WaveStitch: Flexible and Fast Conditional Time Series Generation With Diffusion Models

ADITYA SHANKAR, Delft University of Technology, The Netherlands
LYDIA CHEN, Delft University of Technology, Université de Neuchâtel, Switzerland
ARIE VAN DEURSEN, Delft University of Technology, The Netherlands
RIHAN HAI, Delft University of Technology, The Netherlands

Generating temporal data under conditions is crucial for forecasting, imputation, and generative tasks. Such data often has metadata and partially observed signals that jointly influence the generated values. However, existing methods face three key limitations: (1) they condition on either the metadata or observed values, but rarely both together; (2) they adopt either training-time approaches that fail to generalise to unseen scenarios, or inference-time approaches that ignore metadata; and (3) they suffer from trade-offs between generation speed and temporal coherence across time windows, choosing either slow but coherent autoregressive methods or fast but incoherent parallel ones. We propose WaveStitch, a novel diffusion-based method to overcome these hurdles through: (1) dual-sourced conditioning on both metadata and partially observed signals; (2) a hybrid training-inference architecture, incorporating metadata during training and observations at inference via gradient-based guidance; and (3) a novel pipeline-style paradigm that generates time windows in parallel while preserving coherence through an inference-time conditional loss and a stitching mechanism. Across diverse datasets, WaveStitch demonstrates adaptability to arbitrary patterns of observed signals, achieving 1.81x lower mean-squared-error compared to the state-of-the-art, and generates data up to 166.48x faster than autoregressive methods while maintaining coherence. Our code is available at: https://github.com/adis98/WaveStitch.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Time Series Generation, Conditional Generation, Diffusion Models

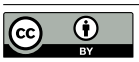
ACM Reference Format:
Aditya Shankar, Lydia Chen, Arie van Deursen, and Rihan Hai. 2025. WaveStitch: Flexible and Fast Conditional Time Series Generation With Diffusion Models. *Proc. ACM Manag. Data* 3, 6 (SIGMOD), Article 377 (December 2025), 25 pages. https://doi.org/10.1145/3769842

1 Introduction

Time series data often suffer from missing or noisy values, or are subject to access restrictions due to privacy concerns, which are important challenges in data management. Although synthetic data generation shows promise in mitigating these issues [26, 32, 39], most generative models are *unconditional*, i.e., they cannot control the characteristics of the generated data.

Conditional synthesis enables injecting user-defined constraints into the generation process, such as provided metadata or partially observed signals [2, 22, 29]. For example, we can use it for data management tasks such as *imputation* [7], *forecasting* [12, 33], and data cleaning [9] by providing clean parts of the data as conditional observations for the model to regenerate or correct the rest. For such tasks, forecasting is often treated as a special case of imputation, where missing values occur

Authors' Contact Information: Aditya Shankar, Delft University of Technology, Delft, The Netherlands, a.shankar@tudelft.nl; Lydia Chen, Delft University of Technology, Université de Neuchâtel, Neuchâtel, Switzerland, lydiaychen@ieee.org; Arie van Deursen, Delft University of Technology, Delft, The Netherlands, arie.vandeursen@tudelft.nl; Rihan Hai, Delft University of Technology, Delft, The Netherlands, r.hai@tudelft.nl.



ACM 2836-6573/2025/12-ART377
https://doi.org/10.1145/3769842

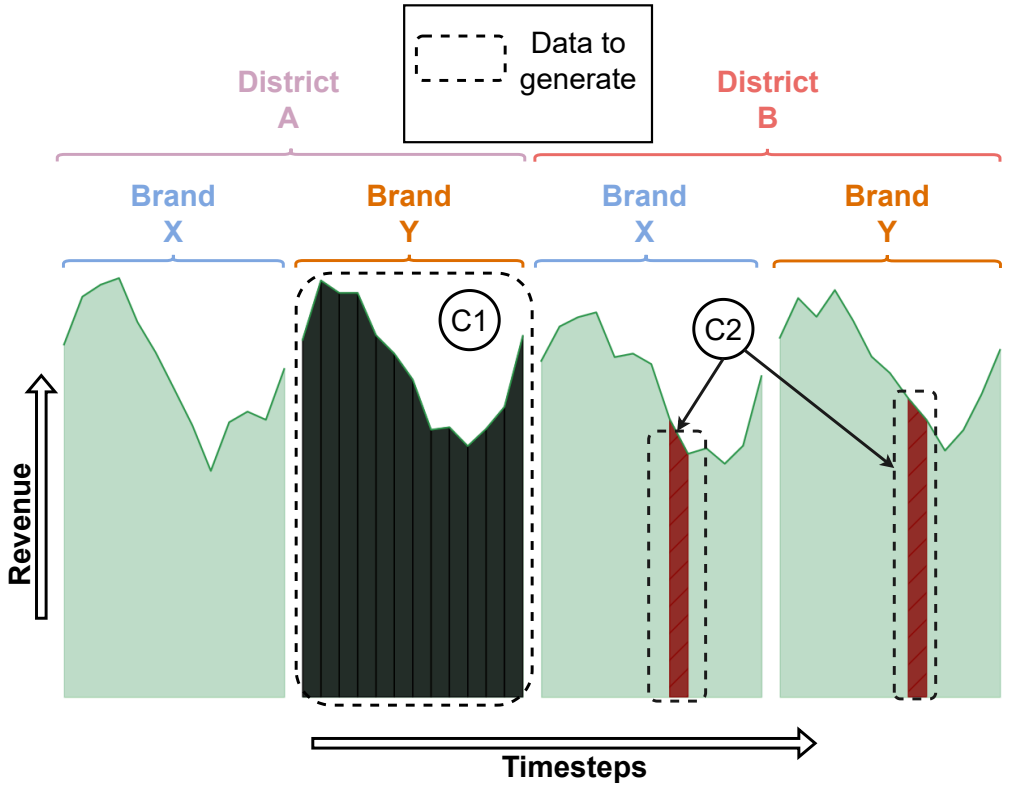
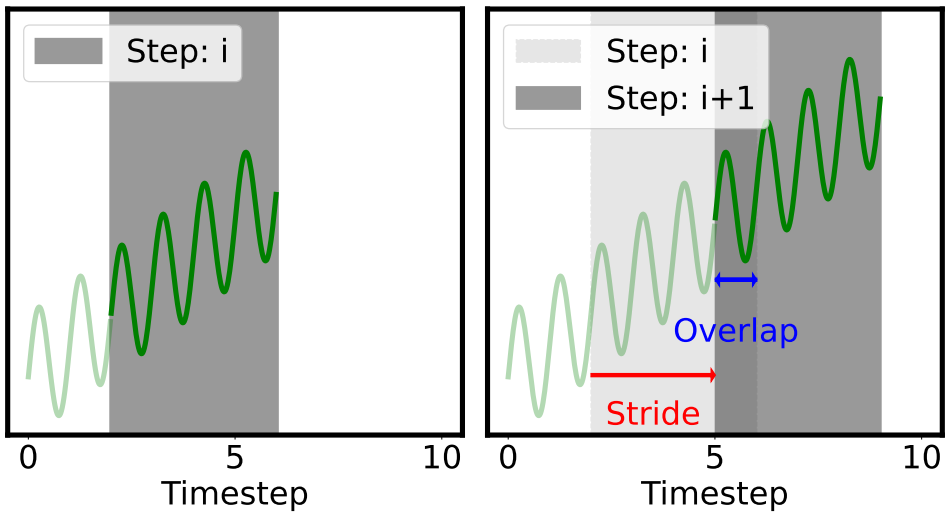


Fig. 1. Time series (signal) of partially observed yearly revenues (in green), grouped by metadata District, Brand and Month. We want to generate missing data under varying conditions, like C1: (District *A* + Brand *Y*), or C2: (District *B* + Month *August*).

only at the tail-end of the time series [2, 22]. Going beyond imputation and forecasting, we can also have *generative* tasks that are more loosely defined and are used for tasks like data augmentation or simulating new scenarios [26, 29, 38]. Unlike forecasting and imputation, generative tasks target producing samples that are consistent and plausible with high-level constraints such as metadata, rather than having an exact match with partially observed signals.

Consider Figure 1 showing a time series signal, *Revenue*, accompanied by metadata *District*, *Brand*, and *Month*. Suppose the data is stored by concatenating records from all district–brand combinations into a single denormalised table, with the row indices defining the timesteps. We can impose conditions by specifying the metadata. For example, a condition such as District *A* + Brand *Y* means "*Generate revenue values for all timesteps with district A and brand Y.*" The model then generates values for all the timesteps matching the condition. However, this task is not straightforward due to the following challenges.

Challenge 1. Existing methods fail to condition on both the metadata (e.g., brand, district) *and* the observed signals in conjunction. Figure 1 shows that data can exhibit varying missingness patterns, such as contiguous blocks (C1), or fine-grained gaps (C2). Existing models condition only on the observed values and ignore the metadata [2, 22], or vice versa [29]. Using only metadata can capture global trends, but would fail to align with the observed signals. Conversely, conditioning only on the observed signals may locally fill in gaps but overlook the broader context given by the



(a) Signal generation Step i (b) Signal generation Step $i+1$

Fig. 2. Autoregressive synthesis with sliding windows.

metadata. The design challenge lies in using both sources to account for their differing influence in shaping global and local temporal patterns.

Challenge 2. The second challenge is deciding *when* to incorporate conditional information, during *training* or *inference*. The choice is influenced by the differing characteristics of metadata and observed values. Metadata is static, and a model can learn its conditional influence during training. In contrast, signals may be observed at arbitrary timesteps (see C1 and C2 in Figure 1), requiring flexibility at inference. However, existing works adopt a *black-or-white* stance, conditioning exclusively during training [2, 29, 34, 41, 42] or during inference [22, 27].

Challenge 3. Time series signals can be generated either *autoregressively* or in *parallel*. Autoregression incurs high runtime costs, generating signals sequentially using a sliding window to propagate temporal dependencies across consecutive segments [34] (see Figure 2). In contrast, parallel methods generate multiple windows simultaneously [22, 29], but ignore maintaining temporal coherence between them. The challenge lies in maintaining both efficiency *and* coherence, a balance that current methods struggle to achieve.

We propose WaveStitch to tackle these challenges using *denoising diffusion probabilistic models* (DDPMs) [15]. These models have demonstrated better performance over alternatives such as generative adversarial networks (GANs) [14] and Variational Autoencoders (VAEs) [21] across various modalities [11, 22, 23, 29], resulting in their adoption for data management tasks such as tabular and time series generation [7, 26, 39].

Our key novelty is a unified framework to handle all three challenges. WaveStitch *dual-sources* its conditioning to use both metadata *and* observed signals. This combination requires *hybridising* training and inference. We first train a base generative model that conditions only on metadata. We inject the observed signals directly *at inference*, guiding the sampling trajectory using gradients from a *conditional loss*. This loss incorporates a novel *stitching* mechanism to enforce coherence across overlapping time windows, thereby restricting the generation process to remain close to the realistic space. The approach is similar to *gradient inversion* [19], where gradients update the *sample* to match a given objective while keeping the model frozen. Moreover, the conditional

loss is easily parallelised, enabling fast synthesis. This innovation allows us to break away from prior approaches and introduce **a new paradigm for time series generation** through *pipeline parallelism*. Multiple time windows are generated simultaneously with dependencies propagating across overlaps, maintaining high efficiency and temporal coherence.

To reiterate, we summarise our key contributions as follows:

1. Dual-sourced Conditioning: WaveStitch combines two complementary sources of information: metadata for capturing global context (Sec. 3.3), and observed signals (Sec. 3.4) for providing fine-grained local cues to refine generation with high precision.

2. Hybrid Architecture: WaveStitch combines the strengths of training and inference-time strategies. It learns the conditional influence of the (static) metadata during training (Sec. 3.3), and dynamically refines samples at inference using any observed signal values via gradient-based guidance (Sec. 3.4). This approach enables the handling of arbitrary patterns of observed signals while preserving the global structure dictated by the metadata.

3. Coherent Pipelined-Parallel Generation: To improve efficiency, we generate consecutive time windows in a *pipelined-parallel* fashion, incorporating a novel *stitching mechanism* to reconcile overlaps at inference. Dependency information propagates across time windows while they are generated in parallel. This process resembles a pipeline, ensuring smooth and coherent transitions without compromising speed (Secs. 3.4, 4).

4. Evaluations on Diverse Tasks: We evaluate WaveStitch on point-wise and synthetic data quality metrics, demonstrating its flexibility across tasks and achieving up to **1.81x** lower Mean-Squared-Error (MSE) compared to the state-of-the-art (SOTA) baselines. Results show that WaveStitch's parallelism enables significantly faster generation than autoregressive synthesis (**166.48x**) while maintaining generation quality. Additional ablations assess the impact of the conditioning strategy, stitch loss formulations, and random imputation tasks with varying degrees of missingness (Sec. 5).

2 Background and Problem Definition

We cover the foundations of diffusion models for time series, their conditional extensions, and formally define the problem while briefly highlighting the research gaps. Table 1 summarises all the key symbols and notations used henceforth.

