RECFLOWTIME: SELF-CONDITIONED RECTIFIED FLOW FOR TIME-SERIES GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Flow matching offers an efficient alternative to diffusion models by learning continuous transport paths from noise to data that can be integrated in relatively few steps. We introduce RecFlowTime, a rectified-flow framework for time-series generation that learns velocities along linear data–noise interpolations. A bidirectional Transformer with rotary position embeddings represents temporal relationships, while self-conditioning progressively refines the estimated clean sequence. During training, minibatch optimal-transport coupling replaces independent pairing, reducing path crossings and producing straighter sampling trajectories. A floored minimum signal-to-noise ratio (min-SNR) weight preserves supervision near the low-noise endpoint and reduces sample roughness. Across four synthetic, financial, sensor, and biomedical benchmarks, RecFlowTime outperforms recent baselines on key measures of distributional fidelity and temporal realism. In addition, RecFlowTime uses only 20 function evaluations, compared with 200–500 for the diffusion baselines, resulting in more than an order-of-magnitude faster sampling and up to a 79% reduction in training time. Although trained unconditionally, RecFlowTime also supports conditional generation for imputation and forecasting without retraining or additional model evaluations.

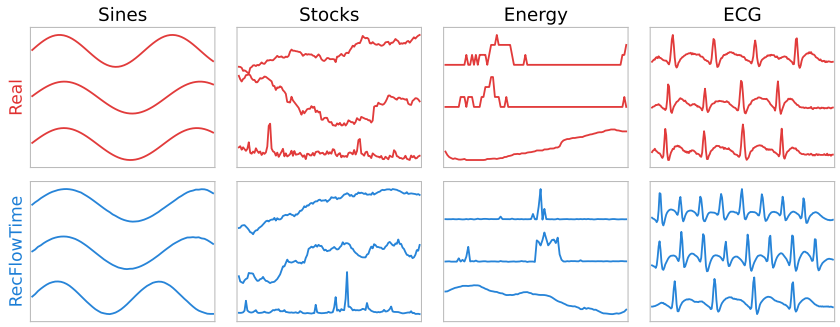

Figure 1: Real sequences (red) and sequences generated by RecFlowTime (blue) on the four benchmarks. Additional samples and comparisons across all methods are provided in Appendix H.

1 INTRODUCTION

Time-series generation aims to produce synthetic sequences that preserve the statistical and temporal properties of real data (Yoon et al., 2019). This capability is valuable when observations are scarce, sensitive, or expensive to collect. In healthcare (Esteban et al., 2017) and finance, for example, synthetic data can support data sharing and model development while limiting direct exposure of confidential records. Synthetic sequences can also augment small training sets and improve the representation of uncommon patterns (Wen et al., 2021). However, realistic generation requires more than matching the distribution of individual values: a model must also preserve local changes, long-range dependencies, periodic behavior, and relationships among features (Jeha et al., 2022; Liao et al., 2024). Existing methods commonly address this challenge by introducing explicit temporal priors (Yuan & Qiao, 2024; Alaa et al., 2021). Some use seasonal–trend decompositions (Yuan

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

& Qiao, 2024) or Fourier-domain reconstruction losses, whereas others employ frequency-aware layers (Cai et al., 2026). Signature-based methods (Liao et al., 2024; Lu & Sester, 2024) provide another form of temporal bias by representing the ordered increments of a path through iterated integrals (Chen, 1957; Lyons, 1998). Thus, the prevailing approach promotes temporal fidelity through specialized architectures or training objectives.

We take a complementary approach by formulating the generative process to improve sample fidelity without explicit temporal priors while also reducing training and sampling costs. Accordingly, we introduce RecFlowTime, a rectified-flow framework (Liu et al., 2023) that learns a velocity field along linear interpolations between data and noise and generates samples by integrating this field from noise to data. To adapt this formulation to time-series data, a bidirectional Transformer parameterizes the velocity field, rotary position embeddings represent relative temporal offsets, and self-conditioning allows successive sampling steps to refine the estimated clean sequence. During training, minibatch optimal-transport coupling replaces independent data–noise pairing (Tong et al., 2024; Pooladian et al., 2023). This coupling reduces path crossings and produces a straighter marginal velocity field, enabling accurate integration with few steps. Finally, a floored loss weight maintains supervision near the low-noise endpoint and reduces excessive sample roughness. Together, these components improve temporal modeling through transport geometry, sequence-aware parameterization, and targeted supervision. Figure 1 compares real sequences with samples generated by RecFlowTime across four benchmark datasets. We review related work on adversarial, variational, diffusion-based, and flow-based time-series generation in Appendix A.

The main contributions of this work are threefold:

1. **Rectified flow for time-series generation.** We introduce a rectified-flow framework that learns velocities along linear paths between data and noise and generates samples using 20 Euler steps. A bidirectional Transformer with rotary position embeddings represents the velocity field, while self-conditioning progressively refines the clean-sequence estimate.
2. **Transport-coupled pairing.** We replace independent data–noise pairing with minibatch optimal transport. This procedure changes only the pairing, leaving the data and noise distributions unchanged, while reducing path crossings and producing straighter sampling trajectories.
3. **Low-noise supervision.** We introduce a lower bound on the minimum signal-to-noise ratio (min-SNR) loss weight to maintain supervision near the low-noise endpoint and reduce excessive sample roughness.

2 Background

Let $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ be a dataset of multivariate time series drawn from an unknown distribution p, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^{T\times F}$ has T time steps and F features, and let $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ denote Gaussian noise of the same shape. We seek a generative distribution p_θ that approximates p and preserves the statistical and temporal properties of the observed sequences.

Rectified flow. Rectified flow (Liu et al., 2023) connects a data sample $\boldsymbol{x}$ to a noise sample $\boldsymbol{\epsilon}$ through the linear interpolation

$$\boldsymbol{x}_t = (1-t)\,\boldsymbol{x} + t\,\boldsymbol{\epsilon}, \qquad t \in [0,1], \tag{1}$$

which equals the data at $t = 0$ and noise at $t = 1$, and whose derivative is the constant velocity $\boldsymbol{\epsilon} - \boldsymbol{x}$. A network $v_\theta(\boldsymbol{x}_t, t)$ is trained to predict that velocity from an intermediate point and its time,

$$\mathcal{L}_{\text{flow}}(\theta) = \mathbb{E}_{\substack{(\boldsymbol{x},\boldsymbol{\epsilon})\sim\pi \\ t\sim\mathcal{U}[0,1]}} \left[w(t)\, \|v_\theta(\boldsymbol{x}_t, t) - (\boldsymbol{\epsilon} - \boldsymbol{x})\|_2^2 \right], \tag{2}$$

after which generation starts from Gaussian noise at $t = 1$ and integrates $d\boldsymbol{x}_t/dt = v_\theta(\boldsymbol{x}_t, t)$ back to $t = 0$.

Two terms in Equation 2 are ordinarily left at their defaults. The coupling π determines which noise sample each data sample is paired with, and is independent in the standard recipe; the weight $w(t)$ determines how much each noise level contributes to the objective. They are where RecFlowTime differs.

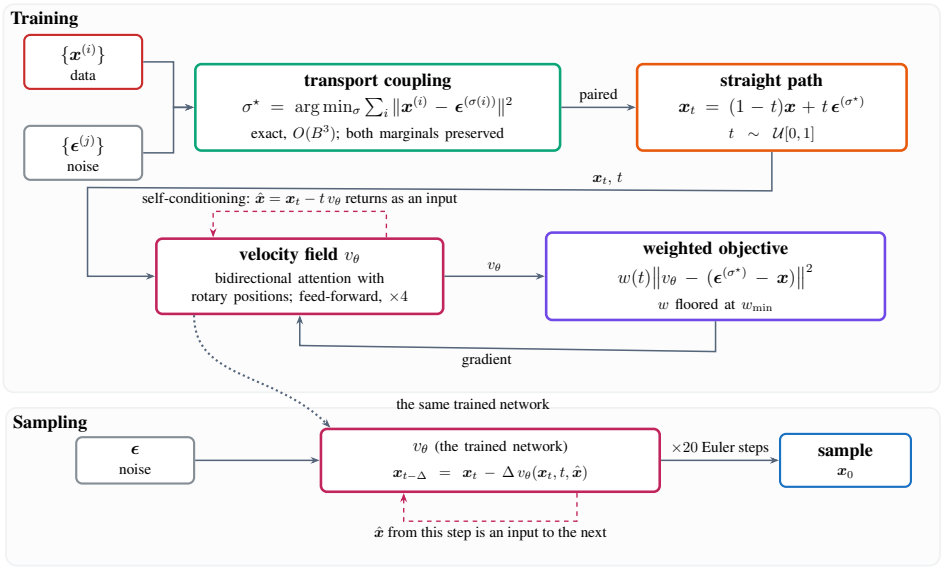

Figure 2: Overview of RecFlowTime training and sampling. During training, minibatch optimal transport decides which noise sequence each data sequence is paired with. Each pair defines a straight path between them, and the velocity along that path is the training target. The network, a self-conditioned Transformer with rotary positions, sees a random point on the path and predicts that velocity. Its squared error is weighted by noise level, with a floor that keeps the low-noise end supervised. During sampling, the same trained field turns noise into a sequence in 20 Euler steps. Each clean-sequence estimate is reused as an input at the next step.

3 Proposed Method: RecFlowTime

RecFlowTime retains the linear interpolation in Equation 1 and the velocity-regression objective in Equation 2, while modifying the pairing of data and noise, the parameterization of the velocity field, and the weighting of noise levels. During training, transport coupling first selects the data–noise pairs that define the interpolation paths. A Transformer then predicts the velocity at a randomly sampled point on each path, and the floored loss weight determines the contribution of that point to the objective. During generation, the trained velocity field transports Gaussian noise to data through 20 Euler steps. Figure 2 summarizes this pipeline. We first describe the velocity network and clean-sequence recovery, followed by the coupling, loss weighting, and sampling procedure. These components do not impose a domain-specific temporal prior: the architecture contains no seasonal–trend decomposition, and the objective includes no frequency-domain term.

3.1 Velocity-Field Network

The network v_θ maps an intermediate sequence $\boldsymbol{x}_t \in \mathbb{R}^{T\times F}$ and its flow time t to a velocity prediction of the same shape. Let $s \in \{1,\dots,T\}$ index sequence positions, and let $\boldsymbol{c} \in \mathbb{R}^{T\times F}$ denote the optional self-conditioning input. The initial representation of each token is

$$\boldsymbol{h}_s^{(0)} = W_x \boldsymbol{x}_{t,s} + W_c \boldsymbol{c}_s + \boldsymbol{e}(t), \tag{3}$$

where W_x and W_c are learned projections and $\boldsymbol{e}(t)$ is obtained from sinusoidal features followed by a two-layer multilayer perceptron. When self-conditioning is unavailable, $\boldsymbol{c}$ is set to zero.

The token representations are processed by a bidirectional Transformer (Vaswani et al., 2017) with rotary position embeddings (Su et al., 2024). The velocity prediction at each position is

$$v_\theta(\boldsymbol{x}_t, t, \boldsymbol{c})_s = W_o\ \mathrm{LN}\Big(\boldsymbol{h}_s^{(L)}\Big), \tag{4}$$

where $\boldsymbol{h}_s^{(L)}$ is the output of the final Transformer block and W_o projects it back to F features. Rotary embeddings allow the attention scores to represent relative temporal offsets. The following subsection derives the clean-sequence estimate used as $\boldsymbol{c}$ during self-conditioning.

3.2 Clean-Sequence Recovery and Self-Conditioning

The velocity prediction can be converted directly into an estimate of the clean sequence. Along the linear path in Equation 1, the target velocity is $\boldsymbol{\epsilon} - \boldsymbol{x}$, and therefore $\boldsymbol{x}_t - t(\boldsymbol{\epsilon} - \boldsymbol{x}) = \boldsymbol{x}$. Replacing the target velocity with the network prediction gives

$$\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}) = \boldsymbol{x}_t - t\, v_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}). \tag{5}$$

Equation 5 provides the self-conditioning signal introduced in the preceding subsection. During training, self-conditioning is applied with probability 0.5. A gradient-free forward pass with $\boldsymbol{c} = \mathbf{0}$ first produces a preliminary clean-sequence estimate, which is then used as $\boldsymbol{c}$ in the main forward pass. Otherwise, $\boldsymbol{c}$ remains zero. During sampling, $\boldsymbol{c}$ is initialized to zero and subsequently replaced by the clean-sequence estimate from the preceding Euler step. Thus, self-conditioning requires an additional pass during some training updates but no additional function evaluations during sampling. The same estimate is also constrained by masked replacement during conditional imputation and forecasting.

Proposition 1 (Bounded recovery)**.** *For fixed $\boldsymbol{x}_t$, t, and $\boldsymbol{c}$, the Jacobian of the clean-sequence estimate in Equation 5 with respect to the network output is $-t\boldsymbol{I}$. Its operator norm is therefore $t \leq 1$ for every $t \in [0, 1]$. In contrast, a noise-prediction diffusion model recovers the clean sequence as*

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_\tau - \sqrt{1 - \bar{\alpha}_\tau}\, \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau)}{\sqrt{\bar{\alpha}_\tau}}.$$

The corresponding Jacobian with respect to $\boldsymbol{\epsilon}_\theta$ is

$$-\sqrt{\frac{1 - \bar{\alpha}_\tau}{\bar{\alpha}_\tau}}\, \boldsymbol{I},$$

whose norm diverges as $\bar{\alpha}_\tau \to 0$.

Appendix C provides the proof. The proposition shows that the conversion from a velocity prediction to a clean-sequence estimate does not amplify prediction errors: a perturbation δv changes the estimate by $-t\,\delta v$. In noise-prediction diffusion, the corresponding error is scaled by $\sqrt{(1 - \bar{\alpha}_\tau)/\bar{\alpha}_\tau}$, which can become large near the high-noise endpoint. Direct clean-sample prediction, as used by Diffusion-TS (Yuan & Qiao, 2024), also avoids this division; however, it does not provide the linear transport paths used by RecFlowTime for few-step sampling.

This completes the definition of the self-conditioning signal. We next describe how data and noise samples are paired to construct the training paths.

3.3 Transport Coupling

Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^{B}$ be a data minibatch and $\{\boldsymbol{\epsilon}^{(j)}\}_{j=1}^{B}$ an independently sampled noise minibatch. Instead of retaining the arbitrary index-wise pairing, we find the minimum-cost one-to-one assignment

$$\sigma^\star = \arg\min_{\sigma \in \mathfrak{S}_B} \sum_{i=1}^{B} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\epsilon}^{(\sigma(i))} \right\|_2^2, \tag{6}$$

where $\mathfrak{S}_B$ is the set of permutations of B elements. Training then uses the matched pairs $(\boldsymbol{x}^{(i)}, \boldsymbol{\epsilon}^{(\sigma^\star(i))})$.

This assignment reduces the total distance between paired endpoints and tends to reduce intersections among their straight conditional paths. When many paths intersect, their velocity targets can point in conflicting directions. The network must average these targets, which can produce a curved marginal velocity field. By reducing such conflicts, transport coupling is intended to produce trajectories that can be integrated more accurately with few steps. Appendix G.3 evaluates this mechanism directly using sampling-trajectory curvature.

Equation 6 is a linear assignment problem. After constructing the pairwise cost matrix in $O(B^2TF)$ operations, we solve the assignment exactly using the Hungarian algorithm in $O(B^3)$ time (Kuhn, 1955). This cost is incurred only during training and does not add any model evaluations during sampling. Appendix B reports its measured contribution to training time.

