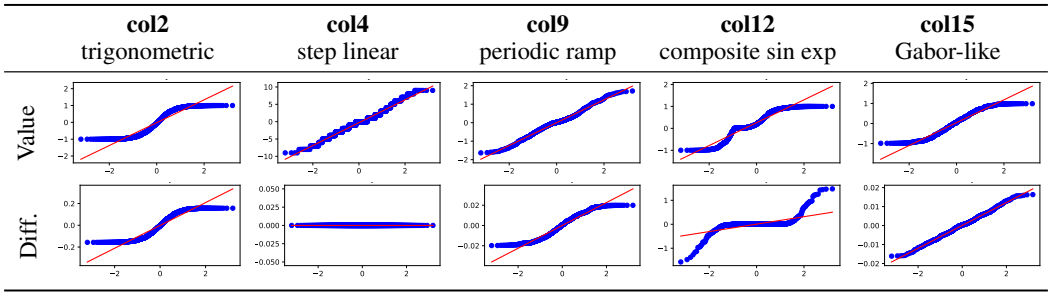


Figure 8: Comparison to normal distribution of the distribution of raw values and differences of consecutive timesteps in Q-Q plots of simulated features from mathematical formulae.

Analyzing the distribution of differences versus values based on time series simulated from mathematical formulae, the conclusion above still holds in most cases, as shown in Figure 8, except for very few exceptional cases, such as periodic ramp signals. Nevertheless, periodic ramp signals' step-wise differences have two major clusters for intra- and inter-period changes. If one observes the two groups separately, the distribution of differences would still be more standard than raw values. Such clustering does not exist for raw values. Given the capability of neural networks in classification tasks, it is believed that a well-designed and well-trained neural network is capable of telling the two clusters above, and hence is likely to learn the combination of the two clusters of close-to-normal distributions, better than one cluster of non-normal distribution.

B Dataset Setup

B.1 Overall Setup

For all datasets, to ensure a fair and efficient comparison with baseline models, we use curated subsets of each dataset. We remove columns with the highest missingness and, for baselines that do not

natively support categorical features, we encode those columns during preprocessing. This yields a consistent input space across datasets with varying domains, sequence lengths, and complexity while preserving realistic data characteristics. All datasets are split at the series level into $8:2$ training and test set.

B.2 Real-world Datasets Details

- **Weather Dataset:** The weather dataset [42] contains the weather information, including temperature, humidity, wind speed, and pressure, of various places. The primary challenge lies in learning from very long sequences despite having a limited number of examples, making it essential for models to capture long-range temporal dependencies effectively.
- **Sensor Dataset:** The sensor dataset [43] includes 17 sensor readings related to either industrial equipment monitoring or cybersecurity scenarios. The main difficulty here is ensuring generalization across a large number of sequences with potentially unseen or highly diverse behaviors.
- **Stocks Dataset:** The stocks dataset [44] features a variety of financial indicators such as stock prices, trading volume, and market volatility. The key challenge is dealing with the inherent noise, irregularity, and rapid changes that are characteristic of financial time series.
- **Walmart Dataset:** The Walmart dataset [45] contains 2 static features (Type and Size) and 3 time-dependent variables capturing store-level sales statistics. As a conditional dataset, it requires models to leverage both static contextual information and dynamic temporal patterns. The primary challenge lies in effectively integrating static and sequential features over relatively short horizons while maintaining the ability to generalize across diverse departments and store profiles.

Due to the limited number of data points for Weather dataset, we use a different data split $(7:3)$ for training and testing, to ensure sufficient test samples for reliable evaluation.

B.3 Simulated Data Configuration

The four static features and their corresponding sampling strategies are w_1, w_2, b, k, all uniformly sampled from $[-1, 1]$.

These condition values are the variables of the mathematical expressions. Time series features are generated using the formulas listed in Table 5, where x indicates the time step.

C FracTS Implementation Details

C.1 Details of Data Processing

Data Requirements. In this paper, we focus on clean tabular data. Missing values, mixed numerical and categorical features, and data types other than categorical, numerical, and datetime, such as textual, audio, and vision data, are not included.

Encoding of Numerical Data. All numerical data are standardized by their mean and standard deviation before being passed to the networks. When numerical data are binned, we use 20 bins by default. Both data processors are implemented using `scikit-learn` preprocessing module.

Datetime Components. Datetime features are often omitted or simply treated as numeric values using Unix timestamps in most tabular models [24, 52, 40]. However, this is not the optimal solution for time series data, as datetimes are typically critical to time series [53] especially for event-based time series with event dates. Thus, prior to the numeric value representation of the datetime value, we put some important datetime components, such as month and day of week, to guide the generation of actual values, in addition to the bin of numeric values.

Datetime components we use include: year (integer), month (integer), day in month (integer), hour (integer), minute (integer), second (integer), millisecond (integer), microsecond (integer), nanosecond (integer), month name (categorical), day of week (integer), day name of week (categorical), days to month end (integer), AM vs. PM (categorical), whether this is month start (categorical), whether this

Table 5: Generating formulas of the 15 time series features of the simulated dataset. Each feature is a function of the time step x and the static conditions w_1, w_2, b, k.

col1: Linear trend $0.001kx + b$	**col2:** Cosine wave with phase shift $\cos(0.05\pi x + b)$
col3: Alternating constant pattern $\left\lfloor \frac{x+\lfloor 25b+0.5 \rfloor}{100} \right\rfloor \bmod 2$	**col4:** Rounded step function $\lfloor -0.01w_1 x + 0.5 \rfloor$
col5: Convex curve $\sqrt{0.01(w_1+1)x + 10 + b}$	**col6:** Non-convex shape (tanh function) $\tanh\left(5k\left(\frac{x}{L_n} - 0.5\right)\right)$
col7: Linear and sinusoidal combination $0.001w_1 x + w_2 + \sin(0.03\pi x - 0.05)$	**col8:** Cosine wave with exponential decay $\exp(-0.001x)\cos(0.003x + b)$
col9: Periodic ramp signal $0.02 \cdot w_2 \cdot (x \bmod 100)$	**col10:** Piecewise exponential and linear function $\begin{cases} \exp(0.005 \cdot x), & \text{if } x < L_n/2 \\ 0.003 \cdot w_2 \cdot x, & \text{otherwise} \end{cases}$
col11: Conditional cosine or sine $\begin{cases} \cos(0.02\pi x + w_1), & \text{if } b \geq 0 \\ \sin(0.02\pi x + w_2), & \text{if } b < 0 \end{cases}$	**col12:** Composite function with exponential inside sine $\sin(\exp(0.01kx))$
col13: Chirp signal [49] with combined frequency growth $\sin\left(2\pi(0.003kx + (0.005kx)^2)\right)$	
col14: Amplitude modulated signal [50] $\left(1 + 0.5\sin\left(\frac{w_1\pi x}{500}\right)\right) \cdot \sin\left(\frac{\pi w_1 x}{50}\right)$	
col15: Gabor-like modulated Gaussian signal [51] $\exp\left(-\frac{k^2x^2}{500000}\right) \cdot \sin\left(\frac{kx}{50}\right)$	

is month end (categorical), whether this is holiday (categorical). For components that do not have different values throughout the entire dataset in the current feature, the component is dropped. For example, a column up to the precision of day will have any component of higher precision, like hour, minute, and second, dropped.

While the exact datetime value can be directly inferred from the numeric representation, it is worthwhile to adapt the datetime value based on the components predicted, due to the limited precision the numeric predictions can achieve. For example, if a date is predicted to be a month's start but the date generated is Sep. 2, then we find the closest month start date Sep. 1, to replace the day of month from 2 to 1.

Aggregation Functions. We first extract the first value of all features in the aggregated auxiliary static features. For categorical features, we further collect the top-frequency category and the proportion of at most 10 globally most frequent categories. For numerical features, we collect the mean and standard deviation as additional aggregated information. For datetime features, we only collect aggregated information on the day name of a week and the month name.

C.2 Experimental Setup

All experiments were run on NVIDIA H100 GPUs using PyTorch with mixed precision training. Each result is reported as the average of three runs.

We use the same transformer model (GPT-2 small) for all datasets to keep the setup consistent and comparable. GPT-2 small has an embedding size of 768, 12 attention heads, and 12 transformer blocks. For the static features, we generate all samples using CTGAN [24]. This ensures diverse and realistic static conditions while keeping the process simple and consistent across datasets.

For the simulated dataset specifically, we disabled all data aggregations due to GPU memory limits. We also skipped the condition compression module for this dataset. Since it only contains 4 conditions, the compression step was unnecessary.