2.1 Diffusion Models for Time Series

Denoising Diffusion Probabilistic Models (DDPMs) are a class of generative models used extensively for time series [2, 22, 29, 48]. As shown in Figure 3, DDPMs operate by gradually corrupting data with a *forward* (noising) process, and then learning to reverse this through a backward *denoising* process [15]. We forward noise a time window of w timesteps starting from timestep i, as follows[1]:

$$\mathbf{x}_t^{(i:i+w-1)} = \sqrt{\alpha_t}\mathbf{x}_{t-1}^{(i:i+w-1)} + \sqrt{1-\alpha_t}\epsilon_t, \tag{1}$$

where $\mathbf{x}_t^{(i:i+w-1)}$ is the multivariate time series signal after noising $t \in \{0, 1 \dots T\}$ steps, $1-\alpha_t$ is the t-th noise variance, $(i : i+w-1)$ indicates the time slice[2], and $\epsilon_t \sim \mathcal{N}(0, I)$ is sampled white noise. With this notation, $\mathbf{x}_0^{(:)}$ and $\mathbf{x}_T^{(:)}$ represent a clean and fully noised sample, respectively. Noising up to any step without recursively applying eq. (1) is done with a *reparameterisation trick* [15]:

$$\mathbf{x}_t^{(:)} = \sqrt{\bar{\alpha}_t}\mathbf{x}_0^{(:)} + \sqrt{1-\bar{\alpha}_t}\epsilon, \tag{2}$$

[1]There are two types of "steps" defined in this study. *Diffusion* steps refer to the iterative noise addition or removal in the generative process, while *timesteps* refer to the temporal axis in the time series data.

[2]both i and $i+w-1$ are included, yielding window size w

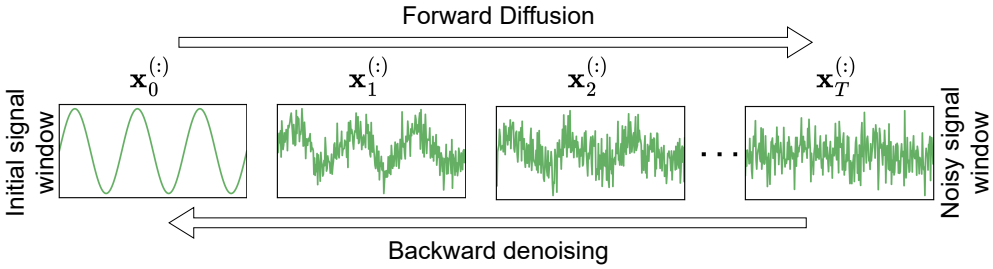


Fig. 3. Diffusion process. $\mathbf{x}_0^{(:)}$ and $\mathbf{x}_T^{(:)}$ represent the clean and fully noised time window respectively.

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ and $\epsilon \sim \mathcal{N}(0, I)$. The reverse process iteratively denoises $\mathbf{x}_t^{(:)}$ by estimating the noise at each step using a neural network $f_\theta(\hat{\mathbf{x}}_t^{(:)}, t)$. This step is given by:

$$\hat{\mathbf{x}}_{t-1}^{(:)} = \frac{1}{\sqrt{\alpha_t}} \left(\hat{\mathbf{x}}_t^{(:)} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\hat{\mathbf{x}}_t^{(:)}, t) \right) + \sigma_t z, \tag{3}$$

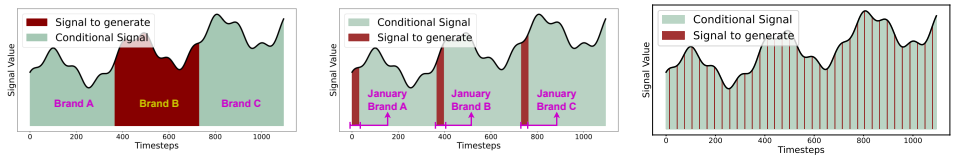
where $\hat{\mathbf{x}}_t^{(:)}$ and $\hat{\mathbf{x}}_{t-1}^{(:)}$ represent the signal estimates at step t and $t-1$ respectively, σ_t is a function of α to ensure sample diversity, and $z \sim \mathcal{N}(0, I)$ [15]. The denoiser trains by noising the signal up to a random step t using eq. (2), and then uses the MSE loss with the sampled noise for backpropagation, pushing the sample towards matching the ground truth, i.e., $\hat{x}_0^{(:)} \simeq x_0^{(:)}$ [15].

2.2 Conditional Diffusion for Time Series

Standard DDPMs cannot generate samples adhering to conditions, as synthetic samples are generated freely from random noise in an unconstrained manner. In contrast, *conditional* DDPMs constrain the denoising process using metadata or observed signals [2, 22, 29]. These conditions can be incorporated at different times, either during *training* or *inference*.

2.2.1 Training-time Conditioning. There are three main ways to condition a DDPM during training. The first way is to provide the conditions as additional inputs (e.g. metadata [29]) during training, or through binary masks to indicate the timesteps with missing and observed signals [2, 42]. The second way is *classifier-free guidance* [16, 41], which trains both a conditional and an unconditional model, and interpolates between their outputs to control the degree of conditioning. The third way is *classifier guidance* [11, 26], where an external classifier or controller guides the sampling of an unconditional denoiser [26]. The classifier functions as a feedback loop, adjusting the model's outputs by estimating the likelihood of meeting the given conditions at each denoising step.

2.2.2 Inference-Time Conditioning. These approaches control sampling directly at inference by steering or guiding the outputs of an unconditional diffusion model. *RePaint* [7, 27], adds a controlled amount of noise to the observed signals, to resemble the noise-corrupted samples seen during training. An unconditional model then denoises the entire sequence, generating both the observed and missing values jointly. *Self-Guidance* [22] conditions on arbitrary patterns of observed values by leveraging the unconditional model's own estimates. Unlike classifier guidance, which relies on a separately trained classifier for likelihood estimation, self-guidance repurposes the denoiser itself for this task. It first computes a rough signal estimate $\hat{\mathbf{x}}_0^{(:)}$ from the current outputs $\hat{\mathbf{x}}_t^{(:)}$, by inverting the forward noising process in eq. (2). The samples are then iteratively corrected using the gradient of the conditional log-likelihood of the observed signals directly at inference.



(a) Root: $(B, *, *)$ (b) Intermediate: $(*, January, *)$ (c) Bottom: $(*, *, 15)$

Fig. 4. (a) Root (coarse-grained), (b) Intermediate and (c) Bottom level (fine-grained) conditions in Example 1.

2.3 Problem Definition

We are given a time series dataset. $\mathcal{D} = \{(\mathbf{a}^{(i)}, \mathbf{x}^{(i)})\}_{i=1}^{N}$ with N timesteps, where $\mathbf{a}^{(i)} \in \mathbb{R}^{1\times L}$ denotes a sample with L metadata features and $\mathbf{x}^{(i)} \in \mathbb{R}^{1\times C}$ is a C-variate signal at the i^{th} timestep. The aim is to generate time series signals given the metadata and signals observed at arbitrary timesteps. Training inputs include complete pairs of metadata and signals, i.e., training uses signals $\mathbf{x}^{(i)}$ with available metadata $\mathbf{a}^{(i)}$. The model must generate time series signals conditioned on the metadata and the observed signals *at inference time*. Conditional metadata may be specified as $\mathbf{a}^{(k)} = (\mathrm{u}, \mathrm{v}, *, *, *)$, indicating that signals should be generated (or possibly re-generated) at all timesteps k where the first two metadata columns match u and v respectively. The asterisk $(*)$ indicates unconstrained features. Therefore, the set $\{\mathbf{a}^{(k)}\}$ is assumed to contain all valid enumerations of the unspecified metadata features that satisfy the given condition. These enumerations are then inserted into $\mathcal{D}$ at their appropriate positions in the timeline. The model then needs to generate signals $\mathbf{x}^{(k)}$ conditioned on $\mathbf{a}^{(k)}$ and any observed signals $\mathbf{x}^{(i\neq k)}$ already present in $\mathcal{D}$. We further illustrate this problem setup using Example 1 and Figure 4.

Example 1. Consider a dataset containing one year's worth of daily sales data for three brands, *A*, *B*, and *C*. Each Sales entry (the signal) corresponds to a particular *Brand*, on a given *Month*, and *Day*. The data is organised by concatenating the entries from all three brands into a table where each row is assigned a timestep index based on its position.

A condition on the root-level metadata feature, such as $(Brand = B, *, *)$, requires generating sales for the entire year for Brand *B*. As Figure 4a shows, the generated portions comprise a long *contiguous block* of timesteps, resembling *coarse-grained* synthesis. In contrast, a condition on the bottom-level metadata feature, such as $(*, *, Day = 15)$, generates sales for the 15^{th} day of each month and brand, resembling *fine-grained* tasks, as shown in Figure 4c. Similarly, tasks may also have intermediate granularity by conditioning mid-level features, like in Figure 4b .

2.4 Research Gaps

The inability to generalise to unseen conditional patterns limits training time methods. For instance, TimeWeaver [29] conditions only on the metadata and cannot leverage information from the observed signals. Similarly, SSSD [2] requires knowing the conditional mask of observations, which can lead to overfitting on the patterns seen during training. Classifier-free guidance requires both a conditional and unconditional model variant, which still requires training the conditional one with specific conditions in mind [16]. Even classifier guidance depends on pre-trained classifiers tailored to each condition type [26], making it inflexible during inference since the classifiers require training with the right conditions in advance.

Among inference-time methods, RePaint separately adds noise to the observed and missing portions, causing misalignment between the two [10, 27]. These inconsistencies are further amplified in time series, where maintaining coherence across time windows is also critical. Hence,

Table 1. Summary of key symbols.

Symbol	Description
$\mathbf{x}_t^{(:)}, \hat{\mathbf{x}}_t^{(:)}$	Ground-truth signal and model estimate at the t^{th} diffusion step
$(\alpha_t, \bar{\alpha}_t), \sigma_t$	Diffusion parameters, sampling variance
f_θ	Denoiser neural network
$\mathcal{X}, \mathcal{A}, \mathcal{M}$	Signal, metadata, and mask datasets
$\mathcal{D}$	Full dataset comprising metadata and signals
$\mathcal{X}_w, \mathcal{A}_w, \mathcal{M}_w$	Windowed signal, metadata, and mask datasets
N or M, L, C, T	Number of timesteps, metadata features, signal channels, and diffusion steps
w, s, b, η	Window size, stride, mini-batch size, guidance coefficient
$\mathbf{x}^{(i)}, \mathbf{m}^{(i)}, \mathbf{a}^{(i)}$	Signal (ground-truth), mask, and metadata at the i^{th} timestep
$\mathbf{x}_w^{(j)}, \mathbf{m}_w^{(j)}, \mathbf{a}_w^{(j)}$	j^{th} signal window (ground-truth), mask, and metadata window
$\hat{\mathbf{x}}_{w,t}^{(j)}$	j^{th} signal window (model estimate) at diffusion step t
$t_\theta(w)$	Denoising time (with f_θ) for a window of size w

RePaint's effectiveness in image inpainting [27] does not carry over to time series generation. Although ImDiffusion [7] applies RePaint for time series, it focuses on anomaly detection rather than imputation or generation and also does not consider metadata. Gradient-based inference-time conditioning using self-guidance [22] improves upon RePaint by jointly updating known and unknown regions. However, it ignores metadata and conditions only on the observed signals within a given window, limiting its use for maintaining coherence across time windows.

3 WaveStitch

This section first gives an overview of our approach and the preprocessing step for encoding categorical features (Sec. 3.2). We then detail the core components of WaveStitch (Sec. 3.3, Sec. 3.4): a *dual-sourced* conditioning strategy that *hybridises* training and inference approaches by combining a *metadata-conditioned* model during training with the observed signals at inference. We also theoretically analyse the role of the inference-time *conditional loss* (Sec. 3.5), to understand how it guides the sampling trajectory to produce coherent, observation-aligned outputs.

3.1 Overview

Real-world time series tasks often involve conditioning on partial and irregular information, such as signal observations at arbitrary timesteps. A generative model must handle static metadata and arbitrarily-positioned signal observations to operate under these settings. As illustrated in Example 1 and Figure 4, the observed (in green) and missing (in red) signals can vary in position and size depending on the metadata conditions. Since this structure is not known a priori at inference, WaveStitch hybridises training-time *and* inference-time conditioning to enable dual-source conditioning for adapting to a wide range of conditional patterns.

We first train a *metadata-conditioned* denoiser f_θ to predict noise using *only* metadata, without relying on specific patterns of observed signals. This approach enables WaveStitch to generate reasonable signal estimates even in the worst-case scenario, where we are given only the metadata and no observed signals as conditions. WaveStitch then iteratively refines the generated samples during inference via a *gradient-based* correction. At each denoising step, we adjust the generated sample using the gradient of a novel *conditional loss* computed over observed signals. This approach is similar to *gradient inversion attacks* [19], where an attacker iteratively optimises a dummy input using gradients to retrieve the original data while keeping the model frozen.