The construction has two important properties. First, it changes only the pairing: the data minibatch remains unchanged, and the noise minibatch is only permuted. The empirical data and noise marginals are therefore preserved exactly. Second, the assignment is computed within each minibatch and only approximates a coupling between the full distributions. This approximation becomes less accurate as the batch size decreases (Pooladian et al., 2023). We use $B = 128$ in all experiments.

3.4 Low-Noise Supervision

Transport coupling determines the endpoints of each training path, while the weight $w(t)$ in Equation 2 determines how strongly each position along that path contributes to the objective. We use a floored minimum signal-to-noise ratio (min-SNR) weight:

$$w(t) = \max\left(\frac{\min(\mathrm{SNR}(t), \gamma)}{\mathrm{SNR}(t)}, w_{\min}\right), \qquad \mathrm{SNR}(t) = \left(\frac{1-t}{t}\right)^2, \qquad t \in (0,1). \tag{7}$$

We define the endpoint values by continuity as $w(0) = w_{\min}$ and $w(1) = 1$, and use Equation 7 with $\gamma = 5$ and $w_{\min} = 0.1$. Min-SNR weighting balances supervision across noise levels by reducing the contribution of high-SNR examples. For the linear path used here, however, $\mathrm{SNR}(t) \to \infty$ as $t \to 0$, causing the original min-SNR weight to approach zero. For example, its value is approximately 5.1×10^{-4} at $t = 0.01$.

This behavior provides almost no supervision near the data endpoint. Because reverse-time sampling terminates at $t = 0$, prediction errors in this region directly affect fine-scale temporal variation in the generated sequence. The floor maintains $w(t) \geq w_{\min}$ while preserving min-SNR weighting elsewhere, preventing the training signal from vanishing as $t \to 0$. On Sines, it reduces normalized sample roughness from 2.60 to 1.25 and also improves Context-FID and spectral distance. Appendix G.2 provides the complete analysis.

3.5 Sampling

With the training paths, network parameterization, and loss weighting defined, generation uses only the trained velocity field. Sampling begins from $\boldsymbol{x}_1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ with the self-conditioning input initialized as $\boldsymbol{c}_1 = \mathbf{0}$. At each of the 20 uniform Euler steps, we compute

$$\boldsymbol{v}_t = v_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}_t), \qquad \boldsymbol{x}_{t-\Delta} = \boldsymbol{x}_t - \Delta\, \boldsymbol{v}_t, \qquad \boldsymbol{c}_{t-\Delta} = \hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}_t). \tag{8}$$

The clean-sequence estimate produced at each step is therefore reused as the self-conditioning input at the next step without requiring another network evaluation. Transport coupling and loss weighting are used only during training and add no sampling cost. Appendix B provides the architecture and training procedure, while Appendix G.5 reports training convergence and optimization stability.

4 Experiments

Code and data. A reference implementation is available at `https://anonymous.4open.science/r/RecFlowTime-ICLR27`. It is written in Python 3.13.13 and PyTorch 2.12.0 and it runs on CPU, CUDA or Apple Silicon. The repository contains the model, the training loop, every metric reported here, all four datasets and tutorial Jupyter notebooks.

4.1 Setup

Datasets. We evaluate RecFlowTime and the baselines on four benchmarks: Sines ($T = 64$, 4 features), Energy ($T = 96$, 5 features), Stocks ($T = 128$, 6 features), and ECG ($T = 192$, 2 features). Appendix D provides the dataset sources, construction procedures, preprocessing steps, and train–test splits.

Metrics. We report four metrics commonly used in the time-series generation literature: Context-FID (Jeha et al., 2022), computed from TS2Vec representations (Yue et al., 2022); the discriminative and predictive scores of Yoon et al. (2019); and the correlational score. Because these metrics do not fully characterize temporal behavior, we add three deterministic distances. The *spectral*

Table 1: Unconditional generation results on four benchmarks. We retrained all methods on identical data splits and evaluated them using the same implementation (Appendix F.2 and F.3). Learned metrics report the mean ± standard deviation across two evaluator seeds; entries shown as ± 0.000 have a nonzero spread below 0.0005. The correlational score is deterministic for fixed samples. Lower values are better, and boldface indicates the best result in each column.

Metric	Method	Sines	Stocks	Energy	ECG
Discriminative Score (Lower is better)	TimeGAN	0.419±0.016	0.164±0.055	0.228±0.142	0.173±0.002
	TimeVAE	0.023±0.018	0.178±0.075	0.071±0.011	0.338±0.069
	Diffusion-TS	0.030±0.008	0.141±0.009	0.249±0.009	0.198±0.005
	PaCoDi	0.362±0.087	0.075±0.040	0.248±0.015	0.172±0.034
	RecFlowTime	**0.009±0.006**	**0.026±0.009**	**0.013±0.004**	**0.114±0.027**
Predictive Score (Lower is better)	TimeGAN	0.017±0.000	0.016±0.000	0.074±0.007	0.066±0.003
	TimeVAE	0.010±0.000	0.014±0.000	0.033±0.000	0.064±0.001
	Diffusion-TS	0.010±0.000	0.013±0.001	0.035±0.000	0.060±0.001
	PaCoDi	0.010±0.000	0.012±0.000	0.034±0.000	0.059±0.000
	RecFlowTime	**0.010±0.000**	**0.012±0.000**	**0.030±0.000**	**0.057±0.000**
Context-FID Score (Lower is better)	TimeGAN	5.634±0.135	6.755±0.441	30.001±0.583	20.277±0.151
	TimeVAE	0.442±0.008	12.301±2.726	8.288±0.478	11.789±0.671
	Diffusion-TS	0.361±0.014	2.572±0.099	2.991±0.111	6.503±0.146
	PaCoDi	0.774±0.113	1.682±0.234	0.911±0.134	4.560±0.374
	RecFlowTime	**0.322±0.009**	**1.577±0.060**	**0.830±0.039**	**2.122±0.130**
Correlational Score (Lower is better)	TimeGAN	1.623±0.000	0.845±0.000	15.432±0.000	0.352±0.000
	TimeVAE	0.232±0.000	1.401±0.000	1.282±0.000	0.334±0.000
	Diffusion-TS	0.212±0.000	0.311±0.000	0.618±0.000	0.184±0.000
	PaCoDi	0.303±0.000	**0.225±0.000**	**0.338±0.000**	0.238±0.000
	RecFlowTime	**0.132±0.000**	0.318±0.000	0.416±0.000	**0.173±0.000**

distance compares the average log power spectra of real and generated data, the *increment distance* computes the 1-Wasserstein distance between their distributions of step-to-step changes, and the *autocorrelation distance* compares their average autocorrelation functions. Lower values indicate better performance for every metric. Appendix E provides complete metric definitions and evaluator configurations.

Baselines. We compare RecFlowTime with four time-series generators spanning three model families: the adversarial TimeGAN (Yoon et al., 2019), the variational TimeVAE (Desai et al., 2021), the time-domain diffusion model Diffusion-TS (Yuan & Qiao, 2024), and the frequency-domain diffusion model PaCoDi (Cai et al., 2026). We retrain every baseline on the same data splits used for RecFlowTime and evaluate all methods with the same code. Appendix F reports the hyperparameters, architecture dimensions, and parameter counts for each method.

4.2 Quantitative Results

Table 1 compares the methods using four measures of distributional, predictive, and cross-feature fidelity. RecFlowTime achieves the lowest discriminative score and Context-FID on every dataset. Relative to the strongest baseline, it reduces the discriminative score by 61%, 65%, 82%, and 34% on Sines, Stocks, Energy, and ECG, respectively. The corresponding Context-FID reductions are 11%, 6%, 9%, and 53%.

RecFlowTime also obtains the lowest or tied predictive score on all four datasets. For the correlational score, it leads on Sines and ECG, improving over the strongest baseline by 38% and 6%, respectively, while PaCoDi performs best on Stocks and Energy. Overall, RecFlowTime provides the strongest performance across the learned metrics.

4.3 Temporal Fidelity

Table 2 reports the three deterministic measures of temporal fidelity introduced above. Because these distances are deterministic functions of the evaluated samples, they do not require error bars over evaluator seeds. RecFlowTime achieves the lowest increment distance on all four datasets. Relative to the strongest baseline on each dataset, it reduces this distance by 8% on Sines, 15% on Stocks, 47% on Energy, and 29% on ECG. It also ties TimeVAE for the lowest autocorrelation distance on

Table 2: Temporal-fidelity metrics on four benchmarks. All metrics are deterministic for fixed generated samples, and lower values indicate better agreement with the real data. Boldface denotes the best result for each dataset and metric.

Metric	Method	Sines	Stocks	Energy	ECG
Increment distance (Lower is better)	TimeGAN	0.0127	0.0078	0.0263	0.0199
	TimeVAE	0.0012	0.0113	0.0250	0.0216
	Diffusion-TS	0.0018	0.0069	0.0180	0.0233
	PaCoDi	0.0037	0.0046	0.0210	0.0194
	RecFlowTime	**0.0011**	**0.0039**	**0.0095**	**0.0137**
Autocorrelation distance (Lower is better)	TimeGAN	0.0218	0.4400	0.3073	0.0724
	TimeVAE	0.0041	0.1274	0.2392	0.0334
	Diffusion-TS	0.0112	0.0936	0.0749	0.0105
	PaCoDi	0.0075	0.0583	0.0391	0.0137
	RecFlowTime	**0.0041**	**0.0262**	**0.0276**	**0.0068**
Spectral distance (Lower is better)	TimeGAN	0.2727	1.1775	3.3183	1.4042
	TimeVAE	0.1130	2.0722	2.8760	1.8034
	Diffusion-TS	**0.0472**	1.2261	1.4527	0.9191
	PaCoDi	0.0675	**0.1309**	0.2248	**0.2197**
	RecFlowTime	0.1095	0.3703	**0.2191**	0.5909

Sines and improves over the strongest baseline by 55%, 29%, and 35% on Stocks, Energy, and ECG, respectively. Spectral performance is more dataset dependent. Diffusion-TS leads on Sines, PaCoDi on Stocks and ECG, and RecFlowTime on Energy, where it improves over PaCoDi by 2.5%. Thus, RecFlowTime most consistently preserves local changes and lag-dependent structure. Appendix H provides the corresponding visual comparisons.

4.4 Ablation

To isolate the contribution of each component, we retrain six variants of RecFlowTime: **A1** uses independent pairing instead of transport coupling; **A2** replaces rotary embeddings with fixed sinusoidal encodings; **A3** removes self-conditioning; **A4** removes min-SNR weighting, including its loss floor; **A5** retains min-SNR weighting but removes the minimum loss floor; and **A6** replaces rectified flow with ϵ-prediction diffusion. A6 uses a cosine schedule and DDIM sampling with the same number of steps while retaining the remaining components.

Table 3 shows that the rectified-flow core has the largest effect on Context-FID. Replacing it with diffusion increases mean Context-FID by 113%. Removing self-conditioning produces the second-largest increase (36%), followed by min-SNR weighting (21%), the min-SNR floor (15%), transport coupling (12%), and rotary embeddings (8%). The complete model achieves the lowest Context-FID on Sines, Stocks, and Energy. The discriminative score gives a different ordering. Replacing rectified flow with diffusion causes the largest mean degradation ($1{,}183\%$), followed by removing rotary embeddings (143%), min-SNR weighting (99%), transport coupling (96%), self-conditioning (75%), and the min-SNR floor (2%). Overall, the components serve complementary roles, and no component improves every metric on every dataset. Appendix G.1 provides the complete analysis using four additional metrics.

4.5 Qualitative Evaluation

We complement the quantitative results with visual comparisons of global distributional coverage and marginal fidelity. Figure 3 presents these two aspects for RecFlowTime and PaCoDi. The PCA places each sequence as a single point, so the generated cloud should cover the same region as the real one. The kernel density estimate pools every value of every sequence into one distribution, thereby evaluating marginal fidelity.

RecFlowTime tracks the real marginal distribution of values on all four datasets, while PaCoDi under-weights the lower mode on Sines and Energy. The PCA projections show a different pattern: RecFlowTime covers the real region more closely on ECG, PaCoDi does so on Energy, and the two are comparable on Sines and Stocks. Neither model dominates across both views, and RecFlowTime is competitive or better in most of them. Appendix H provides additional qualitative analyses for all methods.

Table 3: Full leave-one-out ablation on the learned evaluation metrics. Values are mean ± standard deviation over two evaluator seeds. For each metric, ablations are ranked by their mean relative degradation across the four datasets; rank 1 indicates the largest degradation and therefore the greatest measured contribution. Lower metric values are better. Temporal-fidelity metrics are reported in Table 10.

Metric	Rank	Configuration	Sines	Stocks	Energy	ECG
Context-FID (Lower is better)	–	RecFlowTime	**0.322±0.009**	**1.577±0.060**	**0.830±0.039**	2.122±0.130
	5	– transport coupling	0.327±0.030	1.925±0.098	0.845±0.025	2.589±0.137
	6	– rotary embeddings	0.332±0.002	2.063±0.338	0.938±0.057	**1.849±0.003**
	2	– self-conditioning	0.340±0.026	1.960±0.142	0.835±0.012	4.555±0.274
	3	– min-SNR weighting	0.334±0.007	1.948±0.337	1.020±0.081	2.841±0.020
	4	– min-SNR floor only	0.345±0.006	1.978±0.301	0.834±0.004	2.662±0.173
	1	rectified flow → diffusion	0.521±0.005	5.984±0.109	1.620±0.038	2.446±0.086
Discriminative score (Lower is better)	–	RecFlowTime	**0.009±0.006**	0.026±0.009	**0.013±0.004**	0.114±0.027
	4	– transport coupling	0.019±0.006	0.011±0.006	0.051±0.034	0.142±0.052
	2	– rotary embeddings	0.017±0.005	0.037±0.003	0.076±0.071	**0.034±0.009**
	5	– self-conditioning	0.017±0.000	0.020±0.014	0.032±0.024	0.188±0.041
	3	– min-SNR weighting	0.028±0.008	0.023±0.011	0.043±0.026	0.062±0.003
	6	– min-SNR floor only	0.010±0.006	**0.011±0.006**	0.022±0.001	0.084±0.013
	1	rectified flow → diffusion	0.255±0.000	0.198±0.026	0.170±0.010	0.111±0.052

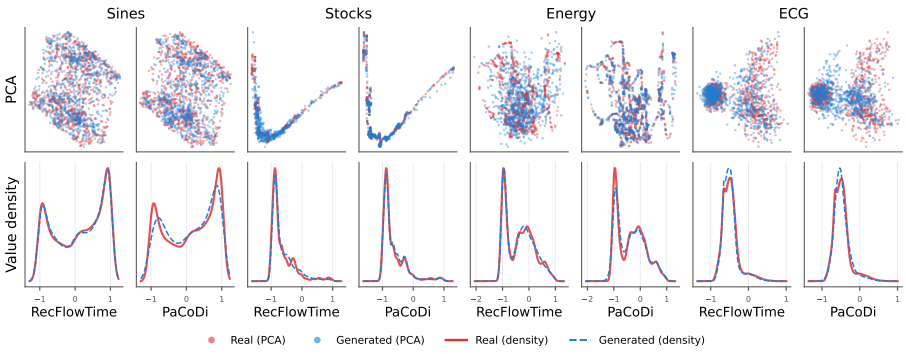

Figure 3: RecFlowTime compared with PaCoDi, the best-performing baseline. The upper two rows project real and generated sequences onto the first two principal components. The lower two rows give a kernel density estimate of the pooled values.

4.6 Sampling and Training Cost

Table 4 reports wall-clock cost on an Apple M2 (16 GB unified memory, PyTorch 2.12.0, MPS backend). Each method was trained using method-specific hyperparameters and a training budget selected to achieve stable performance without unnecessary additional optimization; complete configurations are reported in Appendix F.