Training is done with the AdamW optimizer using a learning rate of 2×10^{-4}. We apply a warmup of 10 epochs and train for a total of 200 epochs. Validation is performed after every epoch, and the best model is selected based on the lowest validation loss.

The convergence dynamics of our models are illustrated in Figure 9 and Figure 10, which displays the training and validation losses across all five datasets. To provide a comprehensive view of the computational requirements, we detail the total trainable parameters and the average training time for each dataset in Table 6.

Table 6: Model size and average training time per dataset.

Dataset	Params (M)	Time (h)
Weather	729.8	1.1
Sensor	432.8	4.6
Stocks	617.8	7.3
Walmart	1147.3	1.8
Simulated	732.5	3.7

Training, inference, and memory cost. To further characterize computational efficiency, we compare the training and inference time of FracTS with the most competitive baseline used for each of the Weather and Walmart datasets, and measure the GPU memory consumption of FracTS on an NVIDIA H100 GPU with 80 GB of memory (Tables 7, 7a and 7b). FracTS incurs a higher training cost than the compared baselines. For inference, PaD-TS is faster on Weather, while FracTS is faster than TabularARGN on the substantially longer Walmart sequences.

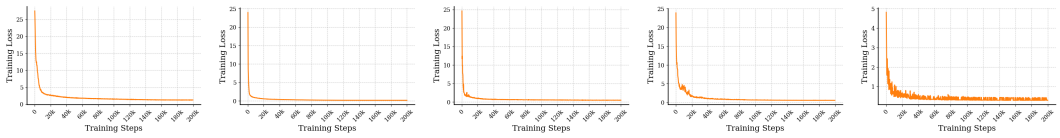
Table 7: Computational cost of FracTS on the two datasets used in the efficiency analysis. Memory is measured on an NVIDIA H100 GPU with 80 GB. Training batch sizes are selected for data throughput, so the memory figures are not a controlled comparison against baselines with different batch-size configurations.

(a) Training and inference time.

Dataset	Model	Train (s)	Inf. (s)
Weather	FracTS	4068	240
Weather	PaD-TS	1560	10
Walmart	FracTS	6300	241
Walmart	TabularARGN	240	300

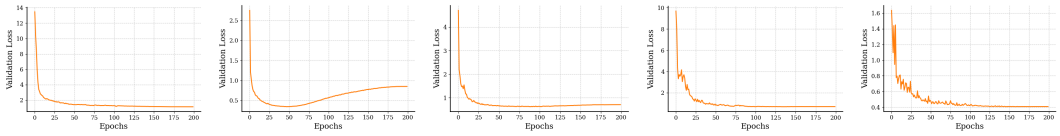
(b) GPU memory consumption.

Dataset	Len. L	Train (GB)	Inf. (GB)
Weather	2171	73.0	3.5
Walmart	10474	77.0	5.5



(a) Weather **(b)** Sensors **(c)** Stocks **(d)** Walmart **(e)** Simulated

Figure 9: Training loss curves across the five evaluated datasets. The x-axis represents the training steps, demonstrating that the models consistently converge over the course of the training regime.



(a) Weather **(b)** Sensors **(c)** Stocks **(d)** Walmart **(e)** Simulated

Figure 10: Validation loss curves across the five evaluated datasets. The x-axis represents the training epochs, illustrating that the models consistently converge within the 200-epoch training regime.

D Baselines

Following gives the detailed introduction of each baseline.

Generation Length. FracTS generates the entire time series and supports variable and long generation lengths without truncation. For baselines that require fixed-length inputs, we pad sequences to the maximum sequence length.

By default, the baseline models construct training samples by forming intervals or overlapping segments from the given time series. However, since our dataset consists of distinct and independent time series, we generally treat each complete time series as a single window during training. To ensure consistent input lengths across samples, shorter series are zero-padded to match the maximum sequence length. After training, we applied a depadding step using a threshold of 1×10^{-2} to remove the trailing padded values. The only exception to this approach is AECGAN; because its algorithm cannot function when an entire time series is treated as a single window, we split each time series into shorter windows specifically for this model. For all baselines, we used their official implementations with default hyperparameters and settings, modifying only the preprocessing pipeline to adapt to these configurations.

- **Diffusion-TS** [12]: A diffusion-based generative model that uses an encoder-decoder Transformer for high-quality multivariate time series generation. It reconstructs the sample at each denoising step and applies a Fourier-based loss to enhance the quality and interpretability of generated data.
- **TimeVQVAE** [13]: This method uses a vector-quantized variational autoencoder for time series. TimeVQVAE learns discrete representations and models temporal dependencies with a Transformer prior, capturing both long-range and fine-grained temporal patterns.
- **COSCI-GAN** [14]: A GAN-based model designed for multivariate time series from common sources, such as multi-channel sensor data. COSCI-GAN uses a central discriminator to maintain correlations between different channels and preserve realistic dynamics.
- **AEC-GAN** [10]: An adversarial model that includes an error-correction mechanism to handle long-term dependencies and avoid drift in synthetic sequences. We defined the past (p) and future (q) sequence lengths based on the original experimental settings present in their official code. Specifically, we set $p = 96$ and $q = 192$ for the Stock dataset, $p = 18$ and $q = 36$ for Sensor, and $p = 168$ and $q = 336$ for Weather.
- **Koopman VAE (KoVAE)** [15]: A variational autoencoder that uses a linear Koopman operator in the latent space to model time series dynamics. KoVAE supports both regular and irregular time intervals and incorporates domain knowledge through its spectral constraints.
- **WaveStitch** [38]: WaveStitch is a conditional time series generation model based on denoising diffusion probabilistic models (DDPMs), which conditions on both static metadata and partially observed signals to flexibly adapt to unseen constraints. To incorporate these static conditioning attributes, we one-hot encode the categorical metadata, tile it across all timesteps, and concatenate it with the time series. Each complete series is treated as a single, fixed-length window via padding or truncation. Crucially, both the forward noising and reverse denoising diffusion processes are applied exclusively to the time series features, while the metadata columns remain clamped to serve as a constant conditioning context.
- **TabularARGN** [47]: A flexible auto-regressive generative framework for mixed-type, multivariate, and sequential data that factors the joint distribution into conditional probabilities and supports efficient conditional generation.
- **PaD-TS** [30]: A diffusion-based model that focuses on capturing the patterns of the entire dataset instead of just individual samples. It uses a population-aware training method to ensure the generated data preserves the global statistics and correlations of the real data.
- **TimeAutoDiff** [37]: A unified framework that combines a VAE and latent diffusion to handle time series with both continuous and categorical features. It compresses the data into a latent space first to effectively generate complex, mixed-type sequences. TimeAutoDiff expects each time series as a separate sample, with static conditioning attributes (e.g., simulation parameters or store metadata) broadcast across all timesteps and concatenated with the time-varying columns.

E Evaluation Metric Implementation

Setup: We evaluate quality by comparing many independent real time series (e.g., different stocks or sensors readings) to an equally sized set of synthetic series generated by our model. Each series is a CSV with one or more columns (variables). For every column we extract simple, model-agnostic features that summarize distributional and temporal behavior: *mean*, *std*, *skew*, *quantile*, *length* and *autocorrelation at lag 1*. Aggregating these features across all files gives us feature distributions for the real set and the synthetic set, which are then compared using the KS test.

Ground-truth (simulation): For simulated data with known generator $\mathcal{G}$, we compare reference series $\mathbf{y}^\star = \mathcal{G}(\mathbf{c}_{\text{new}})$ to model output $\hat{\mathbf{y}} \sim p(\cdot \mid \mathbf{c}_{\text{new}})$ via:

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N} \left|\hat{y}_t - y_t^\star\right|, \qquad \text{MSE} = \frac{1}{N}\sum_{t=1}^{N} (\hat{y}_t - y_t^\star)^2.$$

Utility Testing: For the downstream performance evaluation, we used an LSTM network for time series prediction. To measure how useful the generated data is, we evaluated the models using two settings: Train on Synthetic, Test on Real (TSTR) and Train on Real, Test on Real (TRTR). The LSTM was trained to predict the next time step for all columns simultaneously. We used a sliding window approach with window sizes of 10, 100, and 1000 time steps to test both short- and long-term dependencies. The raw values were used as input features to the network. Finally, we calculated the Mean Absolute Error (MAE) for both settings and reported the absolute gap between their performances ($|\text{TSTR} - \text{TRTR}|$). A smaller gap shows that the synthetic data successfully preserves the predictive ML utility of the real data.