3.2 Preprocessing Categorical Metadata

Time series metadata often contains recurring categorical attributes, such as day of the week, month, and hour. To capture the periodic structure in these features, we use sine and cosine transformations inspired by positional encodings in Transformers [43] and related work in time series generation [41]. For a given feature, we map each category $k \in \{0, 1, \ldots, K-1\}$ to an angle $\theta_k = \frac{2\pi k}{K}$ on the unit circle. Each categorical value k is then encoded by the coordinates $(\sin(\theta_k), \cos(\theta_k))$, reflecting the feature's cyclical nature. This representation is more compact than *one-hot encoding* [23, 35] and represents periodic behaviour more naturally.

3.3 Training Metadata-conditioned Denoiser

We illustrate our training process in Figure 5 and show the steps in Algorithm 1. It begins with dividing the entire dataset with N timesteps into overlapping windows of size w (line 4), using a stride of one. We then forward the noise up to a randomly sampled diffusion step t and estimate the added noise, following the standard DDPM training process [15] (lines 6, 7). The noise is estimated by conditioning on the metadata for the window, $\mathbf{a}^{(i:i+w-1)}$, and uses the MSE loss to update f_θ's parameters (line 8). This way, the model learns the interactions between the metadata and the signals without relying on any observed signals. This separation sets the foundation to flexibly condition on the observations during inference (challenge 2). As we showed in Figure 4c, leveraging the observed values can further improve generation quality, particularly for fine-grained tasks where the observed signals from neighbouring timesteps can yield higher precision.

Algorithm 1 Training the Denoiser f_θ Conditioned on Metadata

1: **Input:** Encoded Metadata $\mathcal{A} = \{\mathbf{a}^{(i)}\}_{i=1}^{N}$; Multivariate time series data $\mathcal{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$; Window size w; Maximum number of denoising steps T; Noise schedule $\bar{\alpha} = \{\bar{\alpha}_t\}_{t=1}^{T}$
2: **for** each epoch **do**
3: **for** i in range 1 to $N - w + 1$ **do**
4: **Extract windows:** $\mathbf{x}^{(i:i+w-1)}$ and $\mathbf{a}^{(i:i+w-1)}$ of size w.
5: **Sample** $t \sim U(1, T)$ and $\epsilon^{(i:i+w-1)} \sim \mathcal{N}(0, I)$
6: **Forward noise**:

$$\mathbf{x}_t^{(i:i+w-1)} = \sqrt{\bar{\alpha}_t}\mathbf{x}^{(i:i+w-1)} + \sqrt{1-\bar{\alpha}_t}\epsilon^{(i:i+w-1)}$$

7: **Estimate** noise:

$$\hat{\epsilon}^{(i:i+w-1)} = f_\theta(\mathbf{a}^{(i:i+w-1)}, \mathbf{x}_t^{(i:i+w-1)}, t)$$

8: **Compute loss**:

$$\mathcal{L} = \mathbb{E}_{i \sim N-w+1} \left\| \epsilon^{(i:i+w-1)} - \hat{\epsilon}^{(i:i+w-1)} \right\|^2$$

9: **Update** f_θ parameters using $\mathcal{L}$.

3.4 Signal-conditioned Inference

With our metadata-conditioned denoiser (Sec. 3.3), we additionally condition on the observed signals that are injected directly at inference, which we detail as follows:

3.4.1 Pre-processing for data windows. We define the test set $\mathcal{X} = \{\mathbf{x}^{(i)}\}_{i=1}^{M}$, where $\mathbf{x}^{(i)}$ indicates the observed signals, and the unknown values are left empty, representing the portions to be generated. The complete set of metadata, $\mathcal{A} = \{\mathbf{a}^{(i)}\}_{i=1}^{M}$ is available as part of the setup, consistent with the problem definition (Sec. 2.3). A corresponding mask $\mathcal{M} = \{\mathbf{m}^{(i)}\}_{i=1}^{M}$ is defined, where $\mathbf{m}^{(i)} = 1$ indicates that $\mathbf{x}^{(i)}$ requires generation and $\mathbf{m}^{(i)} = 0$ indicates that the corresponding signals are

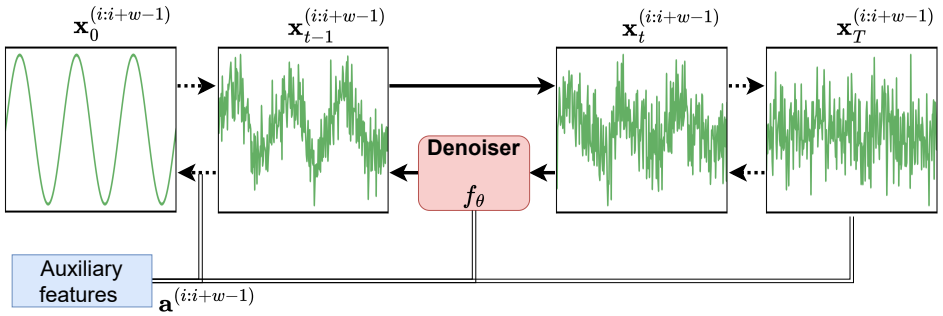


Fig. 5. Metadata-conditioned denoiser training.

observed. We first divide the test set into overlapping time windows of size w, spaced by a stride s, yielding $\mathcal{X}_w = \{\mathbf{x}_w^{(j)}\}_{j=1}^{\lceil \frac{M-w}{s} \rceil+1}$, $\mathcal{M}_w = \{\mathbf{m}_w^{(j)}\}_{j=1}^{\lceil \frac{M-w}{s} \rceil+1}$, and $\mathcal{A}_w = \{\mathbf{a}_w^{(j)}\}_{j=1}^{\lceil \frac{M-w}{s} \rceil+1}$. Following this, we proceed with the generation process detailed in Algorithm 2, which iterates through steps of $t = T, T-1 \ldots 1$. At each step, the metadata-conditioned denoiser first produces an intermediate sample, which is then adjusted using conditional loss gradients.

Algorithm 2 Parallel Denoising with Inference Conditioning

1: **Input:** Windowed metadata ($\mathcal{A}_w$), masks ($\mathcal{M}_w$), and signals ($\mathcal{X}_w$); Timesteps M, window size w; Stride s; mini-batch size b; strength η; Diffusion parameters $\{\alpha_t, \bar{\alpha}_t\}$; denoiser f_θ.
2: **Initialize outputs:**
$$\hat{\mathcal{X}}_w = \{\hat{\mathbf{x}}_{w,T}^{(j)} \sim \mathcal{N}(0, I)\}_{j=1}^{\lceil (M-w)/s \rceil+1}$$
3: **Divide** $\mathcal{X}_w, \mathcal{A}_w, \mathcal{M}_w$ **into** $(M-w)/(b \times s)$ **mini-batches**
4: **for** each mini-batch **do**
5: **for** $(\mathbf{x}_w^{(j)}, \mathbf{a}_w^{(j)}, \mathbf{m}_w^{(j)})$ in mini-batch in **parallel**: **do**
6: **for** step $t = T, T-1, \ldots, 1$ **do**
7: **Dirty estimate** $\hat{\mathbf{x}}_{w,0}^{(j)}$ using eq. (4)
8: **Unconditional denoising:** $\hat{\mathbf{x}'}_{w,t-1}^{(j)}$ using eq. (7)
9: **Compute conditional loss** $\mathcal{L}_{\text{cond}}^{(j)}$ using eq. (5)
10: **Gradient correction** of $\hat{\mathbf{x}'}_{w,t-1}^{(j)}$ using eq. (9)
11: **Re-introduce observations:**
$$\hat{\mathbf{x}}_{w,0}^{(j)} = (1 - \mathbf{m}_w^{(j)}) \cdot \mathbf{x}_w^{(j)} + \mathbf{m}_w^{(j)} \cdot \hat{\mathbf{x}}_{w,0}^{(j)}$$
12: **Merge windows:**
$$\hat{\mathcal{X}} = \hat{\mathbf{x}}_{w,0}^{(1)} \cup \left(\bigcup_{j \geq 2} \hat{\mathbf{x}}_{w,0}^{(j)(w-s+1:w)} \right)$$
13: **return** $\hat{\mathcal{X}}$

3.4.2 Metadata-Conditioned Generation. We first initialise the signal to be generated ($\hat{\mathcal{X}}_w$) in line 2. Then we divide $\mathcal{X}_w$, $\mathcal{M}_w$, and $\mathcal{A}_w$ into mini-batches of size b, each containing samples of w timesteps (line 3). To condition on the observed signals *and* metadata, we first get a dirty estimate

of the fully-denoised signal ($\hat{\mathbf{x}}_{w,0}^{(j)}$) using our metadata-conditioned denoiser (line 7). We get this estimate by reversing the fast-forward noising step in eq. (2) as follows:

$$\hat{\mathbf{x}}_{w,0}^{(j)} = \frac{\hat{\mathbf{x}}_{w,t}^{(j)} - \sqrt{1-\bar{\alpha}_t} \cdot f_\theta\left(\mathbf{a}_w^{(j)}, \hat{\mathbf{x}}_{w,t}^{(j)}, t\right)}{\sqrt{\bar{\alpha}_t}} \tag{4}$$

Here, $\hat{\mathbf{x}}_{w,t}^{(j)}$ is the model's estimate at the t^{th} diffusion step.

3.4.3 Sample adjustment with gradient inversion. The denoiser f_θ estimates samples by only conditioning on the metadata. To further factor in the observed signal values, we correct the samples using the gradients from the *conditional loss* (line 9), defined as follows:

$$\mathcal{L}_{\text{cond}}^{(j)} = \mathcal{L}_{\text{self}}^{(j)} + \mathcal{L}_{\text{stitch}}^{(j)} \tag{5}$$

This loss consists of two components: the *self-guidance* loss ($\mathcal{L}_{\text{self}}^{(j)}$), which enforces consistency with the observed signals *within* each time window, and the *stitch* loss ($\mathcal{L}_{\text{stitch}}^{(j)}$), which promotes coherence *across* overlapping windows.

Self-Guidance: The *self-guidance loss* [22] ensures that the generated outputs match the observed signals within a given time window. It is defined as follows:

$$\mathcal{L}_{\text{self}}^{(j)} = \|(1 - \mathbf{m}_w^{(j)}) \odot (\hat{\mathbf{x}}_{w,0}^{(j)} - \mathbf{x}_w^{(j)})\|^2 \tag{6}$$

Here, the term $1 - \mathbf{m}_w^{(j)}$ is used to compute the loss only on the observed timesteps, enforcing local consistency within a window.

Stitching: Since self-guidance treats each window independently, it may produce inconsistencies across window boundaries by ignoring overlaps. These overlaps are crucial for maintaining temporal continuity, and failing to align them can result in incoherent transitions. *Stitching* overcomes this limitation by enforcing consistency in overlapping regions, restricting the solution space to a more realistic subset than those produced by self-guidance alone. For a given window $\hat{\mathbf{x}}_{w,t}^{(j)}$, we first denoise one step $\hat{\mathbf{x}'}_{w,t-1}^{(j)}$ *unconditionally* using eq. (3) (line 8) :

$$\hat{\mathbf{x}'}_{w,t-1}^{(j)} = \frac{1}{\sqrt{\alpha_t}}\left(\hat{\mathbf{x}}_{w,t}^{(j)} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot f_\theta\left(\mathbf{a}_w^{(j)}, \hat{\mathbf{x}}_{w,t}^{(j)}, t\right)\right) + \sigma_t z \tag{7}$$

We then *"stitch"* these uncorrected estimates across window boundaries to ensure coherence. Specifically, for a window of size w and stride s, the overlap consists of the first $w - s$ timesteps of $\hat{\mathbf{x}'}_{w,t-1}^{(j)}$ and the last $w - s$ timesteps of $\hat{\mathbf{x}'}_{w,t-1}^{(j-1)}$. The stitch loss then penalises discrepancies in this overlap as follows[3]:

$$\mathcal{L}_{\text{stitch}}^{(j)} = \|\hat{\mathbf{x}'}_{w,t-1}^{(j)(1:w-s)} - \hat{\mathbf{x}'}_{w,t-1}^{(j-1)(1+s:w)}\|^2 \tag{8}$$

Adjustment: To dynamically condition on the observed signals, we correct the unconditional outputs from eq. (7) using the conditional loss gradients scaled by guidance coefficient η (line 10):

$$\hat{\mathbf{x}}_{w,t-1}^{(j)} = \hat{\mathbf{x}'}_{w,t-1}^{(j)} - \eta \nabla_{\hat{\mathbf{x}}_{w,t}^{(j)}} \mathcal{L}_{\text{cond}}^{(j)}, \tag{9}$$

where $\mathcal{L}_{\text{cond}}$ is as defined in eq. (5), $z \sim \mathcal{N}(0, I)$, and $\sigma_t = (1-\alpha_t) \times \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}$. At the end of the denoising process, we reintroduce the observations (line 11) and merge the overlapping windows

[3]For the first window, we define $\mathcal{L}_{\text{stitch}}^{(1)} = 0$, as there are no preceding windows.