At 20 function evaluations per sequence, RecFlowTime samples 29–176× faster than Diffusion-TS and 11–25× faster than PaCoDi, and trains 1.1–4.8× faster than both. The minibatch assignment adds no network evaluations. The Hungarian solve depends only on the batch size, while construction of the pairwise cost matrix scales as $O(B^2TF)$. The observed sampling-time difference is therefore primarily attributable to the number of integration steps. Appendix G.4 reports sample fidelity against step count. TimeGAN and TimeVAE sample in a single forward pass and are faster still, at substantially lower fidelity (Tables 1 and 2).

4.7 Conditional Generation

Beyond unconditional generation, RecFlowTime supports imputation and forecasting through masked replacement, without retraining or a separate conditional model. Let m denote a binary mask over the observed entries of a target sequence $\boldsymbol{y}$. After each Euler step, we apply

$$\boldsymbol{x}_t \leftarrow m \odot \big[(1-t)\boldsymbol{y} + t\boldsymbol{\epsilon}\big] + (1-m) \odot \boldsymbol{x}_t,$$

Table 4: Training and sampling costs measured on the same machine: an Apple M2 with 16 GB of unified memory. NFE denotes the number of function evaluations required to generate one sequence. Training time is reported in minutes, and sampling time is reported in milliseconds per sequence.

		Sines		Stocks		Energy		ECG	
Method	NFE	Train	Sample	Train	Sample	Train	Sample	Train	Sample
RecFlowTime	20	14.8	3.7	40.3	10.9	29.8	6.9	87.0	33.3
PaCoDi	200	24.4	50.7	112.0	228.9	45.0	169.8	92.0	350.1
Diffusion-TS	500	71.7	449.5	64.7	634.4	120.3	1215.4	186.9	980.0
TimeVAE	1	23.4	0.35	22.4	0.27	75.4	0.14	39.4	0.48
TimeGAN	1	84.1	0.08	144.4	0.19	129.0	0.10	219.9	0.28

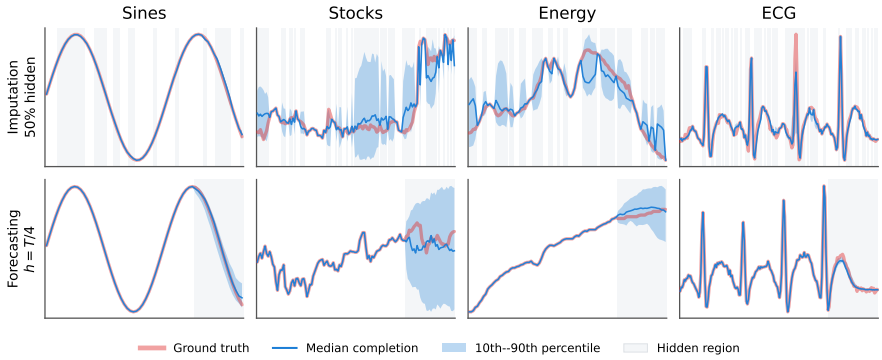

Figure 4: Conditional generation by masked replacement, at 20 Euler steps. Shaded spans are hidden; one feature is shown per dataset. The band is the 10th to 90th percentile over sixteen completions of the same input and the line is their median.

where the observed entries follow the same linear data–noise path used during training, while the unobserved entries remain generated by the model. For imputation, the zero entries of m mark hidden internal intervals; for forecasting, they mark the hidden sequence suffix. At $t = 0$, the update becomes $\boldsymbol{x}_0 \leftarrow m \odot \boldsymbol{y} + (1 - m) \odot \boldsymbol{x}_0$, so every observed entry is reproduced exactly while the generated values are retained at unobserved entries. Because this operation requires no additional model evaluations, both tasks retain the 20-evaluation sampling budget.

Figure 4 shows representative results for imputation with 50% hidden entries and forecasting over the final $T/4$ time steps. For imputation, the median completion closely follows the ground truth on Sines and ECG, including the periodic cycles and ECG peaks. The Stocks and Energy completions capture the broader trajectory but exhibit wider uncertainty within the hidden intervals. For forecasting, RecFlowTime preserves the periodic behavior of Sines and the recurring ECG morphology. For Stocks and Energy, however, the predictive interval gradually widens, indicating greater uncertainty. Appendix I provides the complete protocol and results for the conditional-generation tasks.

5 Conclusion

We introduced RecFlowTime, a rectified-flow framework for time-series generation. Linear data–noise paths enable clean-sequence recovery with bounded sensitivity to velocity errors. A bidirectional Transformer with rotary embeddings represents the velocity field, self-conditioning refines the estimate without additional sampling cost, minibatch optimal transport reduces path crossings, and a floored loss weight preserves low-noise supervision. Across multiple datasets and metrics, RecFlowTime outperforms baselines on key measures of distributional fidelity and temporal realism. It generates samples using 20 function evaluations, making sampling over an order of magnitude faster than both diffusion baselines. Ablations identify the rectified-flow core as the largest contributor to performance, followed by self-conditioning. Although trained unconditionally, RecFlowTime supports conditional generation for imputation and forecasting without retraining or additional evaluations.

AI USE STATEMENT

We did not use generative AI tools in the research, implementation, analysis, or writing of this work.

REFERENCES

Ahmed Alaa, Alex James Chan, and Mihaela van der Schaar. Generative time-series modeling with fourier flows. In *International Conference on Learning Representations*, 2021.

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations*, 2023.

Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *Transactions on Machine Learning Research*, 2023.

Rongyao Cai, Yuxi Wan, Kexin Zhang, Ming Jin, Zhiqiang Ge, Qingsong Wen, and Yong Liu. Parallel complex diffusion for scalable time series generation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '26)*, 2026.

Luis M. Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81–97, 2017.

Kuo-Tsai Chen. Integration of paths, geometric invariants and a generalized Baker-Hausdorff formula. *Annals of Mathematics*, 65(1):163–178, 1957.

Andrea Coletta, Sriram Gopalakrishan, Daniel Borrajo, and Svitlana Vyetrenko. On the constrained time-series generation problem. *arXiv preprint arXiv:2307.01717*, 2023.

Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation, 2021.

Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

Elizabeth Fons, Alejandro Sztrajman, Yousef El-laham, Alexandros Iosifidis, and Svitlana Vyetrenko. Hypertime: Implicit neural representation for time series, 2022.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851, 2020.

Yang Hu, Xiao Wang, Zezhen Ding, Lirong Wu, Huatian Zhang, Stan Z. Li, Sheng Wang, Jiheng Zhang, Ziyun Li, and Tianlong Chen. Flowts: Time series generation via rectified flow. *arXiv preprint arXiv:2411.07506*, 2024.

Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Time-series generation by contrastive imitation. *Advances in Neural Information Processing Systems*, 34:28968–28982, 2021.

Paul Jeha, Michael Bohlke-Schneider, Pedro Mercado, Shubham Kapoor, Rajbir Singh Nirwan, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. PSA-GAN: Progressive self attention GANs for synthetic time series. In *International Conference on Learning Representations*, 2022.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. DiffWave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2021.

Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.

Shujian Liao, Hao Ni, Lukasz Szpruch, Magnus Wiese, Marc Sabate-Vidales, and Baoren Xiao. Conditional Sig-Wasserstein GANs for time series generation. *Mathematical Finance*, 2024. doi: 10.1111/mafi.12423.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

Chung I Lu and Julian Sester. Generative modelling of financial time series with structured noise and mmd-based signature learning. *arXiv preprint arXiv:2407.19848*, 2024.

Terry J. Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 14(2):215–310, 1998.

Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training, 2016.

Aram-Alexandre Pooladian, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. Multisample flow matching: Straightening flows with minibatch couplings. In *International Conference on Machine Learning*, 2023.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, pp. 8857–8868. PMLR, 2021.

Lifeng Shen and James T. Kwok. Non-autoregressive conditional diffusion models for time series prediction. In *International Conference on Machine Learning*, volume 202 of *PMLR*, pp. 31016–31029, 2023.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063.

Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24804–24816, 2021.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. In *International Joint Conference on Artificial Intelligence*, pp. 4653–4660, 2021. doi: 10.24963/ijcai.2021/631.

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

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

Xinyu Yuan and Yan Qiao. Diffusion-ts: Interpretable diffusion for general time series generation. In *International Conference on Learning Representations*, 2024.

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

A Related Work

Adversarial and variational generators. Early deep generative models for time series were largely based on GANs (Goodfellow et al., 2020). C-RNN-GAN (Mogren, 2016) and RCGAN (Esteban et al., 2017), for example, use recurrent generators and discriminators, whereas TimeGAN (Yoon et al., 2019) adds a learned embedding space and a supervised next-step prediction loss. COT-GAN (Xu et al., 2020) replaces the standard adversarial objective with causal optimal transport. Beyond adversarial models, TimeVAE (Desai et al., 2021) uses a variational autoencoder with trend and seasonality blocks. Other approaches include frequency-domain normalizing flows (Alaa et al., 2021), contrastive imitation (Jarrett et al., 2021), and implicit neural representations (Fons et al., 2022). Despite these advances, adversarial methods remain susceptible to unstable optimization and mode collapse, whereas variational models can produce overly smooth sequences.

Diffusion models for time series. Denoising diffusion models (Ho et al., 2020) have been applied to forecasting (Rasul et al., 2021; Shen & Kwok, 2023), imputation (Tashiro et al., 2021; Alcaraz & Strodthoff, 2023), and unconditional generation. DiffWave (Kong et al., 2021), originally developed for audio, is a common generation baseline, while DiffTime (Coletta et al., 2023) formulates constrained time-series generation as a diffusion process. Diffusion-TS (Yuan & Qiao, 2024) predicts clean samples through an interpretable seasonal–trend decoder and includes a Fourier-domain reconstruction loss. PaCoDi (Cai et al., 2026) instead constructs the diffusion path in the frequency domain, where the Fourier transform simplifies temporal dependencies, and generates samples using DDIM (Song et al., 2021). Despite their strong performance, diffusion models incur high sampling costs and may accumulate errors across denoising steps, potentially reducing temporal fidelity.

Flow matching and rectified flow. Flow matching (Lipman et al., 2023), rectified flow (Liu et al., 2023), and stochastic interpolants (Albergo & Vanden-Eijnden, 2023) were developed primarily in the context of continuous generative modeling and have been evaluated extensively on image-generation tasks. These approaches learn a velocity field along a chosen probability path and generate samples by transporting noise directly to data through an ordinary differential equation (ODE). When the learned trajectories are sufficiently straight, the ODE can be integrated accurately with substantially fewer steps than those typically required by diffusion models. Although these methods have achieved strong results in image generation, their application to time-series and other sequential data remains comparatively limited (Hu et al., 2024). Accordingly, our framework adopts rectified flow as its generative core and adapts it to model the temporal structure of sequential data.

B Implementation Details

Algorithm 1 gives the complete training procedure of RecFlowTime, and Figure 5 details the velocity-field architecture.

Architecture. The input and output mappings are defined in Equations 3 and 4. Each sequence is represented by T tokens, one per time step, without patching or downsampling. The network contains four pre-normalization Transformer blocks. Each block applies unmasked multi-head attention with rotary embeddings on the query and key representations, followed by a feed-forward layer with expansion factor four. Because attention is bidirectional, every position can use the complete sequence at each model evaluation. A final layer normalization and linear projection return an F-dimensional velocity at every time step. Table 9 reports the model width and parameter count for each dataset.

Optimization. We train for 8,000 steps with batch size $B = 128$ using Adam with learning rate 2×10^{-4} and coefficients $(0.9, 0.96)$. The learning rate increases linearly during the first 500 steps and then follows a cosine decay. Gradients are clipped to norm 1.0. After every optimization step, we update an exponential moving average of the parameters with decay 0.999. The averaged parameters are used for sampling and evaluation. Table 6 reports the complete configuration.

Coupling overhead. The coupling overhead includes construction of the pairwise cost matrix and solution of the assignment problem. With $B = 128$, the complete operation takes 1.19 ms at $T = 64$

Algorithm 1 Training procedure for RecFlowTime.

Require: dataset $\mathcal{D}$; training steps S; batch size B; EMA decay β

initialize model parameters θ and EMA parameters $\bar{\theta} \leftarrow \theta$
for $s = 1, \dots, S$ **do**
  sample $\{\boldsymbol{x}^{(i)}\}_{i=1}^{B}$ from $\mathcal{D}$
  sample $\{\boldsymbol{\epsilon}^{(j)}\}_{j=1}^{B}$ independently from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  solve Equation 6 for $\sigma^\star$ and reorder the noise batch
  sample $t^{(i)} \sim \mathcal{U}[0,1]$ for each $i \in \{1, \dots, B\}$
  form $\boldsymbol{x}_t^{(i)} = (1 - t^{(i)})\boldsymbol{x}^{(i)} + t^{(i)}\boldsymbol{\epsilon}^{(\sigma^\star(i))}$
  set $\boldsymbol{c}^{(i)} \leftarrow \mathbf{0}$ for all i
  sample $b_{\mathrm{sc}} \sim \mathrm{Bernoulli}(0.5)$
  if $b_{\mathrm{sc}} = 1$ **then**
    compute $\tilde{\boldsymbol{v}}^{(i)} \leftarrow v_\theta(\boldsymbol{x}_t^{(i)}, t^{(i)}, \mathbf{0})$ without gradients
    set $\boldsymbol{c}^{(i)} \leftarrow \mathrm{stopgrad}\left(\boldsymbol{x}_t^{(i)} - t^{(i)}\tilde{\boldsymbol{v}}^{(i)}\right)$
  end if
  compute

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{B} w(t^{(i)}) \left\| v_\theta(\boldsymbol{x}_t^{(i)}, t^{(i)}, \boldsymbol{c}^{(i)}) - \left(\boldsymbol{\epsilon}^{(\sigma^\star(i))} - \boldsymbol{x}^{(i)}\right) \right\|_2^2$$

  update θ using $\nabla_\theta \mathcal{L}$
  update $\bar{\theta} \leftarrow \beta\bar{\theta} + (1 - \beta)\theta$
end for
return $\bar{\theta}$

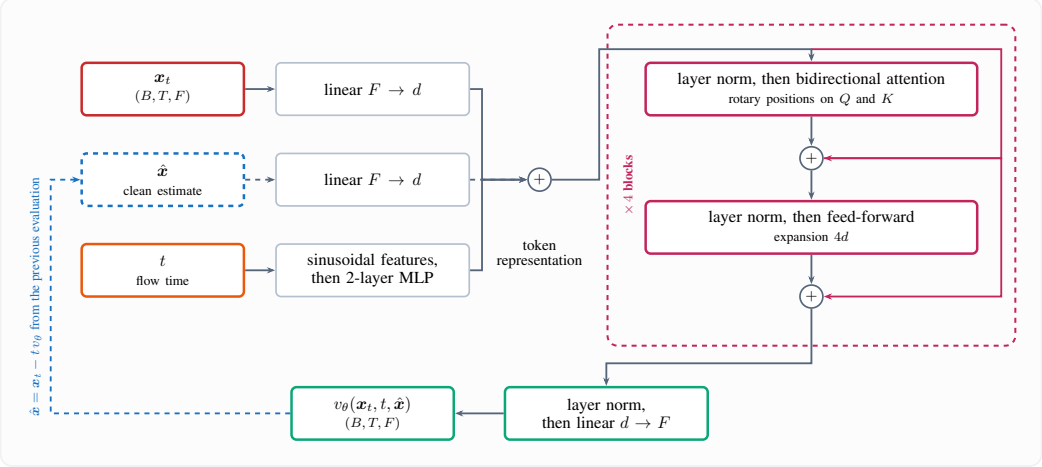

Figure 5: Architecture of the velocity field v_θ. Linear projections combine the intermediate sequence $\boldsymbol{x}_t$, the self-conditioning estimate $\hat{\boldsymbol{x}}$, and the flow-time embedding into one token per time step. Four pre-normalization Transformer blocks apply bidirectional attention with rotary embeddings and feed-forward layers, after which a linear head returns an F-dimensional velocity at each position. The dashed path shows self-conditioning: the clean-sequence estimate derived from the preceding evaluation is provided as input to the next evaluation.

and 1.22 ms at $T = 384$. This overhead is incurred only during training, and its relative contribution decreases as the cost of processing longer sequences increases.