Discriminative Evaluation: Standard sequence-level discriminative evaluation assumes fixed-length inputs and therefore cannot be directly applied to the variable-length sequences generated in our setting. To enable this evaluation, we use fixed-length windows of $W \in \{10, 100, 1000\}$ sampled from real and synthetic sequences and train a discriminative classifier to distinguish between them. Following the standard discriminative-score formulation used in our evaluation, we report discriminative accuracy minus 0.5, where lower values are better and 0 represents ideal indistinguishability.

F Supplementary Experiment Results

F.1 Windowed t-SNE Visualizations

The t-SNE visualizations in the main paper operate at the individual time-step level. To further assess whether the generated data preserve temporal structure over different horizons, we additionally perform windowed t-SNE using fixed-length windows of $W \in \{10, 100, 1000\}$. Figures 11, 12 and 13 show the results for the unconditional datasets and the conditional Walmart and Simulated datasets, respectively. Figure 14 additionally shows the step-level t-SNE comparison between real and generated data on the Sensors dataset.

F.2 More Visualized Results on Realistic Datasets

Table 8 provides a per-column dissection of model performance on the simulated dataset. FracTS demonstrates clear overall superiority, outperforming the baselines in 11 out of the 15 columns. The gains are especially pronounced in capturing fundamental temporal structures, ranging from linear and standard periodic patterns (Cols 1–2) to highly nonlinear or piecewise non-stationary signals. For example, on the alternating constant pattern (Col 3) and the piecewise exponential/linear function (Col 10), FracTS achieves errors that are significantly lower than both WaveStitch and TabularARGN. However, we observe that for highly complex composite and oscillatory signals involving frequency growth or heavy amplitude modulation (Cols 12–15, such as the Chirp and Gabor-like signals), FracTS struggles to maintain this lead, and baselines like WaveStitch perform better. These highly modulated signals are defined by complex analytical formulas that are difficult to recover exactly from data alone. In these cases, WaveStitch produces highly fluctuating outputs that can obtain lower pointwise MAE/MSE values. FracTS produces more realistic overall signal shapes, but when it does

not recover the exact underlying modulation, the generated sequence can drift from the target over longer horizons, resulting in larger MAE/MSE values. Despite this limitation on heavily modulated signals, FracTS provides the most accurate and robust generation overall. On average, FracTS achieves an MAE of 0.545 and an MSE of 0.983, substantially outperforming the closest baselines, WaveStitch (0.884 / 1.233) and TabularARGN (0.952 / 1.759). Figure 15 shows generated versus ground-truth examples for six representative columns.

Table 8: Per-column MAE/MSE against ground truth on the simulated dataset (lower is better; best in **bold**, second best underlined). Column numbers correspond to the generating formulas in Table 5.

	FracTS (Ours)		**TabularARGN**		**WaveStitch**		**TimeAutoDiff**	
Pattern	MAE ↓	MSE ↓	MAE ↓	MSE ↓	MAE ↓	MSE ↓	MAE ↓	MSE ↓
1. Linear trend	$\mathbf{0.525}_{\pm 0.102}$	$\mathbf{0.590}_{\pm 0.250}$	$0.963_{\pm 0.015}$	$1.547_{\pm 0.059}$	$0.885_{\pm 0.083}$	$1.232_{\pm 0.213}$	$0.896_{\pm 0.055}$	$1.466_{\pm 0.208}$
2. Cosine wave	$\mathbf{0.292}_{\pm 0.047}$	$\mathbf{0.166}_{\pm 0.055}$	$1.144_{\pm 0.001}$	$1.993_{\pm 0.001}$	$0.914_{\pm 0.097}$	$1.312_{\pm 0.258}$	$3.298_{\pm 0.526}$	$14.337_{\pm 3.971}$
3. Alternating constant	$\mathbf{0.168}_{\pm 0.027}$	$\mathbf{0.335}_{\pm 0.055}$	$1.272_{\pm 0.125}$	$2.547_{\pm 0.249}$	$0.881_{\pm 0.055}$	$1.217_{\pm 0.145}$	$1.158_{\pm 0.102}$	$2.696_{\pm 0.484}$
4. Rounded step	$\mathbf{0.467}_{\pm 0.057}$	$\mathbf{0.428}_{\pm 0.104}$	$0.838_{\pm 0.057}$	$1.240_{\pm 0.157}$	$0.967_{\pm 0.106}$	$1.453_{\pm 0.283}$	$1.444_{\pm 0.062}$	$3.195_{\pm 0.232}$
5. Convex curve	$\mathbf{0.157}_{\pm 0.223}$	$\mathbf{0.139}_{\pm 0.197}$	$1.024_{\pm 0.008}$	$1.609_{\pm 0.034}$	$0.870_{\pm 0.057}$	$1.187_{\pm 0.135}$	$1.506_{\pm 0.029}$	$3.224_{\pm 0.075}$
6. Non-convex (tanh)	$\mathbf{0.398}_{\pm 0.057}$	$\mathbf{0.299}_{\pm 0.050}$	$1.040_{\pm 0.019}$	$1.625_{\pm 0.068}$	$0.896_{\pm 0.086}$	$1.257_{\pm 0.224}$	$11.625_{\pm 6.562}$	$251.719_{\pm 214.347}$
7. Linear + sinusoid	$\mathbf{0.478}_{\pm 0.054}$	$\mathbf{0.431}_{\pm 0.100}$	$0.928_{\pm 0.041}$	$1.482_{\pm 0.124}$	$0.957_{\pm 0.072}$	$1.425_{\pm 0.188}$	$1.540_{\pm 0.081}$	$3.579_{\pm 0.355}$
8. Cosine + exp. decay	$\mathbf{0.120}_{\pm 0.027}$	$\mathbf{0.130}_{\pm 0.040}$	$0.553_{\pm 0.008}$	$1.460_{\pm 0.008}$	$0.817_{\pm 0.011}$	$1.053_{\pm 0.029}$	$7.478_{\pm 0.952}$	$455.942_{\pm 125.824}$
9. Periodic ramp	$\mathbf{0.462}_{\pm 0.105}$	$\mathbf{0.433}_{\pm 0.206}$	$0.925_{\pm 0.005}$	$1.438_{\pm 0.012}$	$0.917_{\pm 0.085}$	$1.315_{\pm 0.228}$	$1.362_{\pm 0.091}$	$3.528_{\pm 0.523}$
10. Piecewise exp./linear	$\mathbf{0.096}_{\pm 0.018}$	$\mathbf{0.594}_{\pm 0.228}$	$0.282_{\pm 0.001}$	$1.580_{\pm 0.033}$	$0.847_{\pm 0.046}$	$1.152_{\pm 0.126}$	$3.079_{\pm 0.686}$	$154.752_{\pm 52.485}$
11. Conditional cos/sin	$\mathbf{0.497}_{\pm 0.088}$	$\mathbf{0.505}_{\pm 0.172}$	$1.098_{\pm 0.031}$	$1.874_{\pm 0.080}$	$0.969_{\pm 0.161}$	$1.457_{\pm 0.439}$	$1.139_{\pm 0.119}$	$2.083_{\pm 0.431}$
12. Composite sin–exp	$0.873_{\pm 0.072}$	$1.421_{\pm 0.109}$	$1.053_{\pm 0.060}$	$1.969_{\pm 0.160}$	$\mathbf{0.821}_{\pm 0.014}$	$\mathbf{1.060}_{\pm 0.035}$	$13.553_{\pm 3.379}$	$383.759_{\pm 168.309}$
13. Chirp	$1.179_{\pm 0.059}$	$2.132_{\pm 0.236}$	$1.150_{\pm 0.005}$	$2.003_{\pm 0.019}$	$\mathbf{0.824}_{\pm 0.013}$	$\mathbf{1.068}_{\pm 0.032}$	$1.525_{\pm 0.361}$	$3.517_{\pm 1.548}$
14. Amplitude modulated	$1.288_{\pm 0.139}$	$2.454_{\pm 0.465}$	$1.146_{\pm 0.001}$	$1.980_{\pm 0.005}$	$\mathbf{0.829}_{\pm 0.017}$	$\mathbf{1.082}_{\pm 0.038}$	$23.981_{\pm 5.798}$	$844.979_{\pm 353.043}$
15. Gabor-like	$1.184_{\pm 0.124}$	$4.688_{\pm 0.244}$	$\mathbf{0.875}_{\pm 0.071}$	$2.052_{\pm 0.237}$	$0.883_{\pm 0.079}$	$\mathbf{1.238}_{\pm 0.205}$	$2.963_{\pm 0.437}$	$35.127_{\pm 12.562}$
Average	$\mathbf{0.545}_{\pm 0.051}$	$\mathbf{0.983}_{\pm 0.109}$	$0.952_{\pm 0.034}$	$1.759_{\pm 0.034}$	$0.884_{\pm 0.040}$	$1.233_{\pm 0.109}$	$5.103_{\pm 2.088}$	$144.260_{\pm 103.107}$

F.3 Discriminative Evaluation

Table 9 reports the windowed discriminative evaluation described in Appendix E. Table 9a compares FracTS with PaD-TS on Weather and TabularARGN on Walmart, and Table 9b repeats the comparison on feature subsets of Weather. Lower values indicate that real and synthetic samples are more difficult to distinguish.