(line 12), to yield the final sequence. Figure 6 illustrates the conditional denoising process using the combination of self-guidance and stitching.

To summarise, eq. (7) unconditionally denoises the sample without enforcing any constraints. If left unmodified, the reverse diffusion trajectory would follow this path, ignoring both observations and inter-window coherence. Equation (9) adjusts the trajectory using the gradient of the conditional loss, which consists of two terms. The self-guidance loss eq. (6) enforces alignment with the observed signal values *within* each window, while the stitching eq. (8) enforces consistency *across* overlapping windows. Together, these components nudge the model at every denoising step to yield samples that adhere to observations and remain globally coherent.

3.5 Theoretical Analysis

Here, we formalise how the inference-time conditional loss, comprising self-guidance and stitching, constrains generation towards a tighter solution space that promotes global temporal coherence and alignment with the given signal observations.

Definitions. Let $\mathcal{X}^{\delta}_{\text{cond}}$ be the set of samples for which the conditional loss is at most $\delta \geq 0$, i.e., they respect both the overlap consistencies and the observed signals within windows up to a tolerance δ. As a special case, $\mathcal{X}^{0}_{\text{cond}}$ represents the set of samples with zero-conditional loss. Similarly, let $\mathcal{X}^{\delta}_{\text{stitch}}$ and $\mathcal{X}^{\delta}_{\text{self}}$ be the set of samples with at most δ stitch and self-guidance losses respectively. Finally, we define $\mathcal{X}_{\text{real}}$ as the set of real samples, which must obey the following conditions: (a) exactly match the known values, i.e., zero self-guidance loss; and (b) be temporally coherent, i.e., any overlapping windows yield identical values on their shared timesteps (zero stitch loss). A violation of condition (b) would result in conflicting predictions at the same timestep upon merging, which is not possible for a real time series.

Proposition 3.1. *With sets $\mathcal{X}_{real}$, $\mathcal{X}^{\delta}_{self}$, $\mathcal{X}^{\delta}_{stitch}$, and $\mathcal{X}^{\delta}_{cond}$ defined as above, we have the following chain of inclusions:*

$$\mathcal{X}_{real} \subseteq \mathcal{X}^{0}_{cond} \subseteq \mathcal{X}^{\delta}_{cond} \subseteq \mathcal{X}^{\delta}_{self}.$$

Proof. By definition, the conditional loss is the sum of the self-guidance loss and the stitch loss, with all losses being non-negative (see eq. (5)). Therefore, for any $x \in \mathcal{X}^{\delta}_{\text{cond}}$, we have:

$$\mathcal{L}_{\text{cond}}(x) \leq \delta \quad \Rightarrow \quad \mathcal{L}_{\text{self}}(x) \leq \delta \quad \text{and} \quad \mathcal{L}_{\text{stitch}}(x) \leq \delta.$$

This implies that for all $x \in \mathcal{X}^{\delta}_{\text{cond}}$:

$$x \in \mathcal{X}^{\delta}_{\text{self}} \cap \mathcal{X}^{\delta}_{\text{stitch}} \Rightarrow \mathcal{X}^{\delta}_{\text{cond}} \subseteq \mathcal{X}^{\delta}_{\text{self}}. \tag{I}$$

Next, since the zero conditional loss is a special case of loss less than or equal to δ, we have:

$$\mathcal{X}^{0}_{\text{cond}} \subseteq \mathcal{X}^{\delta}_{\text{cond}}. \tag{II}$$

Finally, according to the earlier definition, any $x \in \mathcal{X}_{\text{real}}$ must have zero self-guidance loss and zero stitch loss. Hence, for all $x \in \mathcal{X}_{\text{real}}$, we have:

$$x \in \mathcal{X}^{0}_{\text{self}} \cap \mathcal{X}^{0}_{\text{stitch}} = \mathcal{X}^{0}_{\text{cond}} \Rightarrow \mathcal{X}_{\text{real}} \subseteq \mathcal{X}^{0}_{\text{cond}}. \tag{III}$$

Combining (I), (II), and (III), we then obtain the chain of inclusions:

$$\mathcal{X}_{\text{real}} \subseteq \mathcal{X}^{0}_{\text{cond}} \subseteq \mathcal{X}^{\delta}_{\text{cond}} \subseteq \mathcal{X}^{\delta}_{\text{self}}.$$

□

Proposition 3.1 shows that the conditional loss restricts generation to a subset closer to the real data space. While self-guidance ensures agreement with the known values within each window, it does not constrain the unknown regions. As a result, overlapping windows may assign conflicting

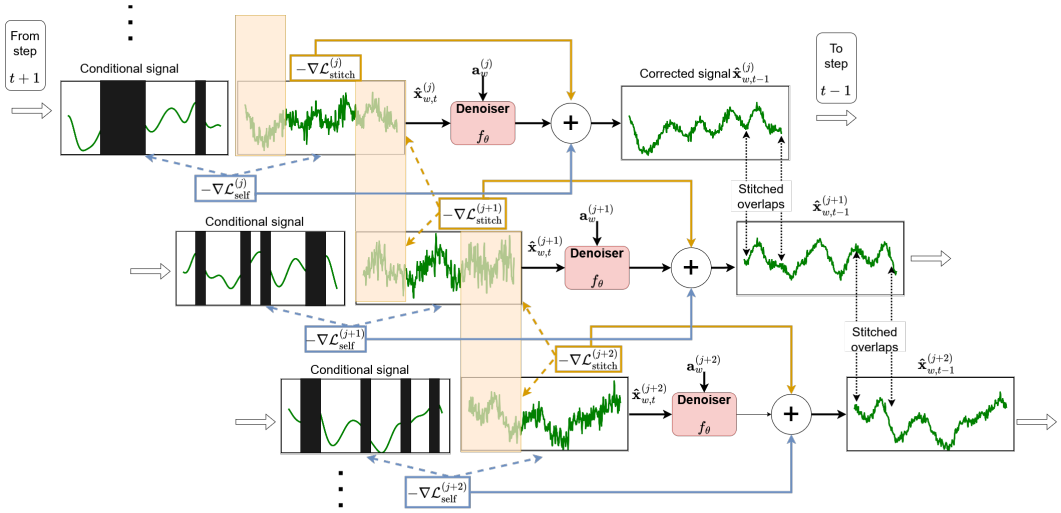


Fig. 6. Parallel Inference Conditioning with Stitching and Self-Guidance Loss. The denoising process flows from left to right. At each step t, the model receives denoised windows from step $t+1$. It then refines these using two types of gradient corrections: the self-guidance loss (in blue), computed using observed signals, and the stitching loss (in orange) on the overlaps. These corrections update the metadata-conditioned estimate before passing it to the next denoising step $t-1$.

values to the same timestep, yielding inconsistent samples after merging. Stitching penalises inconsistencies in the overlaps. Hence, the sample set with zero conditional loss is a subset of the self-guided sample space and contains all realistic samples. Since the model trains on realistic and temporally coherent samples, the inference-time conditional loss does not push it away from the training distribution. Rather, it keeps samples within the real space by bringing the model's outputs closer to the characteristics it was initially optimised to reproduce.

While zero conditional loss is the limit, the proposition implies that even minimising it to a small value $\delta > 0$ brings the samples closer to realistic behaviour, even without exact convergence. Crucially, zero conditional loss is a necessary but not a sufficient condition for realism. Hence, there may exist samples that perfectly match the observations with coherent overlaps but still diverge from the true sequence. This observation reflects an inherent ambiguity in conditional generation. If multiple completions can satisfy the given conditions, minimising the conditional loss improves plausibility, but cannot guarantee a match to the ground truth.

4 Pipelined Parallel Design of WaveStitch

The conditioning mechanism described in Sec. 3.4 could be applied autoregressively by processing each window sequentially. However, most steps in Algorithm 2 can be performed independently across windows. Even the conditional loss computation (Algorithm 2, line 9) can be made parallel-compatible by a parallel *shift* (or *roll*) operation to align overlapping regions, to compute the pairwise loss between consecutive windows concurrently. This process resembles a *pipeline*, as shown in Figure 6. At each denoising step, non-overlapping parts adjust to maintain coherence with their neighbours. Although these adjustments are only enforced with the neighbouring window, coherence gradually propagates over the whole pipeline over successive steps.

The stride s is crucial in balancing efficiency and coherence. When s is large enough to eliminate overlap, the process becomes a *divide-and-conquer* strategy [24, 46], that synthesises windows independently. Conversely, smaller strides enforce stronger dependencies between windows due to

Table 2. Datasets and Experiment Setup. For BeijingAirQuality (BQ), AustraliaTourism (AT), MetroTraffic (MT), PanamaEnergy (PE), and RossmanSales (RS). Length refers to the sequence length of the test set, equalling the number of samples for the ***R***-level condition.

Dataset	Metadata Features	Channels	Length	Condition
BQ	Year, Station, Month, Day, Hour	11	8496	***R***: $(2017, *, *, *, *)$ ***I***: $(2017, *, 2, *, *)$ ***B***: $(2017, *, *, *, 11)$
MT	Year, Month, Day, Hour	5	7949	***R***: $(2018, *, *, *)$ ***I***: $(2018, *, 15, *)$ ***B***: $(2018, *, *, 6)$
PE	Year, Month, Day, Hour, City	4	12819	***R***: $(2020, *, *, *, *)$ ***I***: $(2020, *, 5, *, *)$ ***B***: $(2020, *, *, *, San)$
RS	Year, Month, Day, Store	2	1757	***R***: $(2015, *, *, *)$ ***I***: $(2015, 3, *, *)$ ***B***: $(2015, *, *, 9)$
AT	Year, Month, State, Region, Purpose	1	1232	***R***: $(2016, *, *, *, *)$ ***I***: $(2016, *, Queensland, *, *)$ ***B***: $(2016, *, *, *, Holiday)$

larger overlaps. In the extreme case of $s = 1$, maximum coherence is achieved as nearly all timesteps are shared between consecutive windows. However, the optimal stride depends on the time series itself. Smaller strides ensure better alignment for signals with strong temporal dependencies, while larger strides may suffice for weaker ones. This flexibility allows our method to adapt to varying degrees of temporal dependence for efficient and coherent signal generation.

Time Complexity and Speedup: WaveStitch substantially improves the generation speed compared to autoregression due to parallelism. Using the notation in Algorithm 2, each dataset has around $\frac{M-w}{s}$ time windows, where M is the total length, w is the window size, and s is the stride. With sequential execution, the total cost is $O(T \cdot t_\theta(w) \cdot (M-w)/s)$, where $t_\theta(w)$ is the time taken to denoise a window by one step. In contrast, the parallel approach processes b windows in parallel per mini-batch, which significantly reduces the time complexity to $O(T \cdot t_\theta(w) \cdot \frac{M-w}{b \cdot s})$. This analysis suggests an *ideal speedup* factor of approximately b over the autoregressive approach, representing an upper bound on performance under optimal hardware utilisation.

5 Evaluation

We conduct a comprehensive set of experiments to evaluate how well WaveStitch addresses the following research questions:

RQ1: How effective is WaveStitch's dual-source and hybrid conditioning strategy in integrating metadata and observed signals compared to SOTA methods?
RQ2: How does the speed and quality of parallel synthesis compare to autoregressive generation?
RQ3: Does stitching improve performance compared to using only self-guidance or RePaint [7, 27]?
RQ4: What is the impact of alternative conditional loss formulations on the generation quality?
RQ5: Does WaveStitch generalise to random imputation tasks with varying missingness ratios?

System Configuration. We use a 12-core AMD Ryzen 9 5900 processor with an NVIDIA RTX 3090 GPU. For software, we use CUDA 12.1, Ubuntu 20.04.6, and Pytorch 2.2.1 with Python 3.12.

5.1 Datasets and Evaluation Framework

Datasets. We use public multivariate and univariate datasets from diverse domains: BeijingAirQuality (BQ) [6], MetroTraffic Volume (MT) [17], PanamaEnergy (PE) [1], RossmanSales (RS) [13], and AustraliaTourism (AT) [5]. Each dataset includes metadata, grouped in the order specified in Table 2,

reflecting how the data is organised. The time series channels for each dataset are as follows. For BQ, we use all gas sensor readings, temperature, pressure, dewpoint, rainfall, and wind speed from the first six stations, sorted alphabetically; for MT, we use temperature, rainfall, snowfall, traffic volume, and cloud cover levels; for PE, we use energy data such as T2M, QV2M, TQL, and W2M; for RS, we use sales and customer totals from stores 1-10; and for AT we use the number of trips.

Evaluation Tasks. Designing evaluation scenarios for our problem is not trivial. Random or standard 80-20 splits would disrupt temporal dependencies. To address this, we place all samples with a shared root-level value (e.g., a specific year) in the test set, ensuring unseen metadata combinations at inference. This approach supports coarse-grained imputation tasks, as well as fine-grained or mid-level tasks, by specifying lower-level metadata features within the test set.