Sampling implementation. Sampling uses the EMA parameters and the 20-step Euler procedure in Equation 8. The self-conditioning input is initialized to zero. After each network evaluation, the

predicted velocity is converted into a clean-sequence estimate using Equation 5, and this estimate is reused at the next step. Because the data are scaled to $[-1, 1]$, the estimate is clipped to this range before it is used for self-conditioning. The clipping affects only the conditioning input and does not modify the Euler update. Transport coupling and loss weighting are not used during sampling.

C Proof of Proposition 1

Proof. **Rectified flow.** Equation 5 gives

$$\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}) = \boldsymbol{x}_t - t\, v_\theta(\boldsymbol{x}_t, t, \boldsymbol{c}).$$

For fixed $\boldsymbol{x}_t$, t, and $\boldsymbol{c}$, the clean-sequence estimate is an affine function of the network output, with linear part $-t\boldsymbol{I}$. Therefore,

$$\frac{\partial \hat{\boldsymbol{x}}_\theta}{\partial v_\theta} = -t\boldsymbol{I}.$$

Its operator norm is t, which is at most 1 for every $t \in [0, 1]$.

Noise prediction. The diffusion forward process is

$$\boldsymbol{x}_\tau = \sqrt{\bar{\alpha}_\tau}\, \boldsymbol{x} + \sqrt{1 - \bar{\alpha}_\tau}\, \boldsymbol{\epsilon}.$$

Solving for $\boldsymbol{x}$ and replacing the true noise with the predicted noise gives

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_\tau - \sqrt{1 - \bar{\alpha}_\tau}\, \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau)}{\sqrt{\bar{\alpha}_\tau}}.$$

For fixed $\boldsymbol{x}_\tau$ and τ, the Jacobian with respect to the predicted noise is

$$\frac{\partial \hat{\boldsymbol{x}}}{\partial \boldsymbol{\epsilon}_\theta} = -\sqrt{\frac{1 - \bar{\alpha}_\tau}{\bar{\alpha}_\tau}}\, \boldsymbol{I}.$$

Its operator norm is $\sqrt{(1 - \bar{\alpha}_\tau)/\bar{\alpha}_\tau}$, which diverges as $\bar{\alpha}_\tau \to 0$, corresponding to the high-noise limit.

□

D Datasets

Table 5: Summary of the four datasets. N denotes the number of sequences obtained after windowing, T denotes sequence length, and "Features" denotes the number of variables per time step.

Dataset	Features	N	T	Source
Sines	4	10,000	64	Synthetic
Stocks	6	4,304	128	Google, 2004–2022
Energy	5	19,640	96	Candanedo et al. (2017)
ECG	2	8,000	192	Synthetic

Sines. We simulate multivariate, continuous-valued sequences with periodic temporal structure. Each feature is generated as $0.5(\sin(\eta t + \theta) + 1)$, where $\eta \sim \mathcal{U}[0.1, 0.2]$ and $\theta \sim \mathcal{U}[0, 0.1]$ are sampled independently for every feature and sequence. This dataset provides a controlled setting for evaluating whether a model can reproduce smooth periodic patterns.

Stocks. The Stocks dataset contains continuous-valued, aperiodic financial sequences with strong correlation among features. The six features are opening price, highest price, lowest price, closing price, adjusted closing price, and trading volume. This dataset is useful for evaluating multivariate dependencies and irregular temporal variation. Our source covers 2004–2022, whereas the benchmark version used by Yuan & Qiao (2024) covers 2004–2019.

Energy. The Energy dataset (Candanedo et al., 2017) contains continuous-valued household measurements characterized by noisy periodic behavior and correlated features. We use five features: *Appliances*, *lights*, *T4*, *RH_4*, and *T9*. This subset retains both energy-consumption and environmental measurements while providing a compact multivariate benchmark with more complex temporal behavior than the Stocks data.

ECG. We simulate each ECG lead as

$$x_f(t) = \sum_b \sum_{k=1}^{5} a_{b,k,f} \exp\left[-\frac{(t-\mu_{b,k,f})^2}{2\sigma_{b,k,f}^2}\right] + \epsilon_f(t), \qquad \epsilon_f(t) \sim \mathcal{N}(0, 0.02^2),$$

where the five Gaussian components represent the P, Q, R, S, and T waves. Their amplitudes $a_{b,k,f}$, positions $\mu_{b,k,f}$, and widths $\sigma_{b,k,f}$ vary across beats, while lead-specific gains and temporal offsets produce correlated but distinct signals. This dataset evaluates whether a model can preserve sharp local waveforms and repeated temporal structure.

Preprocessing. The resulting dataset sizes and sequence dimensions are summarized in Table 5. For all four datasets, each feature is scaled to $[-1, 1]$ using its minimum and maximum values across all sequences and time steps. This symmetric range provides a consistent, zero-centered scale for learning transport from the Gaussian prior. Using a fixed random seed, we then allocate 90% of the sequences to training and 10% to testing. All models are trained for a fixed number of optimization steps, without early stopping or checkpoint selection. The test split provides the real sequences used to evaluate RecFlowTime and all baselines.

E Evaluation Metrics and Protocol

We evaluate each method quantitatively and qualitatively. The quantitative evaluation uses learned metrics, cross-feature correlation, and deterministic measures of temporal fidelity, as reported in Tables 1 and 2. The qualitative evaluation uses visualization grids to compare distributional coverage, generated sequences, and temporal dependencies.

Quantitative evaluation. All methods are trained on the same data splits and evaluated using the same code. Consequently, differences in their scores cannot be attributed to different datasets or metric implementations. We implement the discriminative and predictive scores following the TimeGAN protocol (Yoon et al., 2019) and compute Context-FID from TS2Vec representations (Jeha et al., 2022; Yue et al., 2022). Lower values indicate better performance for every metric.

- **Discriminative score.** A GRU classifier is trained for 1,200 iterations to distinguish real from generated sequences. Its hidden size is $\max(8, 4F)$: four times the number of input features, with a minimum of eight hidden units. The real and generated sets are independently divided into 80% training and 20% test subsets. The evaluator performs binary classification, and its score is $|\text{accuracy} - 0.5|$ on the combined test set. A lower score indicates that the generated sequences are more difficult to distinguish from real ones and therefore exhibit greater sample realism.
- **Predictive score.** A GRU with $\max(8, 4F)$ hidden units is trained on generated sequences for 1,200 iterations. Given all F features through time t, it predicts the features at time $t + 1$. The score is the mean absolute prediction error on real sequences. This metric evaluates whether the generated data preserve the temporal relationships required for next-step prediction.
- **Context-FID.** We train a TS2Vec encoder exclusively on the real training split. The fitted encoder is then frozen and used to encode the held-out real sequences and generated sequences. TS2Vec produces a representation at every time step, and max pooling over time converts these outputs into one 128-dimensional vector per sequence. Context-FID compares the means and covariance matrices of the real and generated representation vectors using the Fréchet distance. A lower score indicates that the generated sequences occupy a distribution closer to that of the real sequences in the learned representation space.

- **Correlational score.** This metric evaluates whether the generated data preserve simultaneous relationships among features. For the real and generated sets separately, we pool all sequences and time steps and standardize each feature. We then compute the cross-feature correlation matrix for each set and sum the absolute differences between corresponding matrix entries. A lower score indicates closer agreement between the real and generated feature dependencies.
- **Spectral distance.** This metric evaluates whether real and generated sequences contain similar periodic behavior and frequency-dependent variation. For every sequence and feature, we compute the power spectrum along the temporal dimension, add a small numerical constant, and take its logarithm. The log power spectra are then averaged across sequences. The score is the mean absolute difference between the real and generated spectra across all frequencies and features. A lower score indicates better agreement in frequency content.
- **Increment distance.** This metric evaluates local changes between consecutive time steps. For each feature, we compute the signed step-to-step differences and pool them across sequences and time. We then calculate the 1-Wasserstein distance between the real and generated increment distributions and average the result across features. A lower score indicates that the generated data better reproduce the magnitude and direction of local changes, including smooth transitions, fluctuations, and abrupt movements.
- **Autocorrelation distance.** This metric evaluates whether each feature depends on its previous values in the same way in real and generated sequences. For every sequence and feature, we subtract its temporal mean and normalize by its temporal variance. We compute autocorrelation values over the first $\min(T-1, 12)$ temporal lags and average them across sequences. The score is the mean absolute difference between the real and generated autocorrelations across lags and features. A lower score indicates better preservation of short-range persistence, oscillation, and repeated temporal behavior.

The spectral, increment, autocorrelation, and correlational scores are deterministic for fixed samples. The discriminative score, predictive score, and Context-FID depend on learned evaluators. The training split is used to fit the Context-FID encoder, while the held-out real sequences provide the reference distribution for all comparisons. Each method is evaluated using the same number of generated sequences. We train the discriminative network, predictive network, and TS2Vec encoder using two evaluator seeds and report the mean $\pm$ standard deviation. The reported uncertainty therefore reflects evaluator variation only.

Qualitative evaluation. We complement the quantitative metrics with two types of visualization.

- **Projection-based visualizations.** PCA and t-SNE reduce each complete sequence to two dimensions so that the distributions of real and generated samples can be compared visually. Figure 12 uses a linear PCA projection fitted on the real sequences and then applied to every method, making panels within each dataset directly comparable. Figure 13 uses a nonlinear embedding that emphasizes local neighborhood structure. Because each t-SNE panel is fitted independently, panels are compared by their distributional shape and overlap rather than their absolute coordinates.
- **Distributional and temporal visualizations.** Figure 16 compares the distributions of observed values after pooling over sequences, time steps, and features; it therefore evaluates value coverage without considering temporal order. Figure 17 displays autocorrelation across temporal lags, revealing persistence and repeated temporal structure. Figure 18 compares mean log power spectra, showing whether generated sequences reproduce the frequency content of the real data. Finally, Figure 19 compares distributions of signed step-to-step increments, highlighting differences in local variation, smoothness, and abrupt changes.

These qualitative visualizations are provided in Appendix H.

F Model and Baseline Configurations

This section documents the configurations used to train RecFlowTime and the four baselines. All methods use identical data splits and are evaluated with the same metric implementation and eval-

Table 6: Hyperparameters of RecFlowTime. The listed settings are shared across all datasets; dataset-specific model dimensions are reported in Table 9.

Setting	Value
Optimizer	Adam
Learning rate	2×10^{-4}
Betas	$(0.9, 0.96)$
Batch size	128
Training steps	8,000
Gradient clipping	1.0
EMA decay	0.999
Sampling steps (NFE)	20
min-SNR γ	5.0
min-SNR floor $w_{\min}$	0.1
Transformer layers	4
Attention heads	4
Feed-forward multiplier	4
Dropout	0.0

Table 7: Baseline implementations used in the experiments. We retrain every method on the same training and test splits used for RecFlowTime and evaluate all methods using the same metric implementation. The final column identifies the repository commit used for each baseline.

Method	Venue	Repository	Commit
TimeGAN (Yoon et al., 2019)	NeurIPS 2019	`https://github.com/jsyoon0823/TimeGAN`	`8f6181c`
TimeVAE (Desai et al., 2021)	arXiv 2021	`https://github.com/abudesai/timeVAE`	`9914576`
Diffusion-TS (Yuan & Qiao, 2024)	ICLR 2024	`https://github.com/Y-debug-sys/Diffusion-TS`	`566307e`
PaCoDi (Cai et al., 2026)	KDD 2026	`https://github.com/RongyaoCai/PaCoDi`	`8c77d91`

uator seeds. We retain each method's published configuration whenever possible, modifying only dimensions that must accommodate the sequence length or number of features. Architectures, parameter counts, and training budgets are not matched because the methods belong to different model families and use different training procedures.

F.1 RecFlowTime Configuration

Table 6 reports the hyperparameters shared across all four datasets. Appendix B describes the training procedure and network architecture, while Table 9 reports the dataset-specific model dimensions and parameter counts.

F.2 Baseline Selection and Implementations

We compare RecFlowTime with four time-series generators spanning three major model families. TimeGAN (Yoon et al., 2019) represents adversarial generation, while TimeVAE (Desai et al., 2021) represents variational generation. Diffusion-TS (Yuan & Qiao, 2024) is a time-domain diffusion model with a seasonal–trend decoder, whereas PaCoDi (Cai et al., 2026) performs diffusion in the frequency domain. Table 7 reports the exact code repository and commit used for each baseline.

Compatibility changes. TimeGAN was developed for TensorFlow 1.15, which is incompatible with current Python releases. We therefore run it through the TensorFlow 2 compatibility layer and replace `tf.contrib.layers.fully_connected` with `tf.layers.dense`, which uses the same Glorot initialization. Diffusion-TS and PaCoDi contain hard-coded CUDA calls, which we redirect to the available accelerator without changing their computations. For PaCoDi, we additionally use a single-process data loader and disable pinned memory; neither setting changes the model. We make no other modifications to the baseline implementations.

Table 8: Principal hyperparameters used for each method. Ranges denote settings that vary across datasets. Training budgets are reported in each method's native unit and are therefore not directly comparable.

	RecFlowTime	PaCoDi	Diffusion-TS	TimeGAN	TimeVAE
Family	Rectified flow	Spectral diffusion	Diffusion	GAN	VAE
Backbone	Transformer	DiT (real/imag)	Enc–dec Transformer	GRU	Dense
Width	96–160	128–256	64–96	24	50–200
Depth	4	4	1–4 enc, 2–3 dec	3	3
Batch size	128	64–128	64–128	128	64
Learning rate	2×10^{-4}	1×10^{-5}	1×10^{-5}	1×10^{-3}	1×10^{-3}
Training budget	8k steps	8k steps	12k steps	8k iters $\times 3$	1k epochs
Sampling NFE	20	200	500	1	1

Table 9: Architecture size of every method on each dataset. *Width* denotes d_{model} for RecFlowTime and Diffusion-TS, the embedding size for PaCoDi, the GRU hidden size for TimeGAN, and the sequence of channel widths for TimeVAE. *Layers* counts Transformer blocks for RecFlowTime and PaCoDi, encoder plus decoder blocks for Diffusion-TS, stacked GRU layers per TimeGAN subnetwork, and encoder or decoder blocks for TimeVAE. Parameter counts include trainable parameters only.

Dataset	Method	Width	Layers	Parameters
Sines	TimeGAN	24	3	48,437
	TimeVAE	50–100–200	3	257,568
	Diffusion-TS	64	$1 + 2$	254,364
	PaCoDi	128	4	2,129,800
	RecFlowTime	96	4	523,108
Stocks	TimeGAN	24	3	48,775
	TimeVAE	50–100–200	3	823,010
	Diffusion-TS	64	$2 + 2$	373,222
	PaCoDi	256	4	8,455,948
	RecFlowTime	128	4	927,622
Energy	TimeGAN	24	3	48,606
	TimeVAE	50–100–200	3	442,976
	Diffusion-TS	96	$4 + 3$	1,165,679
	PaCoDi	256	4	8,454,922
	RecFlowTime	112	4	710,981
ECG	TimeGAN	24	3	48,099
	TimeVAE	50–100–200	3	418,982
	Diffusion-TS	64	$3 + 2$	519,058
	PaCoDi	256	4	8,451,844
	RecFlowTime	160	4	1,444,322

F.3 Cross-Method Configurations and Capacity

Table 8 compares the principal architecture, optimization, and sampling settings used for all methods. Training budgets are expressed in each method's native unit and are therefore not directly equivalent: TimeVAE uses epochs, TimeGAN uses iterations within three training phases, and RecFlowTime, Diffusion-TS, and PaCoDi use optimizer steps. Similarly, the number of function evaluations reflects the native sampling procedure of each model family. Measured training and sampling times are reported separately in Table 4.