Table 9: Windowed discriminative evaluation, measured as $\mathrm{Accuracy} - 0.5$ (lower is better, with 0 indicating ideal indistinguishability). (a) compares FracTS against the strongest baseline per dataset; (b) isolates the effect of the feature subset used for evaluation on Weather while keeping the generator fixed.

(a) FracTS versus the strongest baseline per dataset.

Dataset	**Model**	$W{=}10$	$W{=}100$	$W{=}1000$
Weather	**FracTS**	$\mathbf{0.342}_{\pm 0.021}$	$\mathbf{0.425}_{\pm 0.025}$	$\mathbf{0.283}_{\pm 0.052}$
Weather	PaD-TS	$0.418_{\pm 0.034}$	$0.431_{\pm 0.036}$	$0.423_{\pm 0.021}$
Walmart	**FracTS**	$\mathbf{0.064}_{\pm 0.056}$	$\mathbf{0.118}_{\pm 0.113}$	$\mathbf{0.170}_{\pm 0.125}$
Walmart	TabularARGN	$0.266_{\pm 0.033}$	$0.331_{\pm 0.153}$	$0.310_{\pm 0.157}$

(b) Feature-subset study on Weather.

Evaluation setting	$W{=}10$	$W{=}100$	$W{=}1000$
All features (17)	0.342	0.425	0.283
Remove 5 worst-performing features (12)	0.296	0.364	0.290
Retain 5 best-performing features (5)	0.130	0.175	0.215

FracTS achieves lower discriminative scores than the corresponding competitive baselines across all evaluated window sizes on both Weather and Walmart. Although FracTS achieves lower discriminative scores than the compared baselines, the absolute scores on Weather indicate that generated and real samples remain distinguishable under this evaluation. For example, a score of 0.425 at $W = 100$ corresponds to a discriminator accuracy of 0.925. We therefore additionally examine how this behavior changes across different subsets of the Weather features.

Effect of dimensionality on discriminative evaluation. To further investigate the relatively high discriminative score on the 17-feature Weather dataset, we evaluate the same generated output while

varying only the subset of features used for discriminative evaluation. The generator remains trained on all 17 features.

Removing only five of the worst-performing features reduces the discriminative score at $W = 100$ from 0.425 to 0.364, while retaining only the five best-captured features reduces it further to 0.175. This indicates that the discriminative score on the high-dimensional Weather dataset is strongly influenced by a subset of challenging features. Theorem F.1 formalizes this dimensionality effect.

Theorem F.1. *Suppose a multivariate time series dataset exhibits n mutually independent core properties (e.g., marginal distributions, feature correlations, autocorrelation structures), a discriminator D flags a property as real with probability p_d when the property is faithfully present and with probability $1 - p_d$ otherwise, and a generator G reproduces each property faithfully with probability p_g. Under independence across properties and a balanced real/synthetic test set, the expected discriminative accuracy is*

$$\mathbb{E}[A(n)] = \tfrac{1}{2} + \tfrac{1}{2}\, p_d^n - \tfrac{1}{2}\left(p_g p_d + (1 - p_g)(1 - p_d)\right)^n . \tag{4}$$

For any generator with imperfect fidelity ($p_g < 1$) and any non-random discriminator ($p_d > 1/2$), $\mathbb{E}[A(n)] - \frac{1}{2}$ is strictly positive for all n, and $\mathbb{E}[A(n)]$ converges to $\frac{1}{2}$ exponentially fast as n grows.

G Theoretical Analysis

Theorem 4.1. *For a sequence of total length L generated by a K-level hierarchical autoregressive architecture with branching factor l (where $K = \log_l L$), if the local parameter estimation error per step is δ, the total KL divergence between the ground-truth distribution $p(X_{\cdot,1:L})$ and the generated distribution $p(\hat{X}_{\cdot,1:L})$ satisfies:*

$$D_{KL}\left(p(X_{\cdot,L} \mid c) \| p(\hat{X}_{\cdot,L} \mid c)\right) = \Omega(\delta l K).$$

Proof of Theorem 4.1. According to Wang et al. [10] (Prop. 1), for a conventional autoregressive generation process $AR(1)$ of l steps with maximum eigenvalue $\lambda \geq 1$ and parameter discrepancy δ, the divergence between the true local density and the estimated density grows at least nearly with sequence length:

$$D_{KL}\left(p(X_{\cdot,l} \mid X_{\cdot,0}) \| p(\hat{X}_{\cdot,l} \mid X_{\cdot,0})\right) = \Omega(\delta l) \tag{5}$$

Our introduction of static conditions can be considered altering p based on the conditions, which does not affect the conclusion.

In i-th level, let $X^{[i]}_{\cdot,j} \sim X_{\cdot,(j-1)L^{[i+1]}+1:jL^{[i+1]}}$ be the j-th (global) step in this level, its parent is then $X^{[i-1]}_{\cdot,\lceil j/l \rceil}$. Between each hierarchical level, consider the information from previous level(s) to be passed and shared for all steps in the current level, then in i-th level

$$\begin{aligned} D^{[i]}_{KL} &= D_{KL}\left(p(X^{[i]}_{\cdot,j}, X^{[i-1]}_{\cdot,\lceil j/l \rceil}) \| p(\hat{X}^{[i]}_{\cdot,j}, \hat{X}^{[i-1]}_{\cdot,\lceil j/l \rceil})\right) \\ &= D_{KL}\left(p(X_{\cdot,\lceil j/l \rceil}) \| p(\hat{X}_{\cdot,\lceil j/l \rceil})\right) + D_{KL}\left(p(X^{[i]}_{\cdot,j} \mid X^{[i-1]}_{\cdot,\lceil j/l \rceil}) \| p(\hat{X}^{[i]}_{\cdot,j} \mid X^{[i-1]}_{\cdot,\lceil j/l \rceil})\right) \\ &= D^{[i-1]}_{KL} + \Omega(\delta(j \mod l)) \end{aligned} \tag{6}$$

Thus, by induction, we have

$$D_{KL}\left(p(X_{\cdot,L} \mid c) \| p(\hat{X}_{\cdot,L} \mid c)\right) = \Omega(\delta l K). \tag{7}$$

□

Theorem 4.2. *Let B be the batch size, L the sequence length, d the hidden dimension, and l the local patch size per level. For a hierarchical model like FracTS with $K = \log_l L$ levels, the total activation memory complexity is $\mathcal{O}(BL(l + d))$.*

Proof of Theorem 4.2. Recall the memory requirement for canonical Transformer [29] is $\mathcal{O}(BL^2 + BLd)$. The batch size at each level is $B\left\lceil \frac{L}{L^{[i]}} \right\rceil$, sequence length is $\left\lceil \frac{L^{[i]}}{L^{[i+1]}} \right\rceil$ Then the memory requirement for FracTS is

$$
\begin{aligned}
&\sum_{i=0}^{K-1} \mathcal{O}\left(B\left\lceil \frac{L}{L^{[i]}} \right\rceil \cdot \left\lceil \frac{L^{[i]}}{L^{[i+1]}} \right\rceil^2 + B\left\lceil \frac{L}{L^{[i]}} \right\rceil \cdot \left\lceil \frac{L^{[i]}}{L^{[i+1]}} \right\rceil d \right) \\
&= B\sum_{i=0}^{K-1} \mathcal{O}\left(\frac{Ll}{L^{[i+1]}} + \frac{Ld}{L^{[i+1]}} \right) \\
&= BL(l+d)\sum_{i=0}^{K-1} \mathcal{O}\left(\frac{1}{L/l^{i+1}} \right) \\
&= B(l+d)\sum_{i=0}^{K-1} \mathcal{O}\left(l^{i+1} \right) \\
&= Bl(l+d)\cdot \mathcal{O}\left(\frac{l^K - 1}{l-1} \right) \\
&= Bl(l+d)\cdot \mathcal{O}\left(\frac{L-1}{l-1} \right) \\
&= \mathcal{O}(BL(l+d))
\end{aligned} \tag{8}
$$

□

H Full Results with Standard Deviations

Tables 10 and 11 report the per-dataset results with the standard deviations over the three runs; Tables 3 and 4 in the main paper average these values over the datasets.