To evaluate the model's performance across different scenarios, we use a structured approach that conditions metadata at three levels: *Root* (*R*), *Intermediate* (*I*), and *Bottom* (*B*), each corresponding to a different level in the grouping hierarchy and reflecting varying degrees of missing signal information and task granularity (see Table 2). Root Level *(R)* conditions fix the root-level metadata feature (e.g. *Year* in a *Year->Month->Day* hierarchy), creating tasks missing contiguous signal blocks (e.g., Figure 4a). Intermediate Level *(I)* tasks restrict mid-level metadata features, giving missing blocks of intermediate size (e.g., Figure 4b). Finally, Bottom Level *(B)* tasks restrict the last metadata feature, yielding fine-grained imputation scenarios (e.g., Figure 4c). As shown in Table 2, both *I* and *B* additionally condition on the root-level feature to ensure that samples remain within the test set. While these configurations represent only a subset of possible scenarios, they cover a broad range of missingness patterns, allowing us to evaluate the model under various conditions. For example, this setup can also capture random missingness by combining multiple *(B)* conditions.

Metrics. Imputation and generation are fundamentally different tasks and require different evaluation metrics. We treat forecasting as a special case of imputation, where all missing values appear only at the tail end of the time series, as discussed in Sec. 1 and explored in prior work [2, 22]. Imputation assumes the presence of *a* ground-truth signal and focuses on accurately recovering missing values. Hence, the Mean Squared Error (MSE) is apt here, as it measures a point-wise match between predictions and ground-truths. In contrast, generation allows for multiple plausible outputs and aims for high-level resemblance with the real signals. Hence, metrics that assess similarity at a more abstract or distributional level are more appropriate here. These include statistics like feature-wise correlations or differences in autocorrelation, which capture structural or periodic resemblance between real and generated signals, rather than strict one-to-one matches. Therefore, we use the *autocorrelation difference* (ACD) [44, 45] and *cross-feature correlations* (x-Corr) [18], in line with previous studies [3, 25, 30]. These capture higher-level similarities between the generated and ground-truth signals, such as similar periodic patterns (ACD) or inter-feature dependencies (x-Corr). Each metric's score is averaged over five trials to ensure robustness.

We compute the ACD between the real and synthetic signal across multiple lags per channel. Let $ACF^{(c)}_{real}(\tau)$ and $ACF^{(c)}_{synth}(\tau)$ represent the autocorrelation values [31] for lag τ in channel c for the real and synthetic data respectively. Given the maximum number of lags G and signal channels C, we can compute the difference between the real and generated autocorrelation values [44]:

$$ACD = \frac{1}{C \cdot G} \sum_{c=1}^{C} \sum_{\tau=1}^{G} \left| ACF^{(c)}_{real}(\tau) - ACF^{(c)}_{synth}(\tau) \right| \tag{10}$$

The ACD measures how similarly the generated and original signals evolve by comparing their autocorrelations across multiple time lags. For the cross-feature correlations, we calculate the pairwise Pearson correlation coefficients between all pairs of channels for both the real and synthetic

time series. We then compute the average absolute difference in these correlations between the real and synthetic data, as done in prior works [18, 30, 45].

5.2 Model Implementation and Baselines

We extend the implementations of several baselines from Table 10 for conditional generation. For all diffusion models, including ours, we use SSSD [2] as the denoiser backbone for consistency, which utilises structured state space layers (S4 layers [40]) and has outperformed transformer-based methods such as CSDI [42]. The architecture consists of 4 hidden layers, four residual layers, and uses residual, skip, and hidden dimensions of size 64. The model employs embeddings for diffusion steps, with dimensions of 32 (input), 64 (middle), and 64 (output). S4 layers are configured with a maximum sequence length of 100, a state dimension of 64, and are used bidirectionally with layer normalisation and no dropout.

To ensure fairness, we re-implement all diffusion baselines: SSSD [2], TimeWeaver [29], TSDiff [22], and TimeAutoDiff [41], using the same backbone architecture described above. SSSD is trained to impute randomly masked signal rows conditioned on the observed rows and metadata. We uniformly sample a number of timesteps between 0 and the total sequence length to randomly mask out during training. The model then trains to reconstruct the masked signals using the observed rows and the metadata. TimeWeaver, by design, conditions solely on metadata. To match this, we train it by masking out all signal rows, so that it learns to generate signals from just the metadata. We extend TSDiff to condition on metadata using the same metadata-conditioned denoiser as TimeWeaver. The observed signals are then conditioned on directly at inference time using self-guidance (see Sec. 2.2). We tried different scales for tuning TSDiff's guidance strength: 0.0, 0.5, 1.0, and 2.0, and report results for 0.5, since it had the best overall performance. For fairness, we adapt TimeAutoDiff from its original latent diffusion architecture to operate in real space using our shared backbone. It uses classifier-free guidance that conditions on labels (see Sec. 2.2), interpolating using two model variants. We adapt it to conditions on both the metadata and signal observations. One branch uses only metadata while the other conditions on both metadata and signal observations, like the SSSD baseline. We train each variant separately, and at inference, we interpolate between them using a guidance strength of 3.0, following prior work [16, 41].

We also compare against the GAN-based method TimeGAN [47], which consists of five networks: encoder, recovery (decoder), generator, supervisor, and discriminator. We extend the base model to incorporate metadata. Each network uses a three-layer GRU [8] with fully connected output heads.

For the diffusion backbones, we use a linear noise schedule with 200 noising and denoising steps, using $\alpha_1 = 0.9999$ and $\alpha_T = 0.98$. The guidance strength η is fixed at 0.1 for all datasets and task configurations. By default, we report WaveStitch's results using a stride of 8, as it achieved the best overall performance. We trained the diffusion backbones using the MSE loss between the estimated and actual noise, and we optimised TimeGAN using the L1-regularised reconstruction loss between the generated and ground-truth signals. We train all models for 300 epochs using a mini-batch size of 1024 and a learning rate of 10^{-4}. For consistency, we divided all the datasets using sliding windows of 32 timesteps (stride 1) for training, and tunable stride lengths at inference.

5.3 Comparison with SOTA Generators (RQ1)

Table 3 shows the average MSE between the generated and ground truth signals over five trials. We use multiple metrics to assess performance: MSE for imputation accuracy, and quality metrics ACD and x-Corr for generative tasks.

We observe that diffusion methods outperform TimeGAN, in line with earlier findings [11, 39]. MSEs generally decrease from R tasks to I and B tasks, since R tasks have a single contiguous missing block. In contrast, I and B tasks require generating several smaller blocks separated by observed

Table 3. Average MSE values (↓) for different datasets and tasks (*R*/*I*/*B*). Standard deviation in subscript; **best** (bold) and second-best scores marked.

	TimeGAN	TimeWeaver	TSDiff	SSSD	Time-AutoDiff	WaveStitch
(***R***)	$8.159_{.115}$	$0.295_{.040}$	$1.223_{.007}$	$1.329_{.004}$	$10.406_{.433}$	$\mathbf{0.152_{.006}}$
AT (***I***)	$8.373_{.062}$	$1.155_{.335}$	$1.480_{.003}$	$1.490_{.011}$	$6.200_{.407}$	$\mathbf{0.246_{.009}}$
(***B***)	$9.717_{.125}$	$2.253_{.006}$	$1.567_{.032}$	$0.652_{.030}$	$4.021_{.144}$	$\mathbf{0.140_{.004}}$
(***R***)	$8.887_{.072}$	$0.533_{.018}$	$0.964_{.009}$	$1.009_{.010}$	$3.957_{.040}$	$\mathbf{0.512_{.009}}$
MT (***I***)	$9.194_{.175}$	$0.603_{.075}$	$0.768_{.027}$	$0.701_{.061}$	$1.667_{.088}$	$\mathbf{0.396_{.019}}$
(***B***)	$7.817_{.222}$	$0.874_{.042}$	$0.200_{.010}$	$0.128_{.005}$	$0.702_{.033}$	$\mathbf{0.111_{.008}}$
(***R***)	$3.148_{.074}$	$1.529_{.022}$	$1.763_{.017}$	$\mathbf{1.330_{.022}}$	$16.901_{.122}$	$1.481_{.019}$
BQ (***I***)	$3.552_{.177}$	$1.067_{.021}$	$1.422_{.025}$	$1.055_{.016}$	$23.695_{.427}$	$\mathbf{0.981_{.015}}$
(***B***)	$2.170_{.039}$	$1.595_{.045}$	$0.162_{.007}$	$0.107_{.002}$	$1.085_{.019}$	$\mathbf{0.099_{.009}}$
(***R***)	$4.052_{.003}$	$0.645_{.014}$	$0.893_{.021}$	$0.640_{.005}$	$4.680_{.066}$	$\mathbf{0.598_{.006}}$
RS (***I***)	$4.065_{.015}$	$0.588_{.043}$	$0.827_{.032}$	$0.569_{.014}$	$2.839_{.081}$	$\mathbf{0.537_{.023}}$
(***B***)	$2.731_{.020}$	$0.965_{.059}$	$0.458_{.011}$	$0.440_{.005}$	$1.019_{.006}$	$\mathbf{0.149_{.002}}$
(***R***)	$6.768_{.007}$	$1.074_{.027}$	$1.667_{.005}$	$2.496_{.008}$	$4.127_{.031}$	$\mathbf{1.006_{.013}}$
PE (***I***)	$6.138_{.102}$	$0.961_{.114}$	$1.179_{.061}$	$1.121_{.055}$	$9.279_{.422}$	$\mathbf{0.587_{.128}}$
(***B***)	$4.092_{.009}$	$1.387_{.029}$	$0.469_{.002}$	$0.216_{.001}$	$1.378_{.003}$	$\mathbf{0.165_{.003}}$

Table 4. Average ACD values (↓) for each dataset and task (*R*/*I*/*B*). Standard deviation in subscript; **best** (bold) and second-best scores marked.)

	TimeGAN	TimeWeaver	TSDiff	SSSD	Time-AutoDiff	WaveStitch
(***R***)	$0.259_{.015}$	$0.024_{.003}$	$0.090_{.003}$	$0.131_{.003}$	$0.151_{.004}$	$\mathbf{0.018_{.002}}$
AT (***I***)	$0.253_{.006}$	$0.093_{.021}$	$0.256_{.002}$	$0.129_{.003}$	$0.226_{.010}$	$\mathbf{0.038_{.001}}$
(***B***)	$0.255_{.024}$	$0.126_{.002}$	$0.090_{.001}$	$0.057_{.001}$	$0.111_{.005}$	$\mathbf{0.022_{.000}}$
(***R***)	$0.467_{.004}$	$0.191_{.005}$	$0.308_{.006}$	$0.332_{.002}$	$0.343_{.006}$	$\mathbf{0.142_{.007}}$
MT (***I***)	$0.437_{.008}$	$0.208_{.025}$	$0.159_{.008}$	$0.181_{.012}$	$0.238_{.007}$	$\mathbf{0.134_{.010}}$
(***B***)	$0.403_{.024}$	$0.234_{.038}$	$0.057_{.001}$	$0.045_{.000}$	$0.063_{.002}$	$\mathbf{0.042_{.001}}$
(***R***)	$0.191_{.002}$	$0.154_{.005}$	$0.284_{.002}$	$0.303_{.004}$	$0.285_{.004}$	$\mathbf{0.127_{.003}}$
BQ (***I***)	$0.231_{.006}$	$\mathbf{0.104_{.003}}$	$0.171_{.006}$	$0.219_{.004}$	$0.321_{.001}$	$0.116_{.010}$
(***B***)	$0.223_{.005}$	$0.265_{.021}$	$0.025_{.001}$	$\mathbf{0.018_{.000}}$	$0.096_{.001}$	$\mathbf{0.020_{.001}}$
(***R***)	$0.289_{.002}$	$0.150_{.001}$	$\mathbf{0.131_{.002}}$	$0.143_{.001}$	$0.218_{.005}$	$0.162_{.001}$
RS (***I***)	$0.289_{.005}$	$0.181_{.004}$	$0.176_{.005}$	$\mathbf{0.137_{.006}}$	$0.194_{.010}$	$0.183_{.007}$
(***B***)	$0.311_{.004}$	$0.241_{.005}$	$0.126_{.006}$	$0.128_{.005}$	$0.213_{.000}$	$\mathbf{0.051_{.001}}$
(***R***)	$0.410_{.000}$	$0.321_{.003}$	$0.388_{.001}$	$0.422_{.001}$	$0.424_{.001}$	$\mathbf{0.294_{.005}}$
PE (***I***)	$0.382_{.002}$	$0.258_{.020}$	$0.309_{.011}$	$0.252_{.005}$	$0.414_{.013}$	$\mathbf{0.178_{.032}}$
(***B***)	$0.417_{.001}$	$0.353_{.024}$	$0.081_{.001}$	$\mathbf{0.044_{.000}}$	$0.333_{.000}$	$0.081_{.007}$

signals. When computing the MSE ratio by dividing the best-performing baseline's MSE with that of algo's for each task and dataset, the average ratio is **1.81x**, highlighting algo's significant advantage over SOTA baselines.