Table 9 reports the dataset-specific architecture dimensions and trainable parameter counts. The models are not capacity matched because their architectures differ substantially. TimeGAN contains approximately $48{,}000$ parameters, PaCoDi contains between 2.1 and 8.5 million, and RecFlowTime contains between 0.52 and 1.44 million.

RecFlowTime uses substantially fewer parameters than PaCoDi on every dataset and requires 20 function evaluations per sample, compared with 200 for PaCoDi and 500 for Diffusion-TS. Nevertheless, parameter count alone does not determine computational cost because the methods differ in architecture, training procedure, and number of sequential sampling steps. Table 4 therefore complements this comparison with runtimes measured on the same hardware.

G Ablation and Diagnostic Analyses

G.1 Full Ablation Analysis

Experimental design. We conduct a leave-one-out ablation to identify the contribution of each component of RecFlowTime. Every variant uses the same dataset-specific architecture, training budget, and evaluation protocol as the complete model, while changing only one component. The six variants are: (1) replacing transport coupling with independent pairing; (2) replacing rotary embeddings with fixed sinusoidal encodings; (3) removing self-conditioning; (4) removing the complete min-SNR weighting scheme; (5) removing only the minimum weight floor while retaining min-SNR weighting; and (6) replacing the rectified-flow core with ϵ-prediction diffusion. The diffusion variant uses a cosine noise schedule and DDIM sampling with the same number of steps while retaining the transport coupling and loss floor. Because these experiments change one component at a time, they measure its contribution within the complete system; they do not capture interactions among multiple removed components. Lower values indicate better performance for every metric.

Ablation results. Table 10 reports all seven metrics for the six ablations. Replacing rectified flow with diffusion produces the largest Context-FID degradation, increasing the score by 113% on average. Removing self-conditioning increases mean Context-FID by 36%, followed by min-SNR weighting (21%), the loss floor (15%), transport coupling (12%), and rotary embeddings (8%). The complete model achieves the lowest Context-FID on Sines, Stocks, and Energy, while removing rotary embeddings improves it on ECG. For the discriminative score, diffusion again produces the largest mean degradation ($1{,}183\%$), followed by rotary embeddings (143%), min-SNR weighting (99%), transport coupling (96%), self-conditioning (75%), and the loss floor (2%). These effects remain dataset dependent: removing coupling or self-conditioning improves the score only on Stocks, while removing rotary embeddings improves it only on ECG. Removing min-SNR weighting or its floor improves the score on Stocks and ECG but worsens it on Sines and Energy.

Table 10 also supplements the learned scores with predictive, spectral, increment, and autocorrelation metrics. The predictive score changes little across most variants and provides limited separation. In contrast, the deterministic metrics reveal component-specific effects. Removing transport coupling worsens increment distance on every dataset, consistent with its role in simplifying the transport geometry. Removing rotary embeddings worsens autocorrelation distance on every dataset, indicating that they help preserve lagged temporal relationships. Removing self-conditioning also worsens increment distance throughout, with its largest effect on ECG. The low-noise components primarily affect fine temporal structure. Removing only the loss floor increases increment distance by 255% on Sines and 48% on Energy and increases autocorrelation distance by 60% on Stocks and 124% on Energy. Appendix G.2 further shows that the floor reduces excessive sample roughness. The diffusion replacement improves a few individual temporal metrics despite its much worse Context-FID, demonstrating that no single summary metric captures overall sample quality.

Overall interpretation. The ablation assigns a distinct role to each component: rectified flow provides the largest improvement in distributional fidelity; self-conditioning refines local dynamics; min-SNR weighting balances supervision across noise levels; coupling improves local changes and path straightness; rotary embeddings preserve lag-dependent structure; and the loss floor controls roughness near the low-noise endpoint. Their effects are complementary, and no component improves every metric on every dataset.

Table 10: Full leave-one-out ablation, all metrics, all datasets, using the same configurations as the main experiments. Lower is better.

Dataset	Configuration	Context-FID	Disc.	Pred.	Spectral	Increment	ACF
Sines	RecFlowTime	0.3221	0.0087	0.0103	0.1095	0.0011	0.0041
	– transport coupling	0.3274	0.0187	0.0114	0.2478	0.0013	0.0033
	– rotary embeddings	0.3320	0.0175	0.0111	0.2037	0.0014	0.0046
	– self-conditioning	0.3404	0.0175	0.0111	0.1185	0.0013	0.0036
	– min-SNR weighting	0.3338	0.0275	0.0097	0.1698	0.0012	0.0034
	– min-SNR floor only	0.3445	0.0102	0.0150	0.2381	0.0039	0.0029
	rectified flow → diffusion	0.5214	0.2550	0.0112	0.0784	0.0030	0.0088
Stocks	RecFlowTime	1.5773	0.0259	0.0123	0.3703	0.0039	0.0262
	– transport coupling	1.9248	0.0115	0.0123	0.4206	0.0047	0.0215
	– rotary embeddings	2.0626	0.0374	0.0128	0.4959	0.0029	0.0369
	– self-conditioning	1.9599	0.0201	0.0124	0.5101	0.0043	0.0284
	– min-SNR weighting	1.9476	0.0230	0.0124	0.4254	0.0040	0.0447
	– min-SNR floor only	1.9785	0.0115	0.0126	0.5358	0.0039	0.0418
	rectified flow → diffusion	5.9841	0.1983	0.0125	1.0573	0.0050	0.0215
Energy	RecFlowTime	0.8304	0.0127	0.0295	0.2191	0.0095	0.0276
	– transport coupling	0.8446	0.0509	0.0297	0.3130	0.0103	0.0297
	– rotary embeddings	0.9381	0.0757	0.0300	0.3459	0.0099	0.0395
	– self-conditioning	0.8348	0.0324	0.0295	0.2668	0.0096	0.0235
	– min-SNR weighting	1.0196	0.0426	0.0295	0.4096	0.0087	0.0366
	– min-SNR floor only	0.8340	0.0219	0.0302	0.5029	0.0141	0.0617
	rectified flow → diffusion	1.6199	0.1698	0.0294	0.3520	0.0119	0.0417
ECG	RecFlowTime	2.1221	0.1141	0.0570	0.5909	0.0137	0.0068
	– transport coupling	2.5892	0.1422	0.0574	0.5701	0.0162	0.0104
	– rotary embeddings	1.8487	0.0344	0.0554	0.2157	0.0097	0.0106
	– self-conditioning	4.5555	0.1875	0.0589	0.7222	0.0219	0.0181
	– min-SNR weighting	2.8414	0.0625	0.0565	0.6428	0.0147	0.0093
	– min-SNR floor only	2.6618	0.0844	0.0559	0.2404	0.0127	0.0086
	rectified flow → diffusion	2.4455	0.1109	0.0539	0.4869	0.0051	0.0174

G.2 Effect of low-noise supervision

We conduct a controlled experiment to determine whether the min-SNR floor reduces excessive local variation in generated sequences. Specifically, we train three otherwise identical models on Sines using $w_{\min} \in \{0, 0.05, 0.10\}$, while holding all other settings fixed. Increasing this floor strengthens supervision near the low-noise endpoint, where the model refines small temporal changes. We evaluate its effect using normalized roughness together with Context-FID, spectral distance, and the discriminative score. Normalized roughness is the mean absolute second difference of the generated sequences divided by that of the real sequences; thus, a value of 1.0 indicates equal local variation.

As shown in Table 11, increasing $w_{\min}$ from 0 to 0.10 reduces normalized roughness from 2.60 to 1.25. Figure 6 shows that this reduction corresponds to the removal of high-frequency fluctuations that are difficult to identify from the sequence values alone but appear clearly in their second differences. Spectral distance also improves from 0.224 to 0.109, while Context-FID changes only slightly, from 0.327 to 0.322. The discriminative score increases from 0.005 to 0.009 but remains close to its ideal value of zero. Overall, $w_{\min} = 0.10$ provides the closest match to real-data roughness while preserving distributional fidelity.

G.3 Path Straightness

The purpose of transport coupling is to simplify the geometry of the learned velocity field by reducing crossings among the data–noise interpolation paths. We test this mechanism directly through a controlled comparison between RecFlowTime and its independent-pairing ablation. The two models use the same architecture, training procedure, and 20-step Euler sampler; they differ only in how data and noise samples are paired during training. For each model, we measure how strongly the direction of the sampling trajectory changes between successive integration steps. Lower curvature

Table 11: Effect of the min-SNR floor on Sines with all other settings held fixed. Roughness is the mean absolute second difference normalized by the corresponding value for the real data; therefore, 1.0 indicates equal roughness.

$w_{\min}$	Roughness ($\times$ real)	Context-FID	Spectral	Discriminative
0.00	2.60	0.327	0.224	**0.005**
0.05	1.77	0.326	0.202	0.010
0.10	**1.25**	**0.322**	**0.109**	0.009

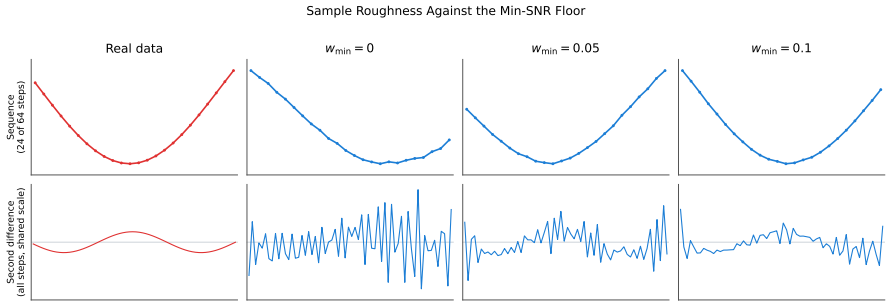

Figure 6: Sample roughness on Sines at each value of $w_{\min}$. The upper row shows a 24-step window of a generated sequence; the lower row shows its second difference over all steps, on a scale shared across columns. The second difference is the quantity the roughness column of Table 11 averages.

indicates a straighter trajectory that can be approximated more accurately using a small number of steps. Figure 7 reports this measurement throughout the sampling process.

As shown in Figure 7, transport coupling reduces average trajectory curvature by 47% on Sines, 45% on Stocks, 33% on Energy, and 23% on ECG. The reduction is observed throughout the sampling trajectory and is large relative to the variation across samples. These results directly support the proposed geometric mechanism: transport coupling produces a straighter marginal velocity field, allowing the sampling trajectory to be integrated accurately using 20 Euler steps. Its clearest contribution is therefore improved transport geometry rather than a uniform improvement across every sample-quality metric.

G.4 Sample Quality and Number of Euler Steps

The main experiments use 20 Euler steps for generation. To examine the trade-off between sampling cost and sample quality, we keep each trained model fixed and vary only the number of Euler steps used at inference. Table 4 compares the resulting 20-step sampling cost with those of the baselines, while Figure 8 reports how Context-FID changes as the number of steps increases. Each setting is evaluated using 600 generated samples and a separately fitted Context-FID encoder. Therefore, comparisons within the sweep primarily indicate trends across step counts, whereas the absolute values are not directly comparable with those in Table 1.

The effect of additional steps varies across datasets. On Sines, Context-FID stabilizes after eight steps: it is 0.557 at 8 steps, 0.548 at 20 steps, and 0.557 at 128 steps. Thus, increasing the sampling cost more than sixfold beyond the 20-step setting provides no measurable improvement. On Stocks, Energy, and ECG, sample quality continues to improve as the number of steps increases. Raising the step count from 20 to 128 reduces Context-FID by 25% on Stocks, 15% on Energy, and 44% on ECG, while increasing sampling time by approximately $6.6\times$. Spectral distance exhibits a similar trend, decreasing by 52%, 28%, and 81%, respectively. In contrast, spectral distance on Sines is lowest at four steps and gradually increases thereafter. These results indicate that the more complex temporal structure of Stocks, Energy, and ECG benefits from finer numerical integration, whereas the simpler sinusoidal trajectories are represented accurately with relatively few steps.

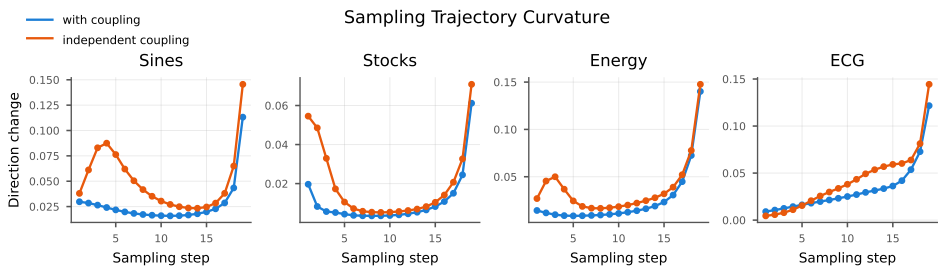

Figure 7: Sampling-trajectory curvature at each integration step, with and without the transport coupling. Lower values indicate straighter trajectories and, therefore, a more favorable geometry for few-step integration.

Accordingly, we do not claim that 20 steps maximize sample quality. Instead, this setting provides a practical balance between fidelity and computational cost: it achieves strong results in Table 1 while requiring $10\times$ fewer function evaluations than PaCoDi and $25\times$ fewer than Diffusion-TS.

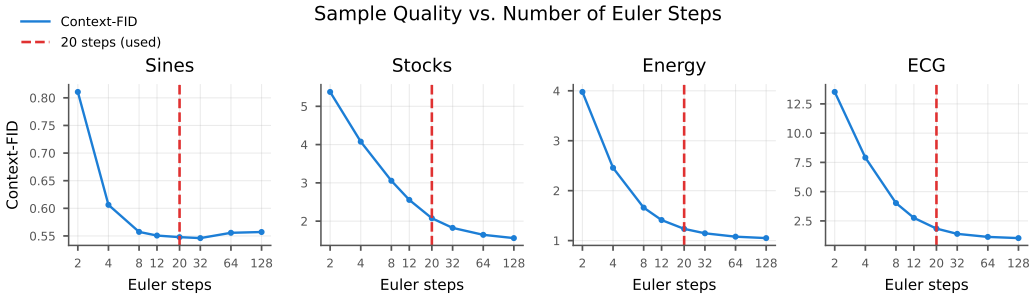

Figure 8: Context-FID as a function of the number of Euler steps. Samples are generated with the deployed integrator, including self-conditioning and the clamp on the clean-sequence estimate. The dashed line marks the 20-step setting used in the main experiments. Each point uses 600 samples and a separately fitted encoder; therefore, values are comparable within each curve.

G.5 Training Dynamics

This section examines the 8,000 optimization steps used to train the models reported in Table 1, with one run per dataset. The analysis assesses whether optimization stabilizes within the training budget and whether the schedule in Table 6 produces consistent training behavior.

Figure 9 evaluates sample quality every 500 steps. The discriminative score initially approaches 0.5, indicating that the evaluator can distinguish real from generated sequences almost perfectly. It reaches its minimum region by approximately step 3,000 on Sines, Stocks, and Energy and by step 4,000 on ECG, which has the longest sequences. Subsequent fluctuations are comparable to the evaluator variability; therefore, they do not provide clear evidence of further improvement or degradation. The predictive score stabilizes earlier, at approximately step 2,000. These results indicate that the principal gains occur well before the end of training and that the 8,000-step budget does not truncate the initial period of rapid improvement.