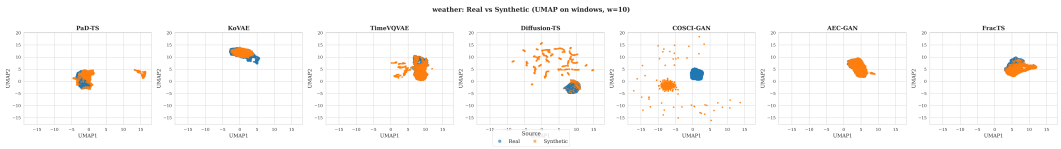
We now explain the negative Δ values in Table 4. For MD raw and CP raw, FracTS wins on Walmart (0.077 against 0.113, and 0.050 against 0.080) while TabularARGN wins on Simulated (0.195 against 0.266, and 0.057 against 0.097). The errors of FracTS on Simulated are about 3.5 and 1.9 times its errors on Walmart on these two columns, so Simulated weighs more in the dataset average, and the average turns negative even though FracTS wins on Walmart. For agg.+cond., FracTS wins on Simulated but only within the run-to-run standard deviation (0.315$\pm$0.007 against 0.319$\pm$0.008), so the two methods tie there; on Walmart, TabularARGN wins clearly (0.370 against 0.503), and this is the only cell where FracTS is not in the top two. The Walmart gap on MD raw (0.077$\pm$0.034 against 0.113$\pm$0.027) is also within the standard deviation, so the two methods tie on that cell as well. Each negative value therefore comes from a single dataset and column against one baseline, while FracTS is best or tied-best on all temporal and predictive columns on both datasets.

I Ablation Study

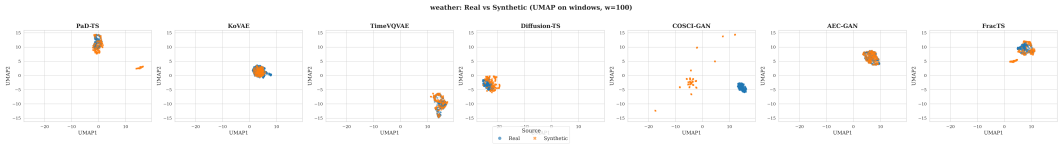
Table 12 shows the ablation experiment results. In particular, column "Data proc." shows the importance of difference data processing steps to the result. "Var. len. by" shows the difference of ways handling variable length input, by naïve padding or passing attention mask as input to the transformer. "Len. by" shows the source of length predicted. Results by varying these values show the effectiveness of the core design components of FracTS introduced in Section 4.

Column "$l^{[i]}$" refers to the length of the transformer at each level. In particular, $l^{[i]} = L$ corresponds to a single-level, non-hierarchical setting (apart from the final layer for raw values). As shown in Table 12, this setting substantially degrades performance on Weather and becomes infeasible on the much longer Walmart sequences, highlighting the importance of the hierarchical decomposition. These results demonstrate the effect of core hyperparameters.

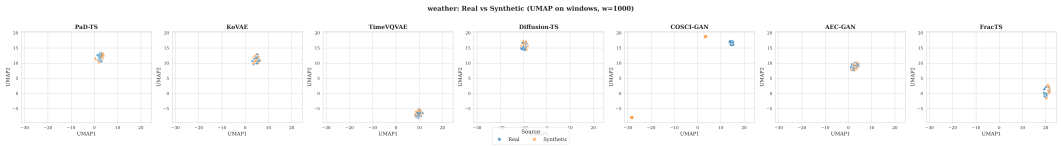
Column "Cond. model q_ϕ" refers to the tabular model used for condition generation, and "Backbone" refers to the transformer backbone. The results highlight both the impact of strong tabular backbones for condition generation and the influence of transformer size on model performance.



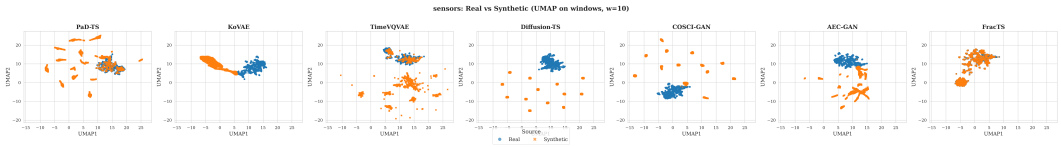
(a) Weather, $W = 10$



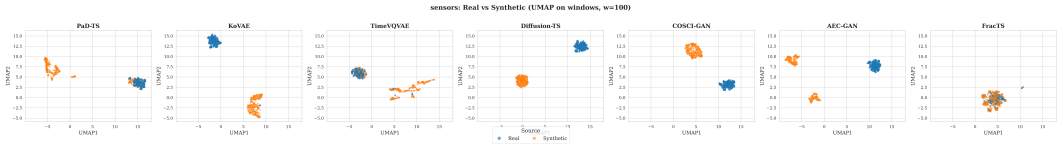
(b) Weather, $W = 100$



(c) Weather, $W = 1000$



(d) Sensors, $W = 10$

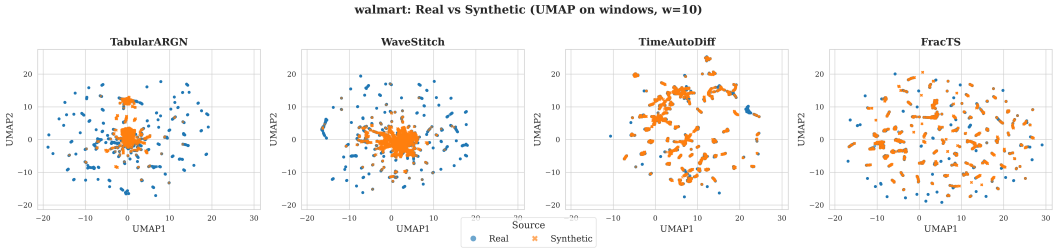


(e) Sensors, $W = 100$

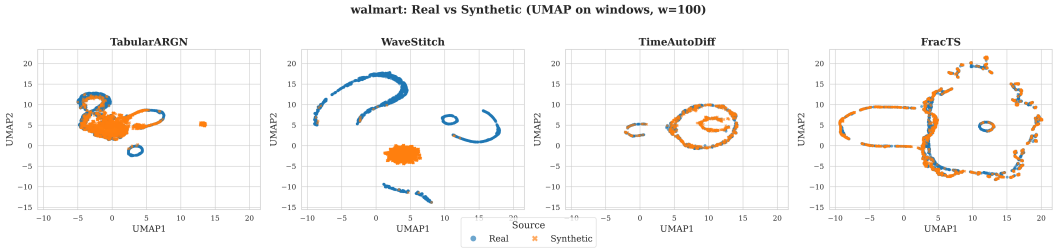
Figure 11: Windowed t-SNE visualizations of real and generated samples for the unconditional datasets at different window sizes. ($W = 1000$ is omitted for Sensors because every Sensors series is shorter than 1000 time steps, so no such window exists.)

Table 10: Unconditional generation performance with standard deviations over three runs (lower is better; best in **bold**).