Table 5. Average x-Corr values (↓) for each dataset and task (R/I/B). Standard deviation in subscript; **best** (bold) and second-best scores marked.

	TimeGAN	TimeWeaver	TSDiff	SSSD	Time-AutoDiff	WaveStitch
(***R***)	$0.558_{.002}$	$\mathbf{0.102_{.006}}$	$0.120_{.003}$	$0.204_{.004}$	$0.150_{.005}$	$\underline{0.104_{.003}}$
MT (***I***)	$0.587_{.017}$	$0.203_{.024}$	$\underline{0.180_{.013}}$	$\mathbf{0.179_{.026}}$	$0.205_{.020}$	$0.206_{.013}$
(***B***)	$0.527_{.015}$	$0.105_{.015}$	$\mathbf{0.062_{.005}}$	$0.091_{.005}$	$0.089_{.011}$	$\underline{0.074_{.005}}$
(***R***)	$0.191_{.004}$	$\mathbf{0.156_{.002}}$	$\underline{0.178_{.003}}$	$0.284_{.003}$	$0.341_{.002}$	$\mathbf{0.156_{.008}}$
BQ (***I***)	$0.232_{.006}$	$\underline{0.191_{.015}}$	$\mathbf{0.187_{.003}}$	$0.200_{.004}$	$0.371_{.002}$	$\mathbf{0.187_{.015}}$
(***B***)	$0.181_{.006}$	$0.285_{.032}$	$0.047_{.002}$	$\mathbf{0.019_{.000}}$	$0.230_{.002}$	$\underline{0.038_{.003}}$
(***R***)	$0.703_{.005}$	$\underline{0.030_{.001}}$	$0.046_{.001}$	$0.026_{.000}$	$\mathbf{0.004_{.000}}$	$0.031_{.001}$
RS (***I***)	$0.741_{.015}$	$\underline{0.006_{.001}}$	$\mathbf{0.004_{.002}}$	$0.009_{.001}$	$\mathbf{0.004_{.002}}$	$0.009_{.001}$
(***B***)	$0.829_{.018}$	$0.127_{.008}$	$0.075_{.003}$	$0.073_{.002}$	$\underline{0.047_{.001}}$	$\mathbf{0.011_{.001}}$
(***R***)	$0.412_{.001}$	$\mathbf{0.145_{.005}}$	$0.298_{.003}$	$\underline{0.195_{.001}}$	$0.255_{.002}$	$0.228_{.006}$
PE (***I***)	$0.535_{.007}$	$\mathbf{0.175_{.043}}$	$0.321_{.021}$	$0.344_{.009}$	$0.293_{.023}$	$\underline{0.232_{.017}}$
(***B***)	$0.366_{.001}$	$0.249_{.025}$	$0.099_{.001}$	$\mathbf{0.031_{.000}}$	$0.160_{.001}$	$\underline{0.056_{.003}}$

Among the baselines, SSSD, TSDiff and TimeWeaver generally perform the best. TimeWeaver performs well on R (coarse-grained) tasks, but underperforms on fine-grained tasks since it ignores the local cues from neighbouring signal observations. In contrast, TSDiff and SSSD do better on fine-grained tasks since they also condition on observed signals. SSSD, trained by randomly masking signal rows, likely encounters many patterns where masked entries are neighboured by observations, possibly explaining its stronger performance on fine-grained tasks. However, this bias towards the masking patterns seen during training could explain its underperformance on coarser-grained tasks. TimeAutoDiff performs poorly across the board, likely due to its sensitivity to the guidance strength and incompatibilities from using two models.

For synthesis tasks, we assess methods based on their ability to preserve temporal patterns and inter-feature dependencies, using ACD and x-Corr scores (Table 4 and Table 5). The AT dataset is excluded from Table 5 since it is univariate and does not have inter-feature dependencies. We see that WaveStitch often performs the best or second-best. Exceptions are likely due to the conditioning being biased towards minimising an MSE-like loss, which may not always yield the best results for ACD and x-Corr. Nevertheless, its performance is still competitive.

Again, while TimeWeaver performs well at root-level tasks, it underperforms on fine-grained tasks. In contrast, conditioning on the signals observed enables TSDiff, SSSD, and WaveStitch to generate more accurate values. TimeGAN consistently underperforms across all metrics, further highlighting the advantage of diffusion-based models for generating high-quality time series data.

5.4 Ablation: Parallel vs. Autoregressive (RQ2)

Table 6 shows the MSE, average runtime (seconds) and the total number of denoiser calls of the parallel (WaveStitch) and autoregressive (WaveStitchAR) implementations under varying stride lengths. We average all results over five trials, using a window size of 32 consistent across all variants, with 200 denoising steps and a mini-batch size of 1024. We also report TimeGAN results to contextualise the tradeoff between generation quality and speed against a fast generative baseline.

Overall, the MSE scores between the parallel (WaveStitch) and autoregressive (WaveStitchAR) variants are comparable, with the parallel version generally doing better. This result indicates that parallel synthesis with stitching maintains similar coherence to sequential generation. While

Table 6. Autoregressive (WaveStitchAR) versus parallel generation across various strides (1, 8, 16, 32), datasets and tasks. The best scores are marked in **bold**.

		R			*I*			*B*		
	Method	**Avg. MSE ↓**	**#Calls ↓**	**Avg. Time ↓**	**Avg. MSE ↓**	**#Calls ↓**	**Avg. Time ↓**	**Avg. MSE ↓**	**#Calls ↓**	**Avg. Time ↓**
AT	WaveStitchAR-8	$0.214_{0.033}$	30800	$298.644_{2.018}$	$\mathbf{0.239_{0.004}}$	5200	$50.279_{0.449}$	$0.145_{0.003}$	30800	$297.488_{1.731}$
	WaveStitchAR-16	$0.177_{0.022}$	15400	$148.225_{1.224}$	$0.256_{0.039}$	3000	$28.859_{0.546}$	$0.159_{0.031}$	15400	$149.243_{0.529}$
	WaveStitchAR-32	$0.290_{0.041}$	7600	$73.919_{0.901}$	$0.350_{0.094}$	1800	$17.511_{0.563}$	$\mathbf{0.138_{0.002}}$	7600	$73.957_{0.628}$
	WaveStitch-1	$0.167_{0.008}$	400	$17.838_{0.650}$	$0.288_{0.045}$	400	$17.784_{0.842}$	$0.152_{0.003}$	400	$17.518_{0.572}$
	WaveStitch-8	$\mathbf{0.152_{0.006}}$	200	$\mathbf{5.258_{0.579}}$	$0.246_{0.009}$	200	$\mathbf{5.447_{0.744}}$	$0.140_{0.004}$	200	$\mathbf{5.283_{0.623}}$
	WaveStitch-16	$0.166_{0.012}$	200	$5.642_{0.640}$	$0.249_{0.005}$	200	$5.819_{0.615}$	$0.140_{0.004}$	200	$5.336_{0.748}$
	WaveStitch-32	$0.304_{0.062}$	200	$5.549_{0.588}$	$0.715_{0.147}$	200	$5.582_{0.588}$	$0.194_{0.065}$	200	$5.368_{0.645}$
	TimeGAN-32	$8.159_{0.115}$	1	$0.020_{0.036}$	$8.373_{0.062}$	1	$0.019_{0.037}$	$9.717_{0.125}$	1	$0.096_{0.010}$
MT	WaveStitchAR-8	$0.827_{0.111}$	198600	$1937.021_{6.488}$	$0.367_{0.017}$	8400	$83.225_{1.212}$	$0.130_{0.009}$	56600	$546.098_{1.696}$
	WaveStitchAR-16	$\mathbf{0.504_{0.042}}$	99200	$971.742_{3.573}$	$\mathbf{0.350_{0.068}}$	5200	$51.712_{0.523}$	$0.119_{0.014}$	55600	$540.363_{2.691}$
	WaveStitchAR-32	$0.548_{0.007}$	49600	$488.252_{1.868}$	$0.369_{0.025}$	3400	$33.458_{0.541}$	$0.125_{0.008}$	46600	$451.989_{1.995}$
	WaveStitch-1	$0.522_{0.011}$	1600	$94.320_{0.643}$	$0.390_{0.061}$	1600	$94.384_{1.069}$	$\mathbf{0.109_{0.008}}$	1600	$94.499_{1.471}$
	WaveStitch-8	$0.512_{0.009}$	200	$11.262_{0.575}$	$0.396_{0.019}$	200	$11.497_{0.533}$	$0.111_{0.008}$	200	$11.164_{0.533}$
	WaveStitch-16	$0.506_{0.012}$	200	$6.848_{0.509}$	$0.378_{0.055}$	200	$7.059_{0.479}$	$0.115_{0.007}$	200	$6.822_{0.542}$
	WaveStitch-32	$0.547_{0.016}$	200	$\mathbf{5.827_{0.499}}$	$0.410_{0.038}$	200	$\mathbf{5.901_{0.629}}$	$0.121_{0.011}$	200	$\mathbf{5.709_{0.637}}$
	TimeGAN-32	$8.887_{0.072}$	1	$0.027_{0.050}$	$9.194_{0.175}$	1	$0.019_{0.036}$	$7.817_{0.222}$	1	$0.020_{0.038}$
BQ	WaveStitchAR-8	$1.764_{0.526}$	212400	$2070.688_{9.150}$	$\mathbf{0.752_{0.014}}$	100800	$977.191_{4.176}$	$0.096_{0.005}$	70800	$686.126_{4.232}$
	WaveStitchAR-16	$2.650_{0.098}$	106200	$1033.736_{3.506}$	$1.132_{0.139}$	51000	$503.286_{2.742}$	$0.104_{0.006}$	70800	$685.728_{1.410}$
	WaveStitchAR-32	$1.511_{0.010}$	53000	$510.891_{1.523}$	$1.034_{0.013}$	26000	$255.775_{1.753}$	$\mathbf{0.095_{0.008}}$	53000	$523.066_{1.998}$
	WaveStitch-1	$1.523_{0.022}$	1800	$101.472_{0.605}$	$1.043_{0.015}$	1800	$101.482_{0.607}$	$0.104_{0.008}$	1800	$101.603_{0.719}$
	WaveStitch-8	$\mathbf{1.481_{0.019}}$	400	$17.776_{0.634}$	$0.981_{0.015}$	400	$17.564_{0.733}$	$0.099_{0.009}$	400	$17.455_{0.762}$
	WaveStitch-16	$1.488_{0.025}$	200	$7.294_{0.518}$	$0.999_{0.011}$	200	$7.319_{0.609}$	$0.101_{0.010}$	200	$7.377_{0.603}$
	WaveStitch-32	$1.530_{0.014}$	200	$\mathbf{5.931_{0.523}}$	$1.051_{0.015}$	200	$\mathbf{5.987_{0.506}}$	$0.111_{0.011}$	200	$\mathbf{5.999_{0.768}}$
	TimeGAN-32	$3.148_{0.074}$	1	$0.026_{0.047}$	$3.552_{0.177}$	1	$0.029_{0.055}$	$2.170_{0.039}$	1	$0.019_{0.037}$
RS	WaveStitchAR-8	$0.647_{0.007}$	43800	$425.083_{1.988}$	$0.606_{0.022}$	6600	$64.161_{0.769}$	$0.173_{0.004}$	34800	$338.959_{1.832}$
	WaveStitchAR-16	$0.609_{0.011}$	21800	$211.279_{0.999}$	$0.581_{0.045}$	3400	$32.882_{0.559}$	$0.156_{0.001}$	21800	$211.105_{0.409}$
	WaveStitchAR-32	$0.642_{0.007}$	10800	$102.649_{0.884}$	$0.643_{0.028}$	1800	$17.595_{0.532}$	$0.155_{0.001}$	10800	$105.553_{0.574}$
	WaveStitch-1	$\mathbf{0.593_{0.008}}$	400	$20.089_{0.579}$	$0.542_{0.022}$	400	$20.110_{0.536}$	$0.165_{0.002}$	400	$20.092_{0.527}$
	WaveStitch-8	$0.598_{0.006}$	200	$5.279_{0.581}$	$\mathbf{0.537_{0.023}}$	200	$5.210_{0.556}$	$\mathbf{0.149_{0.002}}$	200	$5.330_{0.502}$
	WaveStitch-16	$0.597_{0.012}$	200	$\mathbf{5.129_{0.648}}$	$0.553_{0.040}$	200	$\mathbf{4.956_{0.432}}$	$0.150_{0.001}$	200	$\mathbf{5.184_{0.508}}$
	WaveStitch-32	$0.618_{0.009}$	200	$5.798_{0.524}$	$0.605_{0.029}$	200	$5.306_{0.889}$	$0.158_{0.002}$	200	$5.673_{0.536}$
	TimeGAN-32	$4.052_{0.003}$	1	$0.030_{0.057}$	$4.065_{0.015}$	1	$0.019_{0.037}$	$2.731_{0.020}$	1	$0.021_{0.040}$
PE	WaveStitchAR-8	$2.417_{0.020}$	320400	$3139.356_{10.071}$	$\mathbf{0.232_{0.016}}$	10800	$105.714_{0.831}$	$0.184_{0.003}$	320400	$3153.572_{4.575}$
	WaveStitchAR-16	$1.102_{0.027}$	160200	$1566.418_{6.249}$	$0.232_{0.038}$	6000	$58.881_{0.689}$	$0.171_{0.002}$	160200	$1557.470_{3.514}$
	WaveStitchAR-32	$1.071_{0.024}$	80000	$778.270_{2.683}$	$0.966_{0.074}$	3600	$34.829_{0.449}$	$0.175_{0.003}$	80000	$771.878_{3.205}$
	WaveStitch-1	$1.046_{0.016}$	2600	$149.726_{0.569}$	$0.718_{0.126}$	2600	$149.801_{0.651}$	$0.171_{0.003}$	2600	$149.827_{0.568}$
	WaveStitch-8	$\mathbf{1.006_{0.013}}$	400	$19.556_{0.609}$	$0.587_{0.128}$	400	$19.532_{0.555}$	$\mathbf{0.165_{0.003}}$	400	$19.485_{0.553}$
	WaveStitch-16	$1.017_{0.017}$	200	$9.409_{0.585}$	$0.606_{0.087}$	200	$9.376_{0.552}$	$0.165_{0.005}$	200	$9.439_{0.536}$
	WaveStitch-32	$1.073_{0.022}$	200	$\mathbf{6.392_{0.467}}$	$0.688_{0.053}$	200	$\mathbf{6.445_{0.534}}$	$0.183_{0.004}$	200	$\mathbf{6.471_{0.605}}$
	TimeGAN-32	$6.768_{0.007}$	1	$0.029_{0.054}$	$6.138_{0.102}$	1	$0.020_{0.038}$	$4.092_{0.009}$	1	$0.020_{0.037}$