Figure 10 shows an overall smooth decline in the training loss under a single continuous schedule, with no evidence of divergence or sustained oscillation. The loss continues to decrease after the evaluation metrics have stabilized; on Sines, for example, it falls from 0.060 at step 2,000 to 0.036 at step 8,000. This difference arises because the objective averages velocity-regression errors over the entire transport path, including regions whose improvement may have little effect on the evaluated samples. Consequently, the training loss alone is not a reliable criterion for early stopping. Its absolute value should also not be compared across datasets; the final values are approximately 0.007 on Stocks and 0.040 on Energy.

Figure 11 shows a brief period of large gradients early in training. The norm peaks at 9.3 on Sines near step 550 and between 5.1 and 7.7 within the first 25 steps on the other datasets. It then decreases rapidly, with the binned mean remaining below the clipping threshold of 1.0. Overall, clipping affects only 5%–14% of the logged steps; the final clipped step occurs at step 2,950 on Sines and earlier on the remaining datasets. Thus, the complete training configuration exhibits an initial optimization transient but no persistent gradient instability.

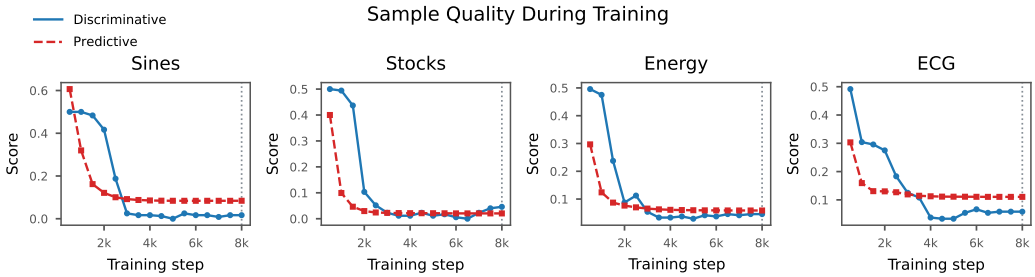

Figure 9: Discriminative and predictive scores measured every 500 training steps; lower values indicate better performance. Both scores improve rapidly during the first 3,000–4,000 steps and subsequently stabilize, indicating that most gains occur during the first half of training.

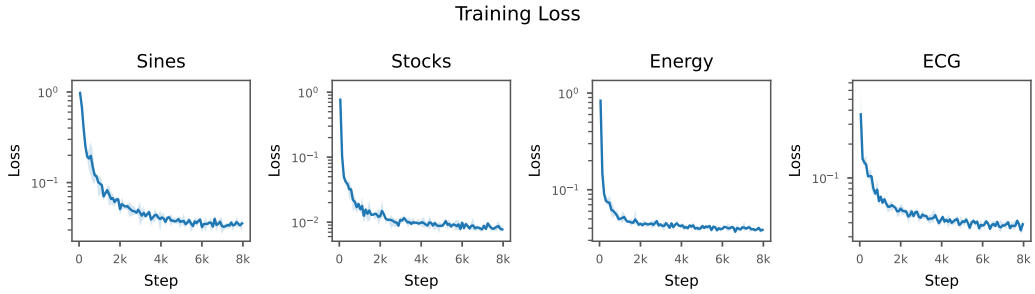

Figure 10: Flow-matching loss over training. Logged values are grouped into approximately 90 bins per dataset; the solid line shows the mean within each bin, and the shaded region shows the corresponding range. The absence of sustained increases or oscillations indicates stable optimization across all four datasets.

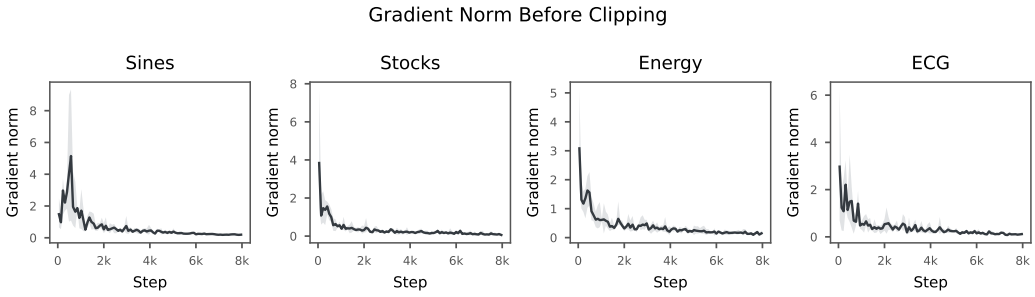

Figure 11: Gradient norm before clipping, grouped into the same bins as Figure 10. The solid line shows the mean within each bin, and the shaded region shows the corresponding range. Large gradients occur primarily at the beginning of training and decrease as optimization progresses, indicating that clipping is most active during the initial stage.

H Additional Qualitative Results

This section presents additional visual comparisons of real and generated sequences. Figure 12 visualizes their distributions using PCA, while Figure 13 provides a complementary nonlinear visualization using t-SNE. Figure 14 displays real and RecFlowTime-generated sequences for every feature, while Figure 15 compares representative samples from all evaluated methods. Figure 16 compares marginal value distributions without considering temporal order. Figure 17 compares lagged temporal dependence through autocorrelation, and Figure 18 compares frequency content through log power spectra. Figure 19 compares the distributions of step-to-step changes.

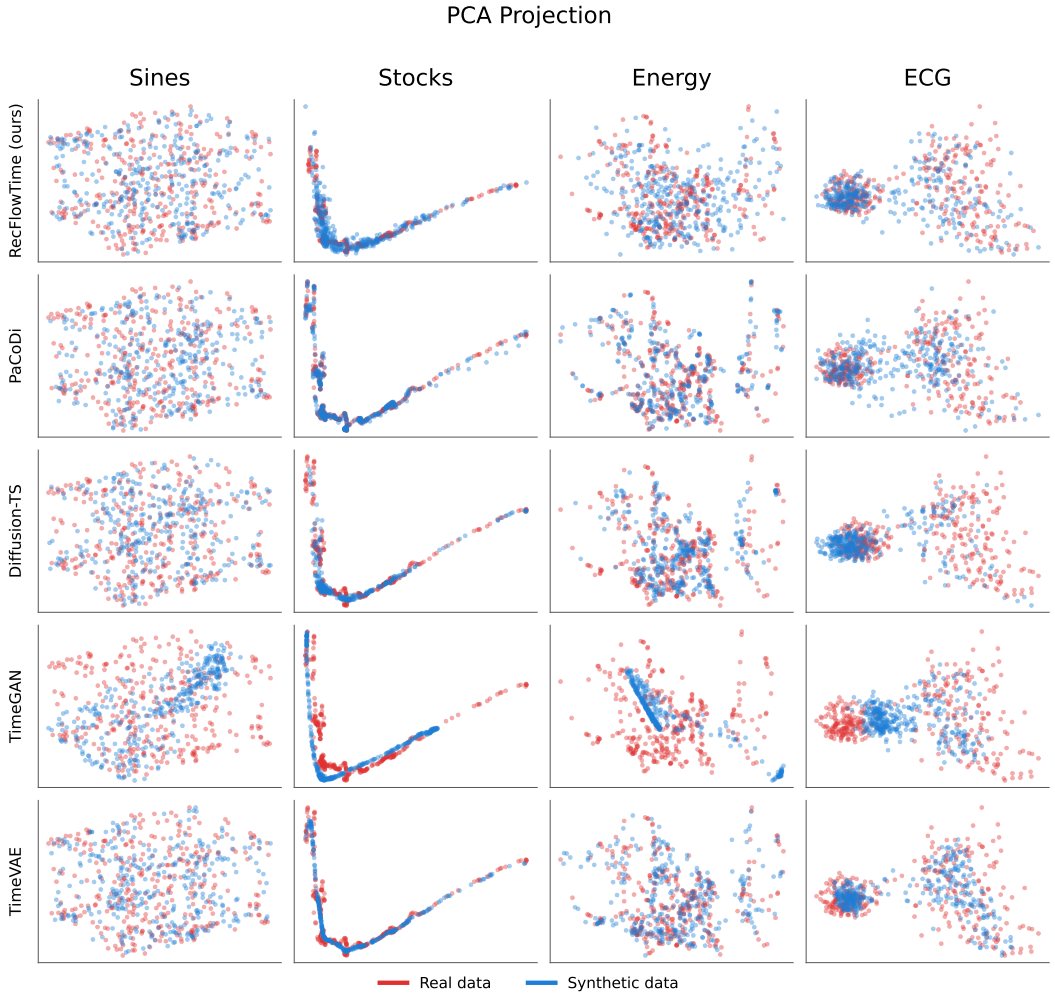

Figure 12: PCA projections of real (red) and generated (blue) sequences. Rows correspond to methods, and columns correspond to datasets. For each dataset, the projection is fitted once on the real sequences and then applied to every method; panels within each column are therefore directly comparable. Overlap between the red and blue points therefore indicates linear/global distributional coverage.

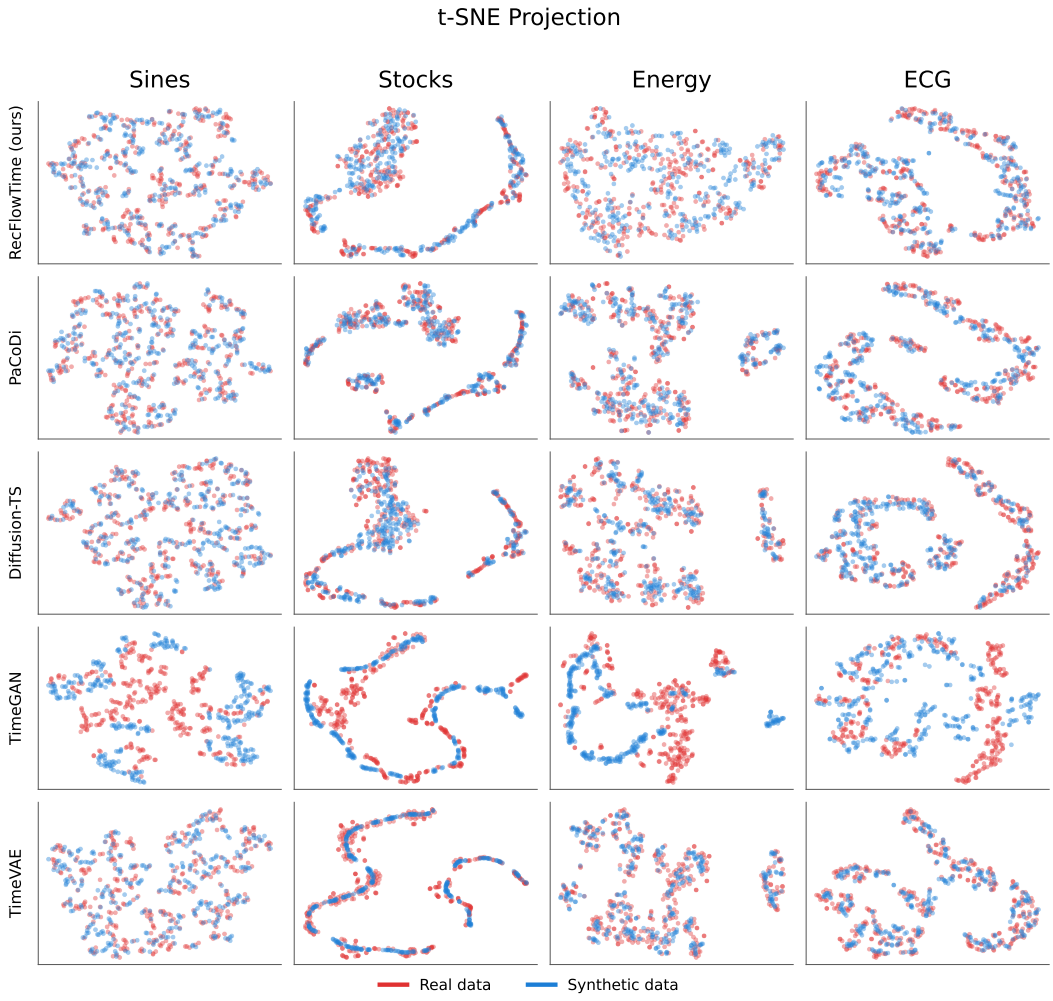

Figure 13: t-SNE projections of real (red) and generated (blue) sequences. t-SNE maps complete sequences to two dimensions while approximately preserving local neighborhoods; overlap between the red and blue points therefore indicates similarity in local distributional structure. Because each panel uses an independently fitted embedding, panels should be compared by distributional shape rather than by absolute position or orientation.

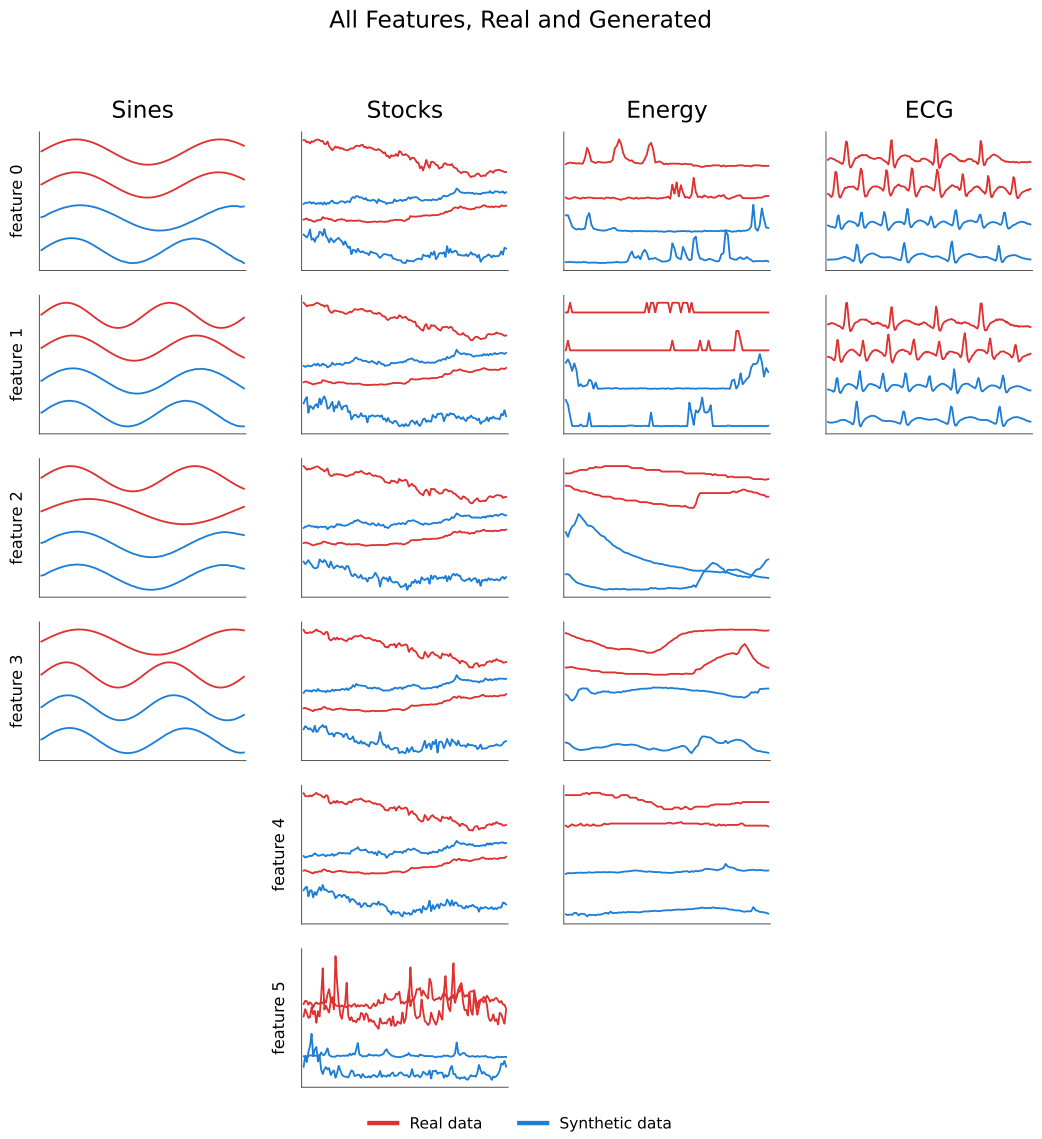

Figure 14: Feature-wise comparison of real and RecFlowTime-generated sequences. Each panel contains two real sequences in red and two generated sequences in blue, with the real traces placed above the generated traces. Columns correspond to datasets, and rows correspond to feature indices; empty positions occur because the datasets contain different numbers of features. The same vertical spacing, determined from the real sequences, is applied to all four traces in each panel. Consequently, mismatches in amplitude, local variation, or smoothness remain visible.