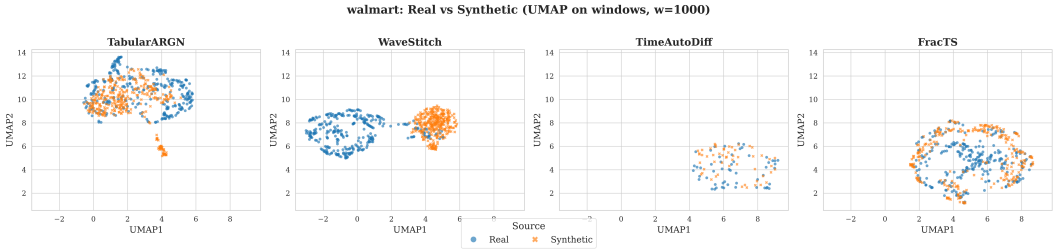
		MD						CP		PM		
	Model	raw	mean	quantile	skewness	autocorr.	length	raw	agg.	W=10	W=100	W=1000
Weather	AEC-GAN	0.503±0.000	0.845±0.000	0.887±0.000	0.613±0.000	0.697±0.000	–	0.393±0.000	0.465±0.000	0.076±0.000	0.077±0.000	0.075±0.000
	KoVAE	0.692±0.238	0.861±0.113	0.877±0.099	0.781±0.170	0.775±0.082	1.000±0.000	0.169±0.010	0.438±0.001	0.286±0.291	0.284±0.279	0.280±0.275
	DiffusionTS	0.805±0.002	1.000±0.000	1.000±0.000	0.714±0.015	0.821±0.045	1.000±0.000	0.361±0.113	0.523±0.045	0.062±0.010	0.082±0.014	0.069±0.020
	TimeVQVAE	0.466±0.049	0.720±0.069	0.705±0.068	0.563±0.022	0.840±0.012	1.000±0.000	0.156±0.013	0.511±0.028	0.047±0.008	0.049±0.007	0.045±0.006
	COSCI-GAN	0.559±0.001	0.879±0.009	0.894±0.021	1.000±0.000	1.000±0.000	1.000±0.000	0.745±0.001	0.677±0.140	0.125±0.004	0.125±0.005	0.135±0.005
	PaD-TS	0.388±0.020	0.581±0.012	0.679±0.036	0.650±0.048	0.788±0.027	1.000±0.000	0.354±0.039	0.542±0.054	0.023±0.002	0.023±0.002	0.021±0.002
	FracTS (Ours)	**0.279**±0.083	**0.573**±0.123	**0.545**±0.136	**0.542**±0.088	**0.621**±0.072	**0.440**±0.244	**0.113**±0.003	**0.358**±0.022	**0.011**±0.002	**0.011**±0.002	**0.015**±0.008
Stocks	AEC-GAN	0.453±0.001	0.537±0.003	0.529±0.002	0.726±0.002	0.618±0.002	–	0.143±0.000	**0.129**±0.002	12 157±18	12 513±19	16 466±24
	KoVAE	0.553±0.000	0.593±0.000	0.489±0.000	0.990±0.000	0.863±0.000	0.885±0.000	0.377±0.000	0.464±0.000	22 384±0	23 039±0	30 312±0
	DiffusionTS	0.703±0.063	0.912±0.026	0.921±0.019	0.587±0.016	0.865±0.040	0.885±0.000	0.777±0.081	0.579±0.025	7 573±0	7 795±0	10 257±0
	TimeVQVAE	0.646±0.001	0.656±0.009	0.653±0.015	0.424±0.041	0.796±0.040	0.885±0.000	0.099±0.010	0.375±0.007	20 467±994	21 067±1 023	27 721±1 346
	COSCI-GAN	0.494±0.082	0.641±0.197	0.633±0.205	0.761±0.121	0.925±0.035	0.885±0.000	0.268±0.086	0.311±0.033	7 175±1 598	7 385±1 645	9 716±2 164
	PaD-TS	0.292±0.020	0.327±0.005	0.328±0.015	0.735±0.017	0.785±0.031	0.885±0.000	0.289±0.012	0.397±0.024	11 524±615	11 862±633	15 607±833
	FracTS (Ours)	**0.225**±0.017	**0.232**±0.026	**0.230**±0.034	**0.139**±0.028	**0.335**±0.039	**0.675**±0.026	**0.023**±0.004	0.148±0.003	**0.001**±0.001	**0.001**±0.001	**0.005**±0.004
Sensor	AEC-GAN	0.615±0.000	0.667±0.000	0.676±0.001	0.511±0.000	0.513±0.001	–	0.328±0.000	0.351±0.000	0.006±0.000	0.037±0.000	–
	KoVAE	0.780±0.112	0.671±0.257	0.653±0.257	0.581±0.065	0.465±0.165	0.984±0.000	0.236±0.047	0.566±0.014	0.159±0.213	0.247±0.132	–
	DiffusionTS	0.577±0.000	0.885±0.000	0.823±0.000	0.756±0.000	0.639±0.000	0.984±0.000	0.260±0.000	0.523±0.000	0.162±0.000	0.178±0.000	–
	TimeVQVAE	0.790±0.015	0.801±0.016	0.742±0.010	0.501±0.031	0.559±0.032	0.984±0.000	0.293±0.018	0.443±0.004	0.045±0.002	0.055±0.004	–
	COSCI-GAN	0.619±0.005	0.877±0.014	0.718±0.024	0.757±0.014	0.649±0.004	0.984±0.000	0.372±0.007	0.430±0.001	0.055±0.004	0.131±0.008	–
	PaD-TS	0.556±0.006	0.576±0.004	0.532±0.014	0.470±0.010	0.569±0.010	0.984±0.000	0.271±0.008	0.470±0.008	**0.004**±0.001	**0.004**±0.001	–
	FracTS (Ours)	**0.437**±0.007	**0.228**±0.019	**0.304**±0.021	**0.130**±0.017	**0.113**±0.007	**0.212**±0.048	**0.026**±0.003	**0.161**±0.003	0.007±0.001	0.008±0.000	–



(a) Walmart, $W = 10$



(b) Walmart, $W = 100$



(c) Walmart, $W = 1000$

Figure 12: Windowed t-SNE visualizations of real and generated samples for the conditional Walmart dataset at different window sizes.

Table 11: Conditional generation performance with standard deviations over three runs (lower is better; best in **bold**).

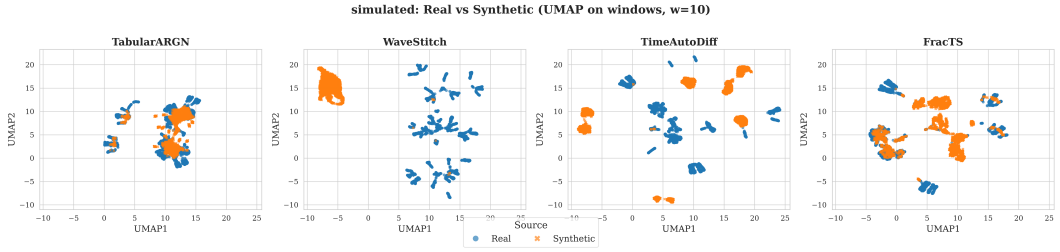
		MD						CP		PM			GT	
	Model	raw	mean	quantile	skewness	autocorr.	length	raw	agg.+cond.	W=10	W=100	W=1000	MSE	MAE
Walmart	TabularARGN	0.113$_{\pm 0.027}$	0.812$_{\pm 0.116}$	0.775$_{\pm 0.109}$	0.741$_{\pm 0.098}$	1.000$_{\pm 0.000}$	0.556$_{\pm 0.000}$	0.080$_{\pm 0.002}$	**0.370**$_{\pm 0.047}$	**0.007**$_{\pm 0.001}$	0.069$_{\pm 0.001}$	**0.005**$_{\pm 0.001}$	–	–
	WaveStitch	0.653$_{\pm 0.015}$	1.000$_{\pm 0.000}$	0.944$_{\pm 0.096}$	0.932$_{\pm 0.069}$	1.000$_{\pm 0.000}$	0.889$_{\pm 0.000}$	0.163$_{\pm 0.021}$	0.619$_{\pm 0.045}$	0.081$_{\pm 0.008}$	0.018$_{\pm 0.008}$	0.076$_{\pm 0.006}$	–	–
	TimeAutoDiff	0.247$_{\pm 0.182}$	0.778$_{\pm 0.185}$	0.802$_{\pm 0.113}$	0.667$_{\pm 0.161}$	0.852$_{\pm 0.148}$	0.889$_{\pm 0.000}$	0.237$_{\pm 0.005}$	0.435$_{\pm 0.019}$	0.142$_{\pm 0.083}$	0.181$_{\pm 0.133}$	0.153$_{\pm 0.096}$	–	–
	FracTS (Ours)	**0.077**$_{\pm 0.034}$	**0.583**$_{\pm 0.167}$	**0.633**$_{\pm 0.130}$	**0.549**$_{\pm 0.043}$	**0.781**$_{\pm 0.088}$	**0.278**$_{\pm 0.073}$	**0.050**$_{\pm 0.006}$	0.503$_{\pm 0.018}$	**0.007**$_{\pm 0.002}$	**0.003**$_{\pm 0.003}$	**0.005**$_{\pm 0.002}$	–	–
Simulated	TabularARGN	**0.195**$_{\pm 0.004}$	0.650$_{\pm 0.032}$	**0.644**$_{\pm 0.030}$	0.756$_{\pm 0.021}$	0.988$_{\pm 0.003}$	**0.000**$_{\pm 0.000}$	**0.057**$_{\pm 0.002}$	0.319$_{\pm 0.008}$	0.104$_{\pm 0.017}$	0.099$_{\pm 0.016}$	0.095$_{\pm 0.015}$	1.759$_{\pm 0.080}$	0.952$_{\pm 0.034}$
	WaveStitch	0.358$_{\pm 0.016}$	0.832$_{\pm 0.045}$	0.837$_{\pm 0.041}$	0.752$_{\pm 0.030}$	0.993$_{\pm 0.000}$	**0.000**$_{\pm 0.000}$	0.129$_{\pm 0.007}$	0.381$_{\pm 0.009}$	0.037$_{\pm 0.002}$	0.042$_{\pm 0.002}$	0.038$_{\pm 0.004}$	1.233$_{\pm 0.109}$	0.884$_{\pm 0.040}$
	TimeAutoDiff	0.458$_{\pm 0.030}$	0.760$_{\pm 0.044}$	0.762$_{\pm 0.037}$	0.927$_{\pm 0.035}$	0.944$_{\pm 0.004}$	**0.000**$_{\pm 0.000}$	0.219$_{\pm 0.014}$	0.407$_{\pm 0.016}$	0.669$_{\pm 0.092}$	0.659$_{\pm 0.088}$	0.630$_{\pm 0.081}$	144.260$_{\pm 103.107}$	5.103$_{\pm 2.088}$
	FracTS (Ours)	0.266$_{\pm 0.048}$	**0.639**$_{\pm 0.016}$	0.658$_{\pm 0.029}$	**0.661**$_{\pm 0.017}$	**0.887**$_{\pm 0.027}$	**0.000**$_{\pm 0.000}$	0.097$_{\pm 0.006}$	**0.315**$_{\pm 0.007}$	**0.016**$_{\pm 0.015}$	**0.012**$_{\pm 0.011}$	**0.014**$_{\pm 0.011}$	**0.983**$_{\pm 0.109}$	**0.545**$_{\pm 0.051}$