both versions share the same model and sliding window setup, they differ in how they apply conditioning over time. In WaveStitchAR, each generated window is used as context for the next one, so early errors can propagate unchecked, degrading quality for the subsequent windows. In contrast, WaveStitch imposes conditions "softly" via the conditional loss, iteratively refining all windows together during denoising. This joint optimisation improves robustness to early mistakes, giving WaveStitch a slight edge over WaveStitchAR.

The parallel approach also has a much faster runtime. For example, in task *R* in the PE dataset, WaveStitch-16 (stride 16) achieves an **average speedup of 166.48** (i.e., $\frac{1566.418}{9.409}$) over its sequential counterpart WaveStitchAR-16. It also achieves a lower MSE and reduces the function calls from **160200** to just **200**. The runtime generally decreases with larger strides, resulting in fewer windows, thus resulting in fewer denoiser calls. However, for shorter sequences like AT, where nearly all windows fit within one mini-batch (1024), the stride length has little impact on runtime.

The effect of stitching becomes evident with WaveStitch-32, which follows a *divide-and-conquer* approach without overlaps (and hence no stitching). It typically performs worse than variants with smaller strides, where stitching allows more contextual information to guide generation. However,

excessive overlap can also propagate errors from earlier windows, so using smaller strides does not always yield better accuracy.

The speedup does not reach the theoretical limit of the mini-batch size b in practice (see Sec. 4), possibly due to the under-utilisation of hardware and other overheads. For example, consider the MT dataset, with a sequence length of 7949 timesteps (see Table 2). For window size 32 and stride 8, the number of windows is roughly $(7949 - 32)/8 \approx 990$. As this is less than the mini-batch size of 1024, the autoregressive model makes roughly[4] $990 \times 200 = 198000$ denoiser calls. This is lower than the worst-case with $1024 \times 200 = 204800$ calls, so we have a lower speedup.

When analysing the speed–quality tradeoff, diffusion models consistently have lower MSEs than TimeGAN despite slower speeds. This difference stems from the core design: while diffusion iteratively corrects errors at each step, GANs generate using a single step, which is unstable.

For WaveStitch-32 (generally the fastest variant), we see the average time *per denoiser call* ranges from ~0.026-0.032 seconds, which is close to TimeGAN (~0.019-0.030 seconds). Thus, the overhead of diffusion stems not from individual operations, but from the larger number of denoising steps. We can achieve further efficiency improvements by reducing the number of steps using techniques like model distillation [28, 37]. These methods can be applied post-hoc on top of our design to further improve sampling efficiency.

5.5 Ablation: Effect of Stitching (RQ3)

Table 7 evaluates the imputation accuracy (MSE) using five trials, of three conditioning strategies: (1) WaveStitch with RePaint-based conditioning [27] and enforced overlap-coherence; (2) WaveStitch with only self-guidance on observations without stitching; and (3) WaveStitch with self-guidance and stitching. Results show that using stitching achieves the best performance in nearly all cases, while relying only on self-guidance performs the worst. RePaint is competitive, but generally loses to ours. Overall, the stitch loss improves performance by up to **35.43%** compared to just self-guidance (AT, I task) and up to **28.36%** over RePaint (RS, B task). This result directly aligns with Proposition 3.1 (Sec. 3.5), which explains how using self-guidance and stitching increases the likelihood of generating realistic samples.

5.6 Choice of Stitch Loss Function (RQ4)

Table 8 compares the imputation accuracy, measured by MSE, of four stitch loss formulations for time series: Mean Absolute Error (MAE), Cosine similarity, Pearson correlation loss [4], and our MSE-based loss. These losses include distance-based objectives (MSE, MAE) and shape-based criteria (cosine similarity, Pearson correlation). Each one promotes alignment differently: Pearson loss promotes linear dependence between the overlaps along the time axis; cosine similarity pushes the waveform shapes to match; and MAE is more robust towards outliers than MSE.

We observe that distance-based losses (MSE, MAE) generally outperform the shape-based objectives. This is because a zero stitch loss for distance-based objectives directly forces values across the overlaps to match. In contrast, shape-similarity losses can become zero even when overlaps differ in absolute value. Although we adopt MSE for stitching due to its simplicity and ease of optimisation, these results indicate that using alternatives, such as MAE, is still viable for task-specific flexibility.

5.7 Random Imputation Tasks (RQ5)

By default, we evaluate WaveStitch on three imputation granularities: R (coarse-grained), I (intermediate), and B (fine-grained), as detailed in Sec. 5.1. However, this does not limit its generalisability

[4]The exact total is 198600 due to a minor implementation detail where additional timesteps are added for conditional context.

Table 7. Comparing conditioning methods on imputation accuracy (MSE).

	RePaint	Self-guidance	Self-guidance + Stitching
(***R***)	$0.167_{.012}$	$0.222_{.030}$	$\mathbf{0.152_{.006}}$
AT (***I***)	$0.278_{.046}$	$0.381_{.054}$	$\mathbf{0.246_{.009}}$
(***B***)	$0.153_{.007}$	$0.178_{.024}$	$\mathbf{0.140_{.004}}$
(***R***)	$\mathbf{0.506_{.009}}$	$0.544_{.010}$	$0.512_{.009}$
MT (***I***)	$0.439_{.037}$	$0.470_{.048}$	$\mathbf{0.396_{.019}}$
(***B***)	$0.140_{.011}$	$0.127_{.007}$	$\mathbf{0.111_{.008}}$
(***R***)	$1.525_{.017}$	$1.541_{.018}$	$\mathbf{1.481_{.019}}$
BQ (***I***)	$1.027_{.015}$	$1.050_{.017}$	$\mathbf{0.981_{.015}}$
(***B***)	$0.111_{.005}$	$0.102_{.010}$	$\mathbf{0.099_{.009}}$
(***R***)	$0.616_{.005}$	$0.615_{.005}$	$\mathbf{0.598_{.006}}$
RS (***I***)	$0.570_{.020}$	$0.561_{.022}$	$\mathbf{0.537_{.023}}$
(***B***)	$0.208_{.005}$	$0.168_{.003}$	$\mathbf{0.149_{.002}}$
(***R***)	$1.016_{.013}$	$1.063_{.013}$	$\mathbf{1.006_{.013}}$
PE (***I***)	$\mathbf{0.511_{.102}}$	$0.847_{.134}$	$0.587_{.128}$
(***B***)	$0.177_{.002}$	$0.178_{.003}$	$\mathbf{0.165_{.003}}$

Table 8. Comparison of stitch loss functions (MSE↓), for various datasets and tasks (R/I/B).

	MSE	MAE	Cosine	Correlation
(***R***)	$\mathbf{0.152_{.006}}$	$0.157_{.006}$	$0.182_{.021}$	$0.182_{.019}$
AT (***I***)	$0.246_{.009}$	$\mathbf{0.233_{.005}}$	$0.273_{.061}$	$0.274_{.062}$
(***B***)	$0.140_{.004}$	$0.128_{.005}$	$\mathbf{0.126_{.002}}$	$\mathbf{0.126_{.002}}$
(***R***)	$\mathbf{0.512_{.009}}$	$0.521_{.008}$	$0.535_{.010}$	$0.539_{.009}$
MT (***I***)	$0.396_{.019}$	$\mathbf{0.330_{.017}}$	$0.402_{.036}$	$0.413_{.038}$
(***B***)	$\mathbf{0.111_{.008}}$	$0.133_{.007}$	$0.135_{.006}$	$0.134_{.006}$
(***R***)	$\mathbf{1.481_{.019}}$	$1.499_{.019}$	$1.533_{.019}$	$1.537_{.018}$
BQ (***I***)	$\mathbf{0.981_{.015}}$	$0.994_{.014}$	$1.033_{.016}$	$1.041_{.016}$
(***B***)	$0.099_{.009}$	$0.095_{.004}$	$\mathbf{0.094_{.003}}$	$\mathbf{0.094_{.003}}$
(***R***)	$0.598_{.006}$	$\mathbf{0.593_{.005}}$	$0.608_{.005}$	$0.603_{.005}$
RS (***I***)	$0.537_{.023}$	$\mathbf{0.531_{.023}}$	$0.553_{.022}$	$0.549_{.024}$
(***B***)	$\mathbf{0.149_{.002}}$	$0.297_{.003}$	$0.327_{.006}$	$0.326_{.006}$
(***R***)	$\mathbf{1.006_{.013}}$	$1.013_{.012}$	$1.046_{.014}$	$1.055_{.013}$
PE (***I***)	$0.587_{.128}$	$\mathbf{0.363_{.092}}$	$0.755_{.129}$	$0.797_{.127}$
(***B***)	$0.165_{.003}$	$\mathbf{0.142_{.003}}$	$0.149_{.003}$	$0.148_{.003}$

to other scenarios. To show this, we also compare the accuracies (MSE) on random imputation scenarios. These tasks are constructed by randomly masking out a certain percentage of signal rows while keeping the metadata intact. Results are shown in Table 9. As expected, increasing the proportion of missing information generally degrades performance. On comparing these results to the *R* tasks (root level) in Table 3, we find that MSE scores remain consistently better than the *R* tasks, which have 100% missing signals (hardest scenario). This highlights that WaveStitch is not limited to specific missingness configurations and is generalisable to other scenarios.

Table 9. MSE for varying missingness ratios

Dataset	25%	50%	75%
MT	$0.092_{.003}$	$0.104_{.002}$	$0.137_{.003}$
BAQ	$0.146_{.004}$	$0.167_{.003}$	$0.295_{.005}$
RS	$0.233_{.007}$	$0.271_{.008}$	$0.351_{.008}$
PE	$0.004_{.000}$	$0.009_{.000}$	$0.030_{.001}$
AT	$0.141_{.003}$	$0.140_{.003}$	$0.138_{.005}$

5.8 Visual Assessment of Generated Data

Figure 7 compares WaveStitch against other baselines: TSDiff and TimeWeaver, on the BQ dataset. Figure 7a illustrates the autocorrelation for each task across 100 lags, along with the real signal.

We see that methods conditioning on observed signals (TSDiff and WaveStitch) better resemble ground truth signals under finer-grained tasks. In contrast, TimeWeaver does significantly worse on bottom-level tasks since it does not condition on the observed signals. Cross-feature correlations (see Figure 7b) further highlight this trend, with WaveStitch and TSDiff maintaining better inter-feature dependencies for fine-grained tasks. These results highlight WaveStitch's strength in capturing key temporal and structural patterns.