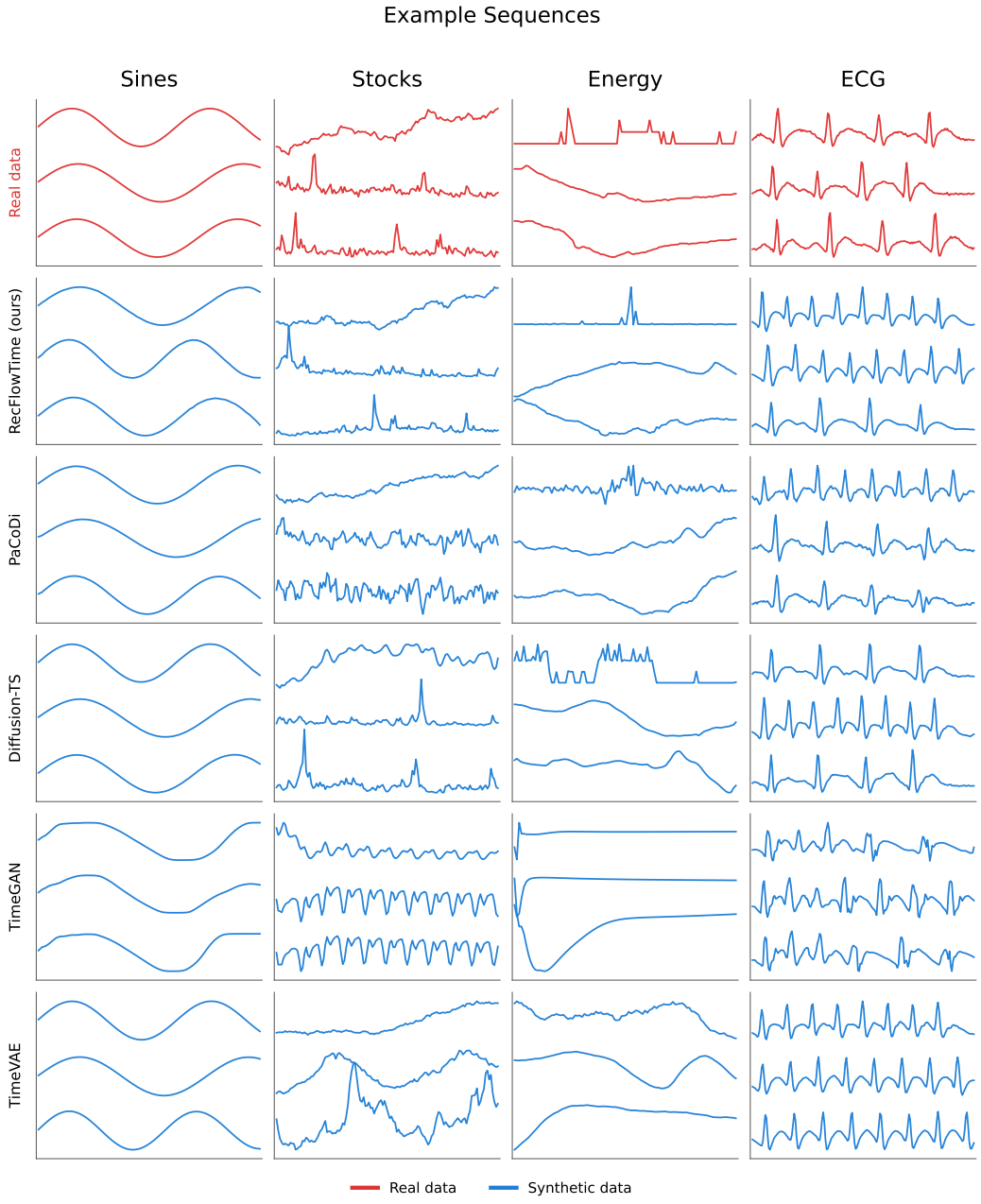

Figure 15: Representative sequences from the real data and from every generation method. Rows identify the source, the real data first and each method below it; columns identify datasets. Sines and ECG show three sequences of feature 0, their remaining features being statistically interchangeable. Stocks shows one sequence of feature 0 and two of feature 5. Energy shows one sequence of feature 1, the top trace in each Energy panel, and two of feature 3. The figure provides a qualitative comparison of temporal shape, amplitude, variability, and local smoothness across methods.

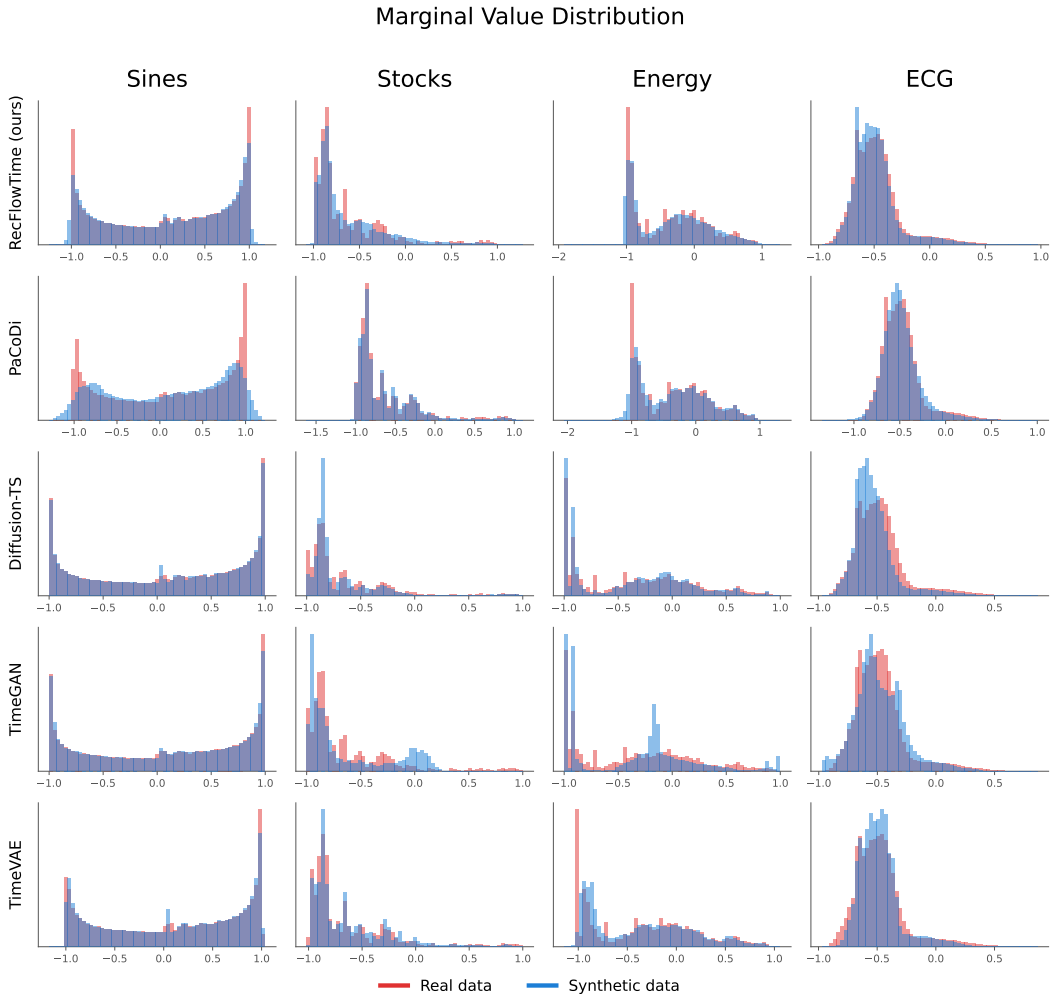

Figure 16: Marginal value distributions for real and generated sequences, pooled across time steps and features. Rows correspond to generation methods, and columns correspond to datasets. These distributions evaluate whether generated samples reproduce the range, frequency, and overall spread of observed values. However, they ignore temporal ordering; therefore, close agreement is necessary but not sufficient for temporal fidelity.

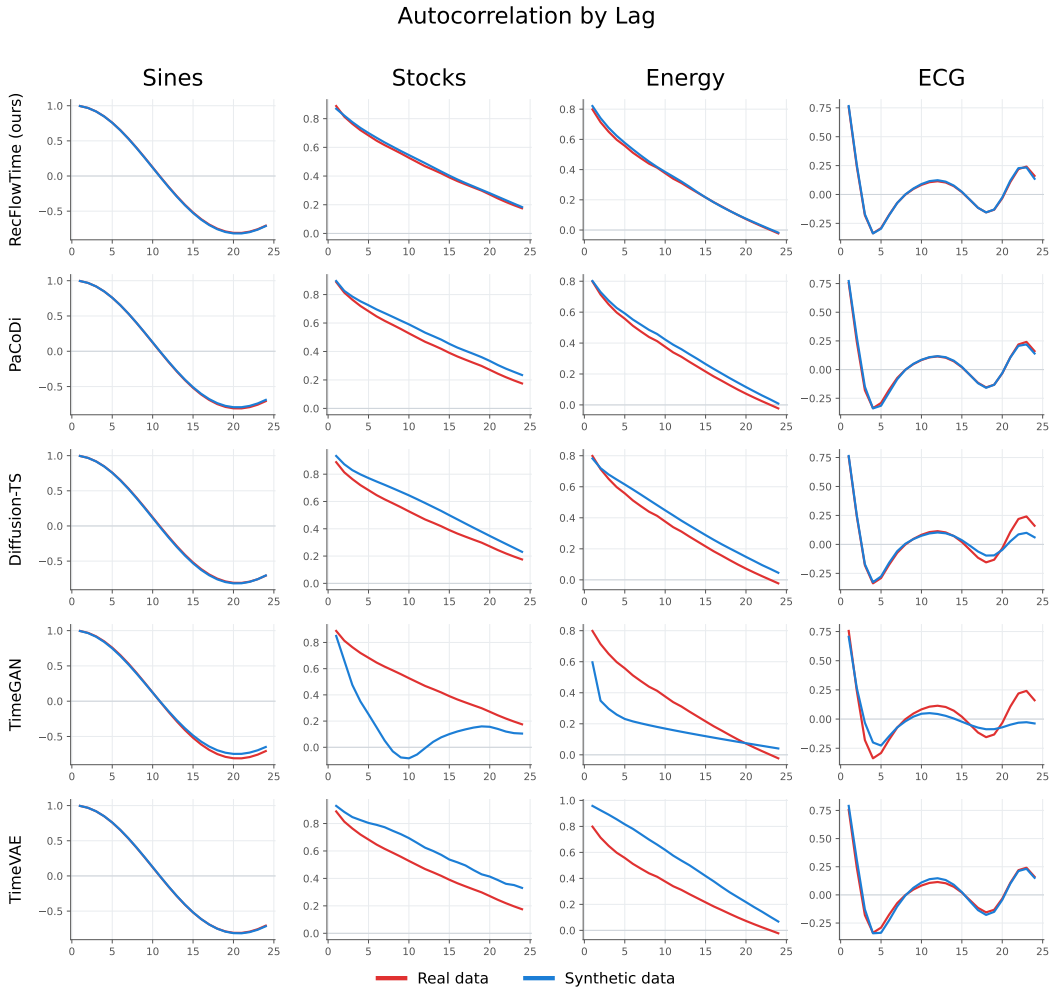

Figure 17: Autocorrelation as a function of temporal lag, averaged across sequences and features. Rows correspond to generation methods, and columns correspond to datasets. Autocorrelation measures how strongly values at one time step are related to values several steps later. Agreement between the real and generated curves therefore indicates that these lagged relationships are preserved. The autocorrelation distance in Table 2 summarizes the discrepancy between the curves.

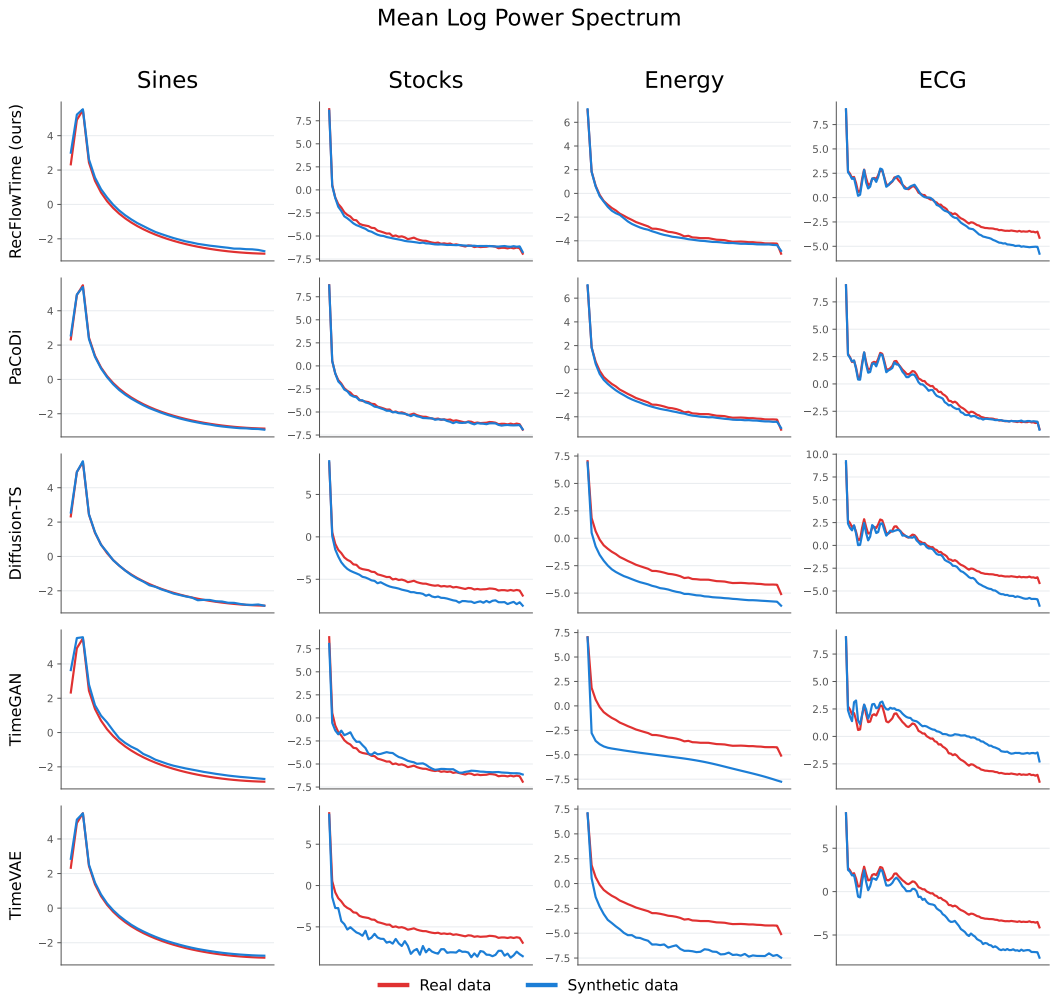

Figure 18: Mean log power spectra of real and generated sequences. Rows correspond to generation methods, and columns correspond to datasets. The spectra show how signal variation is distributed across temporal frequencies: aligned peaks indicate similar periodic behavior, while agreement at high frequencies indicates similar local smoothness. Excessively smooth samples fall below the real-data spectrum at high frequencies, whereas excessively rough samples rise above it. The spectral distance in Table 2 summarizes the discrepancy between the real and generated curves.