For K-Means discretization of numerical features, we use $k = 20$ bins by default. We additionally evaluate $k = 2$ and $k = 200$. A very small number of bins provides limited modal guidance, whereas a very large number of bins reduces the ordinal relationship between adjacent numerical values and can degrade performance.

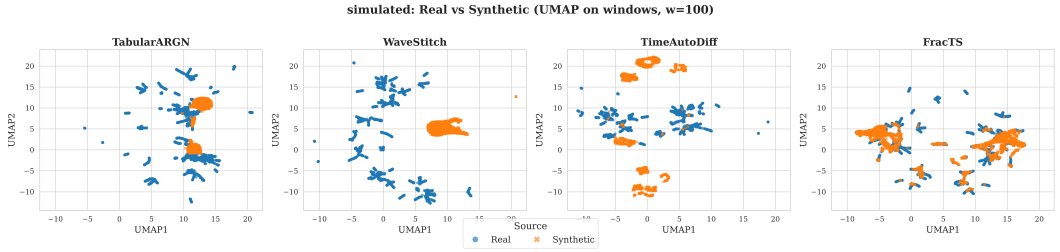
The effect of the Perceiver-based condition compression module is evaluated separately in Table 13.

For $l^{[i]} = L$, the model reduces to a single-level, non-hierarchical transformer. On Weather, this setting degrades substantially at $W = 10$ and does not generate sufficiently long sequences for $W = 100$ and $W = 1000$. On Walmart, training results in out-of-memory (OOM) errors. These results show that flattening the hierarchy becomes increasingly problematic as sequence length grows.

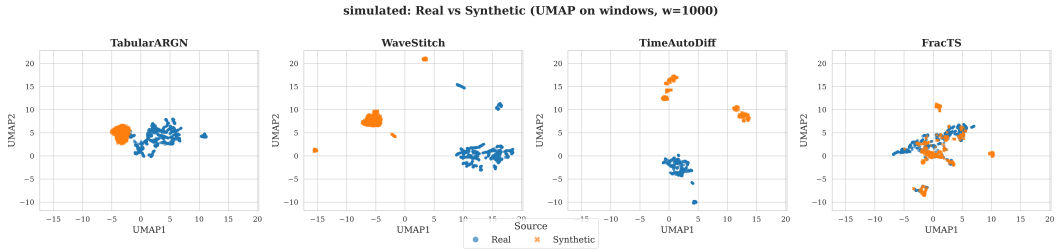
Condition compression. We further evaluate the Perceiver-based condition compression module on the Weather dataset. The module compresses the 53-dimensional condition representation to



(a) Simulated, $W = 10$



(b) Simulated, $W = 100$



(c) Simulated, $W = 1000$

Figure 13: Windowed t-SNE visualizations of real and generated samples for the conditional Simulated dataset at different window sizes.

Table 12: Ablation of FracTS components, reporting PM (predictive ML utility) MAE at prediction windows W on Weather and Walmart (lower is better). – denotes "same as the default setting" (top row, highlighted).

							Weather ↓			Walmart ↓		
Data proc.	**Var.-length handling**	**Length source**	**Global cond.**	$l^{[i]}$	**Backbone**	**Cond. model** q_ϕ	W=10	W=100	W=1000	W=10	W=100	W=1000
FracTS (default)	att. mask	predictor	✓	20	GPT-2	CTGAN	0.010	0.010	0.010	0.009	0.007	0.002
w/o differences	–	–	–	–	–	–	0.021	0.020	0.022	0.027	0.028	0.009
w/o bins	–	–	–	–	–	–	0.104	0.109	0.118	0.003	0.024	0.017
$k = 2$ bins	–	–	–	–	–	–	0.018	0.024	0.034	0.003	0.006	0.008
$k = 200$ bins	–	–	–	–	–	–	0.034	0.035	0.034	0.009	0.007	0.005
w/o aggregation	–	–	–	–	–	–	0.011	0.011	0.014	0.021	0.016	0.011
w/o all above	–	–	–	–	–	–	0.027	0.028	0.031	0.033	0.040	0.041
–	padding	–	–	–	–	–	0.010	0.010	0.010	0.007	0.011	0.016
–	–	truncation	–	–	–	–	0.011	0.012	0.014	0.007	0.009	0.036
–	–	–	✗	–	–	–	0.020	0.018	0.020	0.004	0.001	0.006
–	–	–	–	10	–	–	0.020	0.019	0.020	0.008	0.003	0.026
–	–	–	–	100	–	–	0.014	0.015	0.156	0.078	0.083	0.086
–	–	–	–	L	–	–	0.200	*no data*	*no data*	*OOM*	*OOM*	*OOM*
–	–	–	–	–	Llama	–	0.019	0.020	0.019	0.005	0.003	0.002
–	–	–	–	–	–	REaLTabFormer	0.010	0.011	0.013	0.007	0.000	0.006

16 dimensions. As shown in Table 13, this reduction preserves predictive ML utility with almost no degradation across the three prediction windows. We omit this ablation on Walmart because its condition representation is already low-dimensional, so the Perceiver module provides little effective compression in this setting.

J Limitations

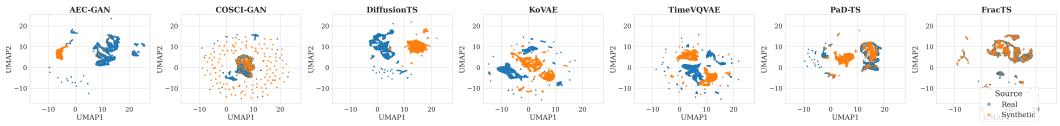
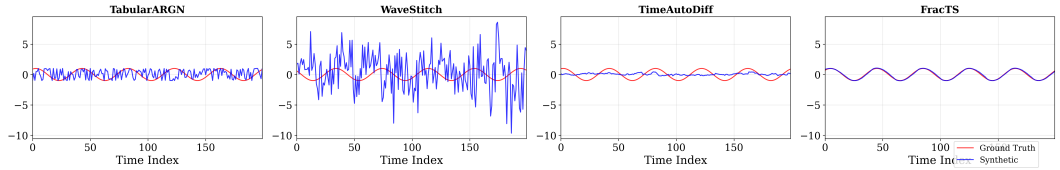
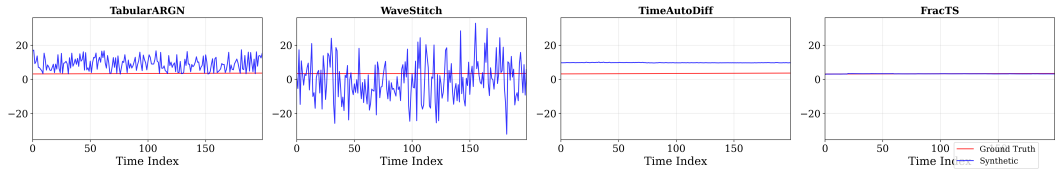


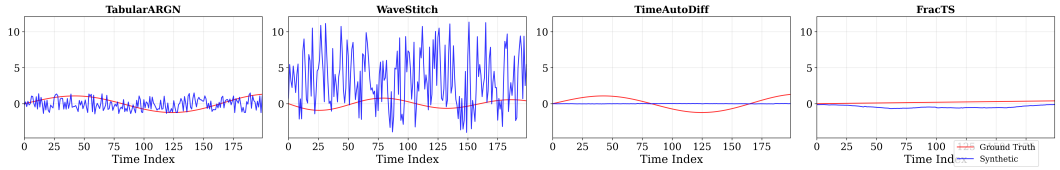
Figure 14: t-SNE of real compared to generated time series data of the Sensors dataset.