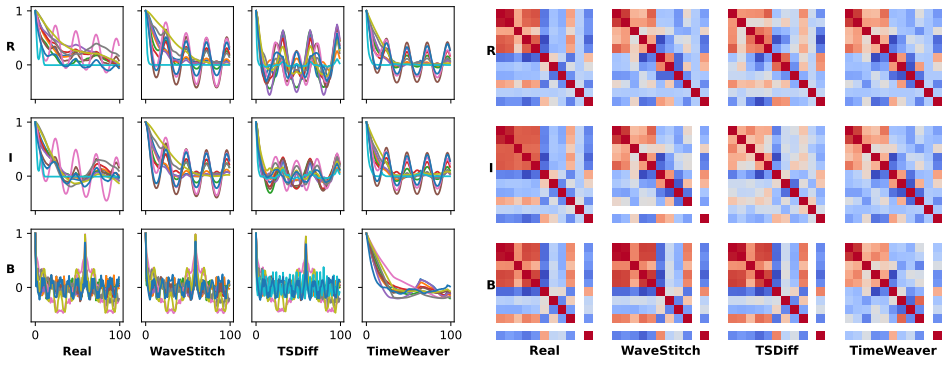
6 Related Work

Table 10 summarises the related works from time series generation and adjacent domains. These works vary in their backbone architectures (GANs versus diffusion); their ability to condition on both metadata and observations; and their conditioning strategies (training versus inference-time).

Time Series Models: Generative models have evolved in recent years. TimeGAN [47] pioneered GAN-based architectures for time series generation, but focuses on unconditional synthesis without leveraging metadata or the observed signals. In contrast, more recent methods employ conditioning to utilise additional information for guiding the generative process.

TimeWeaver [29] trains a conditional diffusion backbone by generating the entire signal just from the metadata. However, it ignores the observed signals, despite their criticality for fine-grained imputation tasks. In contrast, TimeGrad [34] ignores metadata but only handles forecasting tasks by conditioning on historical signals from preceding timesteps, which is unsuitable for imputation at arbitrary timesteps. Moreover, it maintains temporal coherence via autoregression, which is slow due to sequential generation. SSSD [2] includes the conditional masks as inputs during training. While this enables imputing values at arbitrary timesteps, it risks overfitting on the masking patterns seen during training. TSDiff [22] removes this risk by conditioning on the observed signal patterns directly at inference through self-guidance. However, self-guidance only conditions on the observations within a given time window, ignoring dependencies across overlaps. TimeAutoDiff [41] sidesteps this cross-window dependency problem by generating the entire output in one go using a latent transformer-based diffusion model [36] and classifier-free guidance [16] for conditioning on labels. However, this design restricts it to fixed-length sequences and is impractical: longer sequences significantly increase the computational load on the self-attention layers, hurting scalability [20]. Moreover, adapting classifier-free guidance from conditioning on discrete labels to conditioning on the continuous and arbitrarily observed signals is challenging.

Non-Time-Series Models: RePaint [27], designed for image inpainting, perturbs the observed pixels to blend their noise with that of the missing regions. Then, it jointly denoises both using an unconditional model directly at inference. However, independently noising the observed areas leads



(a) Per-channel Autocorrelation (b) Cross-feature Correlations

Fig. 7. Autocorrelation (up to 100 lags), and Cross-feature Correlations for BQ for *R*, *I*, and *B* tasks.

to noise misalignment with the missing regions, producing semantically inconsistent outputs [10, 27]. For video generation, NUWA-XL [46] uses a hierarchical "coarse-to-fine" diffusion approach using two models. It first generates coarse keyframes using prompts on an unconditional model, then fills the gaps using classifier-free guidance [16]using a conditional one. However, its conditional model specifically uses the first and last frames and cannot be conditioned on arbitrary frame positions. Moreover, the recursive refinement introduces sequential dependencies: each refinement stage depends on the previous one, leading to autoregressive-like behaviour.

In summary, many existing related works rely on minimal or rigid conditioning, ignore temporal dependencies, or are inefficient. Instead, WaveStitch offers a lightweight, inference-time solution that flexibly accommodates diverse conditioning signals and maintains temporal consistency without retraining or sacrificing parallelism.

Table 10. Summary of related works

Method	Time Series	Metadata	Observed Values	Cross-window Coherence	Inference-time Conditioning
TimeWeaver [29]	✔	✔	✖	✖	✖
TimeGAN [47]	✔	✖	✖	✖	✖
TSDiff [22]	✔	✖	✔	✖	✔
SSSD [2]	✔	✖	✔	✖	✖
TimeGrad [34]	✔	✖	✔	✔	✖
TimeAutoDiff [41]	✔	✔	✖	✖	✖
RePaint [27]	✖	✖	✔	✖	✔
NUWA-XL [46]	✖	✔	✔	✔	✖
WaveStitch (ours)	✔	✔	✔	✔	✔

7 Conclusion

We present WaveStitch, a novel framework for conditional time series synthesis. WaveStitch makes three key contributions: (1) integrating metadata and observed signals through a dual-sourced conditioning strategy; (2) hybridising training and inference via metadata-conditioned model training and gradient-based refinements on the observed signals using a conditional loss; and (3) coherent

pipelined parallel generation through overlap alignment via a stitching mechanism. Built on a diffusion backbone, WaveStitch integrates these innovations to generate high-quality, temporally consistent time series across various tasks. Empirical results demonstrate that WaveStitch handles complex conditioning patterns and outperforms SOTA baselines with a **1.81x** lower MSE on average, and a speedup of up to **166.48x** over autoregressive methods. While effective, WaveStitch currently treats metadata as static and fully observed. An important direction for future work is to extend the framework to handle partially observed metadata, which introduces additional challenges such as synthesising discrete and continuous-valued features. Other promising avenues include conditioning strategies for black-box settings, exploring alternative modalities for conditioning, and scaling WaveStitch to distributed synthesis settings. These advances would further enhance WaveStitch 's flexibility across tasks and enable conditional generation under data privacy and resource constraints.

Acknowledgements

This publication was supported by the Dutch Research Council (VI.Veni.222.439), the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe Research and Innovation programme (Grant Agreement No. 101192750), the Swiss National Science Foundation (Priv-GSyn project, 200021E_229204), and the DEPMAT project (P20-22 / N21022) of the research programme Perspectief which is partly financed by the Dutch Research Council (NWO).

References

[1] Ernesto Aguilar Madrid. 2021. Short-term electricity load forecasting (Panama case study). Mendeley Data, V1. DOI: https://doi.org/10.17632/byx7sztj59.1.

[2] Juan Lopez Alcaraz and Nils Strodthoff. 2023. Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models. *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=hHiIbk7ApW

[3] Yihao Ang, Qiang Huang, Yifan Bao, Anthony KH Tung, and Zhiyong Huang. 2023. TSGBench: Time Series Generation Benchmark. *Proc. VLDB Endow.* (2023).

[4] Michael R Berthold and Frank Höppner. 2016. On clustering time series using euclidean distance and pearson correlation. *arXiv preprint arXiv:1601.02213* (2016).

[5] Luis Blanche. 2020. Quarterly Tourism In Australia. https://www.kaggle.com/datasets/luisblanche/quarterly-tourism-in-australia. Kaggle.

[6] Song Chen. 2017. Beijing Multi-Site Air Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5RK5G.

[7] Yuhang Chen, Chaoyun Zhang, Minghua Ma, Yudong Liu, Ruomeng Ding, Bowen Li, Shilin He, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. ImDiffusion: Imputed Diffusion Models for Multivariate Time Series Anomaly Detection. *Proc. VLDB Endow.* 17, 3 (Nov. 2023), 359–372. doi:10.14778/3632093.3632101

[8] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi (Eds.). Association for Computational Linguistics, 103–111. doi:10.3115/V1/W14-4012

[9] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2201–2206. doi:10.1145/2882903.2912574

[10] Hyungjin Chung, Byeongsu Sim, Dohoon Ryu, and Jong Chul Ye. 2022. Improving diffusion models for inverse problems using manifold constraints. *Advances in Neural Information Processing Systems* 35 (2022), 25683–25696.

[11] Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems* 34 (2021), 8780–8794.

[12] Christos Faloutsos, Jan Gasthaus, Tim Januschowski, and Yuyang Wang. 2018. Forecasting big time series: old and new. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2102–2105.

[13] FlorianKnauer and Will Cukierski. 2015. Rossmann Store Sales. https://kaggle.com/competitions/rossmann-store-sales. Kaggle.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems* 33 (2020), 6840–6851.

[16] Jonathan Ho and Tim Salimans. 2021. Classifier-Free Diffusion Guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.

[17] John Hogue. 2019. Metro Interstate Traffic Volume. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5X60B.

[18] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. 2021. Time-series generation by contrastive imitation. *Advances in neural information processing systems* 34 (2021), 28968–28982.

[19] Jinwoo Jeon, Kangwook Lee, Sewoong Oh, Jungseul Ok, et al. 2021. Gradient inversion with generative image prior. *Advances in neural information processing systems* 34 (2021), 29898–29908.

[20] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. 2023. On the computational complexity of self-attention. In *International conference on algorithmic learning theory*. PMLR, 597–619.

[21] Diederik P Kingma, Max Welling, et al. 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning* 12, 4 (2019), 307–392.

[22] Marcel Kollovieh, Abdul Fatir Ansari, Michael Bohlke-Schneider, Jasper Zschiegner, Hao Wang, and Yuyang Bernie Wang. 2024. Predict, refine, synthesize: Self-guiding diffusion models for probabilistic time series forecasting. *Advances in Neural Information Processing Systems* 36 (2024).

[23] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. 2023. Tabddpm: Modelling tabular data with diffusion models. In *International Conference on Machine Learning*. PMLR, 17564–17579.

[24] Chengxuan Li, Di Huang, Zeyu Lu, Yang Xiao, Qingqi Pei, and Lei Bai. 2024. A survey on long video generation: Challenges, methods, and prospects. *arXiv preprint arXiv:2403.16407* (2024).

[25] Shujian Liao, Hao Ni, Marc Sabate-Vidales, Lukasz Szpruch, Magnus Wiese, and Baoren Xiao. 2024. Sig-Wasserstein GANs for conditional time series generation. *Mathematical Finance* 34, 2 (2024), 622–670.

[26] Tongyu Liu, Ju Fan, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Controllable Tabular Data Synthesis Using Diffusion Models. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–29.

[27] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. 2022. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11461–11471.

[28] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. 2023. On Distillation of Guided Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 14297–14306.

[29] Sai Shankar Narasimhan, Shubhankar Agarwal, Oguzhan Akcin, Sujay Sanghavi, and Sandeep P. Chinchali. 2024. Time Weaver: A Conditional Time Series Generation Model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Vol. 235. 37293–37320.

[30] Hao Ni, Lukasz Szpruch, Marc Sabate-Vidales, Baoren Xiao, Magnus Wiese, and Shujian Liao. 2021. Sig-Wasserstein GANs for time series generation. In *Proceedings of the Second ACM International Conference on AI in Finance*. 1–8.

[31] Kun Il Park, M Park, et al. 2018. *Fundamentals of probability and stochastic processes with applications to communications*. Springer.

[32] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1071–1083.

[33] Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S Jensen, Zhenli Sheng, et al. 2024. TFB: Towards Comprehensive and Fair Benchmarking of Time Series Forecasting Methods. *Proceedings of the VLDB Endowment* 17, 9 (2024), 2363–2377.

[34] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. 2021. Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8857–8868. https://proceedings.mlr.press/v139/rasul21a.html

[35] Pau Rodríguez, Miguel A Bautista, Jordi Gonzalez, and Sergio Escalera. 2018. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing* 75 (2018), 21–31.

[36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10684–10695.

[37] Tim Salimans and Jonathan Ho. 2022. Progressive Distillation for Fast Sampling of Diffusion Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=TIdIXIpzhoI

[38] Anupam Sanghi and Jayant R Haritsa. 2023. Synthetic Data Generation for Enterprise DBMS. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 3585–3588.

[39] Aditya Shankar, Hans Brouwer, Rihan Hai, and Lydia Chen. 2024. SiloFuse: Cross-silo Synthetic Data Generation with Latent Tabular Diffusion Models. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 110–123. doi:10.1109/ICDE60146.2024.00016

[40] Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. 2023. Simplified State Space Layers for Sequence Modeling. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=Ai8Hw3AXqks

[41] Namjoon Suh, Yuning Yang, Din-Yin Hsieh, Qitong Luan, Shirong Xu, Shixiang Zhu, and Guang Cheng. 2024. TimeAutoDiff: Combining Autoencoder and Diffusion model for time series tabular data synthesizing. *arXiv preprint arXiv:2406.16028* (2024).

[42] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems* 34 (2021), 24804–24816.

[43] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[44] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant GANs: deep generation of financial time series. *Quantitative Finance* 20, 9 (2020), 1419–1440.

[45] Tianlin Xu, Li Kevin Wenliang, Michael Munn, and Beatrice Acciaio. 2020. Cot-gan: Generating sequential data via causal optimal transport. *Advances in neural information processing systems* 33 (2020), 8798–8809.

[46] Shengming Yin, Chenfei Wu, Huan Yang, Jianfeng Wang, Xiaodong Wang, Minheng Ni, Zhengyuan Yang, Linjie Li, Shuguang Liu, Fan Yang, et al. 2023. NUWA-XL: Diffusion over Diffusion for eXtremely Long Video Generation. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

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

[48] Xinyu Yuan and Yan Qiao. 2024. Diffusion-TS: Interpretable Diffusion for General Time Series Generation. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=4h1apFjO99

Received April 2025; revised July 2025; accepted August 2025