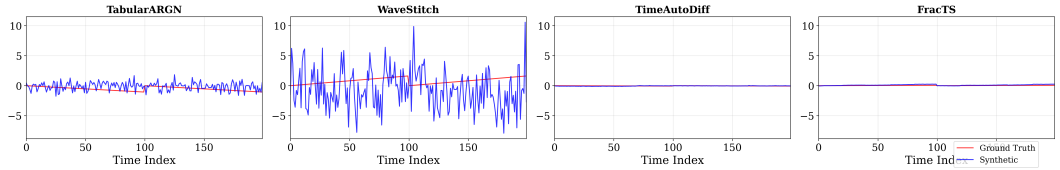
(a) col2: trigonometric



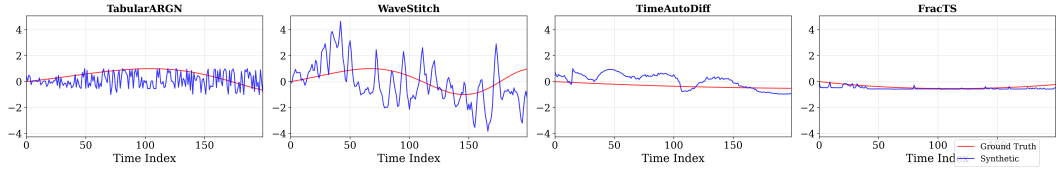
(b) col5: linear



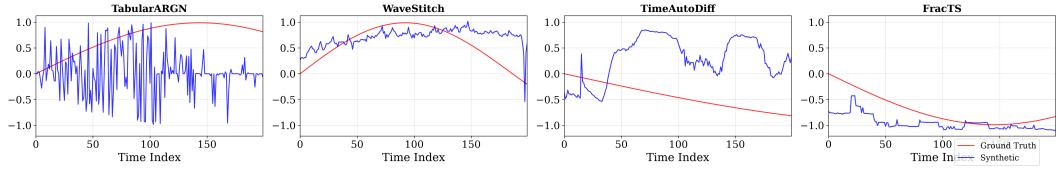
(c) col14: amplitude modulated



(d) col9: periodic ramp



(e) col13: chirp



(f) col15: Gabor-like

Figure 15: Examples of generated versus ground-truth simulated data for six representative generating formulas (Table 5). Each panel overlays FracTS samples on the analytic ground truth.

Computational overhead. FracTS uses multiple transformer models across its hierarchical levels and therefore incurs additional computational cost. Training runtime, inference latency, and GPU memory usage are reported in Appendix C. While FracTS trades some computational efficiency for its ability to model long, variable-length time series, its computational requirements remain practical under the settings evaluated in this work.

Table 13: Ablation of the Perceiver-based condition compression module on Weather. Values report PM (predictive ML utility) MAE; lower is better.

Perceiver compression	W=10	W=100	W=1000
Yes (default)	0.010	0.010	0.010
No	0.010	0.011	0.011

Sensitivity to hierarchy depth. The hierarchy depth is determined by the local patch size $l^{[i]}$ relative to the sequence length. Using a fixed appropriate patch size, such as the default $l^{[i]} = 20$, automatically adapts the number of hierarchy levels to the scale of the dataset. As shown in Appendix I, performance degrades mainly under more extreme choices, such as excessively large patch sizes or the completely flattened single-level setting $l^{[i]} = L$.

Failure modes and scope. FracTS is designed primarily for long and variable-length time series. Applying the model to short, fixed-length sequences may introduce unnecessary architectural complexity compared with conventional generative models. Conversely, for extremely long sequences, GPU memory limitations can arise from the growing effective batch sizes across hierarchical levels; this can be addressed in practical implementations through level-wise gradient accumulation or updates.

FracTS also assumes clean inputs represented using standard tabular data types, including numerical, categorical, and datetime features. It does not directly handle arbitrary textual metadata, identifier fields, or raw missing values. For missing values, we assume that standard preprocessing, such as simple imputation or explicit missing-value indicators, is applied before training.

Irregularly sampled time series. FracTS can represent irregularly sampled sequences through its support for variable sequence lengths and explicit datetime features. For example, the Stock dataset contains irregularities due to market closures, weekends, and trading holidays. By using precise timestamps as input features, together with temporal step differences during preprocessing, FracTS does not require uniformly spaced observations.

Privacy. We evaluate privacy using the Distance-to-Closest-Record (DCR) metric across different window sizes. Higher values indicate lower privacy leakage risk, with 0.5 representing an ideal balance and 1 representing ideal privacy preservation. The results are reported in Table 14.

Table 14: Distance-to-Closest-Record (DCR) privacy evaluation across different window sizes. Higher values indicate lower privacy leakage risk.

Dataset	**Model**	W=10	W=100	W=1000
Weather	FracTS	$0.515_{\pm 0.010}$	$0.273_{\pm 0.058}$	$0.291_{\pm 0.106}$
Weather	PaD-TS	$0.592_{\pm 0.035}$	$0.364_{\pm 0.036}$	$0.125_{\pm 0.046}$
Walmart	FracTS	$0.649_{\pm 0.009}$	$0.314_{\pm 0.022}$	$0.200_{\pm 0.025}$
Walmart	TabularARGN	$0.942_{\pm 0.034}$	$0.893_{\pm 0.029}$	$0.849_{\pm 0.081}$

There is an inherent trade-off between fidelity and privacy. FracTS generally obtains lower DCR values than the compared baselines, although it performs better in some long-horizon settings, such as compared with PaD-TS at $W = 1000$. Privacy risk is limited for shorter windows ($W = 10$), while longer windows show increased leakage risk. This is partly due to the smaller number of samples available for DCR evaluation at $W = 100$ and $W = 1000$, which makes high-fidelity synthetic samples naturally closer to real samples.

Because FracTS uses a standard transformer backbone, differential privacy mechanisms such as DP-SGD can be incorporated when stricter privacy guarantees are required.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly outline the introduction of FracTS, its hierarchical and autoregressive nature, and its performance across five datasets compared to seven baselines.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a dedicated limitations section (Appendix J) discussing computational overhead, sensitivity to hierarchy depth, failure modes and scope, handling of irregularly sampled and missing-value data, and privacy considerations. It also discusses performance limitations on highly complex composite and oscillatory signals.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper provides two theoretical results in Section 4: an analysis of error propagation in the hierarchical autoregressive architecture and an analysis of its activation memory complexity. The assumptions are stated in the theorem statements, and proofs are provided in Appendix G.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides implementation details, including the architecture (GPT2 small), optimization parameters (AdamW, learning rate, weight decay), and specific hyperparameters for different datasets.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly available at https://github.com/zhao-zilong/fracts.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 and Appendix C.2 detail the datasets, data processing pipeline, model architecture, and the specific evaluation setup used.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Performance tables, such as Table 10, include mean values and standard deviations for the reported metrics.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies that all experiments were run on NVIDIA H100 GPUs using PyTorch with mixed precision training.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper follows standard research practices and includes a discussion on the ethical use of data collection .

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The paper notes that synthetic data generation is important for privacy-preserving sharing, but also mentions the necessity of ethical data collection practices .

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper focuses on a general architectural framework for time series and does not release high-risk pretrained language or image models that require specific safety filters.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the original sources for all five datasets used (e.g., Sensor, Weather, Stock, Walmart).

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces a new simulated dataset with 15 columns of diverse temporal patterns, which is documented in Section 5.1 and Appendix B.3.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The research does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

Justification: The research does not involve human subjects, so IRB approval is not applicable.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.