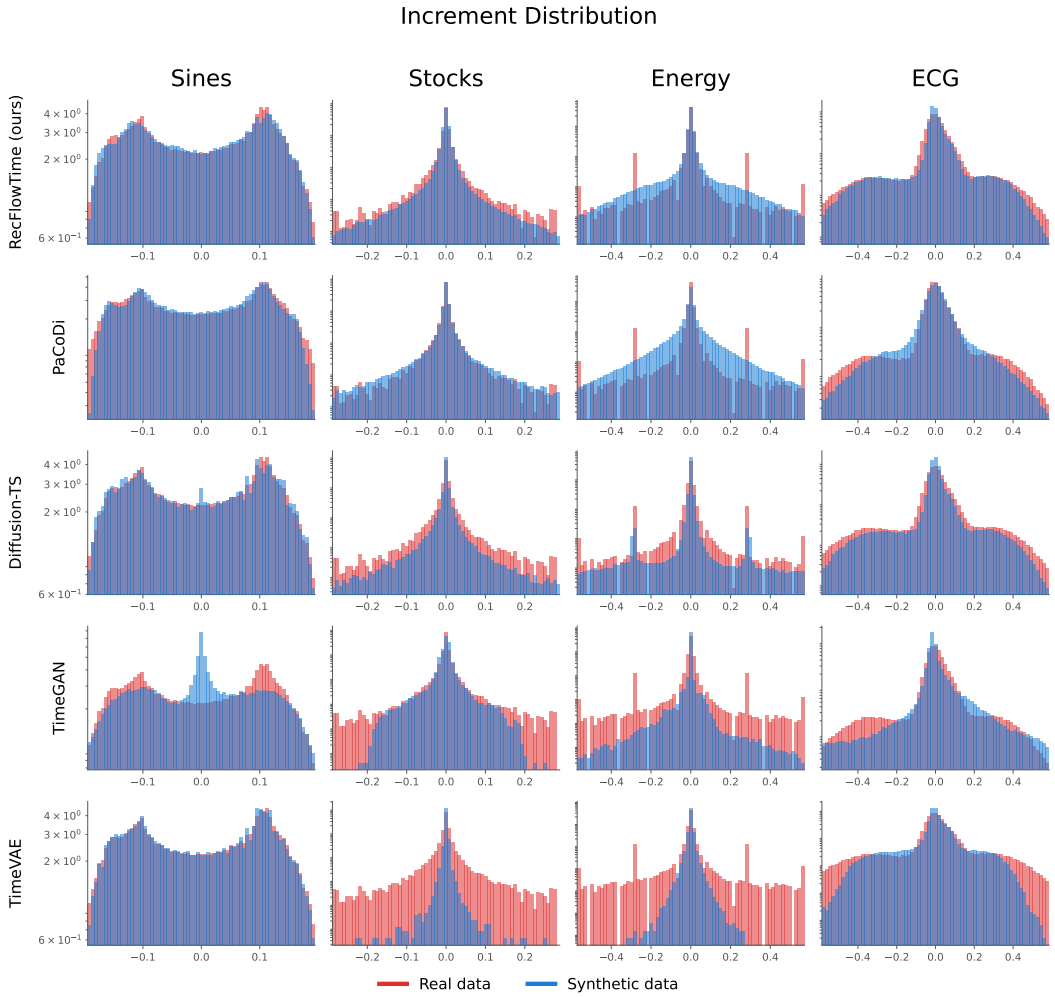

Figure 19: Distributions of signed step-to-step increments in real and generated sequences. Rows correspond to generation methods, with RecFlowTime shown first, and columns correspond to datasets. Real and generated histograms are overlaid within each panel: red indicates increments that occur more frequently in the real data, blue indicates increments produced more frequently by the model, and purple indicates overlap. This visualization reveals whether a model reproduces local temporal variation, including smooth transitions, volatility, and rare abrupt changes; models that are overly smooth concentrate too much mass near zero, whereas overly noisy models produce excessively large increments. The increment distance in Table 2 summarizes these discrepancies. Density is displayed on a logarithmic scale over the central 99.5% of the real-data range because the distributions are sharply concentrated.

I CONDITIONAL GENERATION

Although RecFlowTime is trained without conditioning information, its linear data–noise interpolation provides a direct way to enforce observations during sampling. We use this property for imputation and forecasting without retraining the model, introducing an auxiliary network, or increasing the number of function evaluations. The procedure guarantees exact agreement at observed entries, but it is an approximate conditioning method rather than an exact sampler from the conditional distribution.

I.1 MASKED REPLACEMENT

Let $\boldsymbol{y} \in \mathbb{R}^{T\times F}$ denote the target sequence, and let $m \in \{0,1\}^{T\times F}$ indicate which entries are observed. After each Euler update to time t, we replace the observed entries of the current sample with their values along the linear data–noise path:

$$\boldsymbol{x}_t \leftarrow m \odot \big[(1-t)\boldsymbol{y} + t\boldsymbol{\epsilon}\big] + (1-m)\odot \boldsymbol{x}_t, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{9}$$

where $\boldsymbol{\epsilon}$ is the Gaussian noise used to initialize the sample and is held fixed throughout the trajectory. Equation 9 modifies only the observed entries; the model continues to generate the hidden entries.

At $t = 0$, the replacement becomes

$$\boldsymbol{x}_0 \leftarrow m \odot \boldsymbol{y} + (1-m)\odot \boldsymbol{x}_0,$$

so the final sample matches every observation exactly. The maximum absolute error at observed entries is therefore zero in all evaluated configurations. Before the next model evaluation, we also replace the observed entries of the clean-sequence estimate used for self-conditioning with their values in $\boldsymbol{y}$. This ensures that the current sample and the self-conditioning input satisfy the same observations.

Masked replacement requires only elementwise arithmetic and therefore adds no network evaluations. Every conditional sample uses the same 20 evaluations as unconditional generation. By comparison, the gradient-guidance procedure used by Diffusion-TS (Yuan & Qiao, 2024) requires one to three additional evaluations per step, which would increase the total cost from 20 to between 40 and 80 evaluations.

I.2 EVALUATION PROTOCOL

We evaluate two tasks that differ in which entries are observed. For **imputation**, we hide contiguous segments whose lengths follow a geometric distribution with mean three. The total hidden fraction ranges from 0.1 to 0.9. Under *separate* masking, the missing segments are sampled independently for each feature. Under *concurrent* masking, the same time steps are hidden across all features. We use contiguous segments because isolated missing values can often be recovered through simple interpolation and therefore provide a less demanding evaluation.

For **forecasting**, we provide the initial portion of each sequence and hide the final

$$h \in \{T/8, T/4, T/2\}$$

time steps of every feature. These three settings evaluate short-, medium-, and long-horizon forecasting. Each task configuration contains 250 held-out sequences, and we generate ten conditional completions for every sequence. The quantitative metrics use these ten completions, whereas the visualization bands are constructed from sixteen completions.

Evaluation metrics. A conditional generator produces samples representing multiple plausible completions. In contrast, squared error is minimized by a single value: the conditional mean. Evaluating one randomly generated completion with squared error would therefore penalize valid predictive variation. Instead, we average the ten generated completions for each input and compute the mean squared error of this sample mean against the true hidden values.

We also report the continuous ranked probability score (CRPS), which evaluates both the accuracy and spread of the generated completions. For a hidden scalar value y, it is

$$\mathrm{CRPS} = \mathbb{E}|X - y| - \frac{1}{2}\mathbb{E}|X - X'|,$$

where X and X' are independent samples from the predicted conditional distribution. We estimate both expectations using the ten generated completions. The first term measures their distance from the true value, while the second accounts for their spread. Mean squared error and CRPS are computed only at hidden entries and averaged across sequences, time steps, and features. Lower values are better.

Deterministic baselines. For imputation, the baseline linearly interpolates between neighboring observed values. For forecasting, it repeats the final observed value throughout the forecast horizon. These simple baselines provide meaningful comparisons because they are strong when sequences behave locally like random walks. Under a random-walk model, linear interpolation gives the Brownian-bridge conditional mean for missing interior values, while the last observed value is the conditional mean of each future value. Because a deterministic prediction has no predictive spread, its CRPS reduces to its absolute error. We therefore report the mean absolute error of each baseline alongside the CRPS of RecFlowTime.

I.3 Results

Tables 12 and 13 report imputation and forecasting performance, respectively. Performance depends strongly on the dataset. On ECG, RecFlowTime improves CRPS over interpolation by $4.1\times$ at 10% hidden and $1.9\times$ at 90% hidden; for forecasting, it improves over last-value prediction by $6.1\times$ at $h = T/8$ and $2.7\times$ at $h = T/2$. Sines shows similar gains, including an $11.4\times$ forecasting improvement at $h = T/4$. These results indicate that the model can recover periodic and quasi-periodic structure beyond what local deterministic baselines capture. On Energy, it improves forecasting at every horizon but improves imputation only at low hidden fractions. On Stocks, the deterministic baselines are generally stronger, consistent with the data's random-walk-like behavior, for which interpolation and last-value prediction are difficult to outperform.

Figures 20 and 22 show the median completion and its 10th–90th percentile range. The bands remain narrow on Sines and ECG because their recurring patterns make missing values more predictable from the observed context. In contrast, the bands widen on Stocks and Energy as these datasets contain more irregular variation and therefore admit a broader range of plausible completions. These figures show selected examples, whereas Tables 12 and 13 report averages over all held-out sequences.

Figure 21 shows that RecFlowTime outperforms interpolation across all hidden fractions on ECG and up to approximately 85% on Sines, 40% on Energy, and 20% on Stocks. Performance declines as the hidden fraction increases because the missing intervals become more frequent and longer, leaving fewer neighboring observations from which to infer their values. Concurrent masking is particularly difficult because every feature is missing at the same time steps, preventing the model from using observed features to infer the missing ones.

Figure 23 shows a similar effect in forecasting: uncertainty and error increase as the observed prefix provides less information about increasingly distant future values. RecFlowTime remains better than last-value prediction across all evaluated horizons on Sines and ECG because their recurring structure supports extrapolation. On Stocks, however, the last observed value is a strong baseline for random-walk-like behavior, and RecFlowTime crosses it near $0.1T$. Energy lies between these regimes and remains competitive at the evaluated horizons.

Finally, Figure 24 shows that most improvement occurs within 20 Euler steps. Additional steps initially reduce discretization error in the generated hidden entries, but the benefit diminishes once the trajectory is sufficiently resolved. For example, Sines imputation improves by $3.1\times$ between 5 and 20 steps, whereas increasing the budget from 20 to 64 steps provides smaller gains. Thus, masked replacement retains the 20-step sampling budget used for unconditional generation.

Table 12: Imputation error on the hidden entries: 250 sequences, ten completions each, 20 Euler steps. Entries are hidden in geometric streaks of mean length three, independently per feature (*separate*) or at the same timesteps (*concurrent*). The upper panel is the squared error of the ten-sample mean, which estimates the conditional mean; the lower is CRPS against interpolation's mean absolute error, which is the CRPS of a deterministic prediction. Linear interpolation is a scale, not a competing method: on a near-random walk it is the posterior mean. Lower is better.

MSE of ten-sample mean						
Dataset	Configuration	0.1	0.3	0.5	0.7	0.9
Sines	RecFlowTime, separate	0.0015	0.0030	0.0058	0.0216	0.0313
	RecFlowTime, concurrent	0.0013	0.0033	0.0061	0.0215	0.0319
	linear interpolation	0.0308	0.0263	0.0248	0.0252	0.0267
Stocks	RecFlowTime, separate	0.0029	0.0034	0.0036	0.0072	0.0106
	RecFlowTime, concurrent	0.0031	0.0037	0.0050	0.0132	0.0181
	linear interpolation	0.0026	0.0030	0.0027	0.0026	0.0027
Energy	RecFlowTime, separate	0.0100	0.0125	0.0136	0.0180	0.0210
	RecFlowTime, concurrent	0.0122	0.0128	0.0138	0.0194	0.0232
	linear interpolation	0.0096	0.0097	0.0092	0.0090	0.0090
ECG	RecFlowTime, separate	0.0035	0.0049	0.0076	0.0155	0.0188
	RecFlowTime, concurrent	0.0047	0.0059	0.0093	0.0172	0.0203
	linear interpolation	0.0381	0.0385	0.0384	0.0388	0.0388
CRPS						
Dataset	Configuration	0.1	0.3	0.5	0.7	0.9
Sines	RecFlowTime, separate	0.0130	0.0169	0.0250	0.0512	0.0641
	RecFlowTime, concurrent	0.0132	0.0179	0.0261	0.0514	0.0652
	linear interpolation	0.0917	0.0865	0.0841	0.0839	0.0865
Stocks	RecFlowTime, separate	0.0102	0.0113	0.0143	0.0298	0.0386
	RecFlowTime, concurrent	0.0147	0.0167	0.0225	0.0428	0.0515
	linear interpolation	0.0140	0.0144	0.0138	0.0139	0.0138
Energy	RecFlowTime, separate	0.0277	0.0328	0.0373	0.0512	0.0585
	RecFlowTime, concurrent	0.0323	0.0339	0.0385	0.0544	0.0628
	linear interpolation	0.0321	0.0333	0.0328	0.0323	0.0324
ECG	RecFlowTime, separate	0.0247	0.0285	0.0347	0.0504	0.0564
	RecFlowTime, concurrent	0.0266	0.0303	0.0374	0.0525	0.0585
	linear interpolation	0.1051	0.1078	0.1067	0.1078	0.1081

Table 13: Forecasting the final h time steps of every feature using 250 sequences, ten completions per sequence, and 20 Euler steps. The deterministic baseline repeats the last observed value, which is the optimal mean forecast for a random walk. MSE is computed from the ten-sample mean, whereas CRPS evaluates the distribution of completions. Lower values are better.

		MSE of ten-sample mean			CRPS		
Dataset	Configuration	$h = T/8$	$h = T/4$	$h = T/2$	$h = T/8$	$h = T/4$	$h = T/2$
Sines	RecFlowTime	0.0064	0.0196	0.1649	0.0405	0.0675	0.1901
	last value	0.2403	0.8549	1.0812	0.3933	0.7702	0.8486
Stocks	RecFlowTime	0.0066	0.0114	0.0244	0.0281	0.0395	0.0629
	last value	0.0061	0.0051	0.0065	0.0267	0.0297	0.0388
Energy	RecFlowTime	0.0229	0.0273	0.0361	0.0554	0.0668	0.0851
	last value	0.0239	0.0275	0.0331	0.0624	0.0757	0.0911
ECG	RecFlowTime	0.0041	0.0144	0.0349	0.0254	0.0467	0.0788
	last value	0.0498	0.0803	0.0922	0.1556	0.2025	0.2094

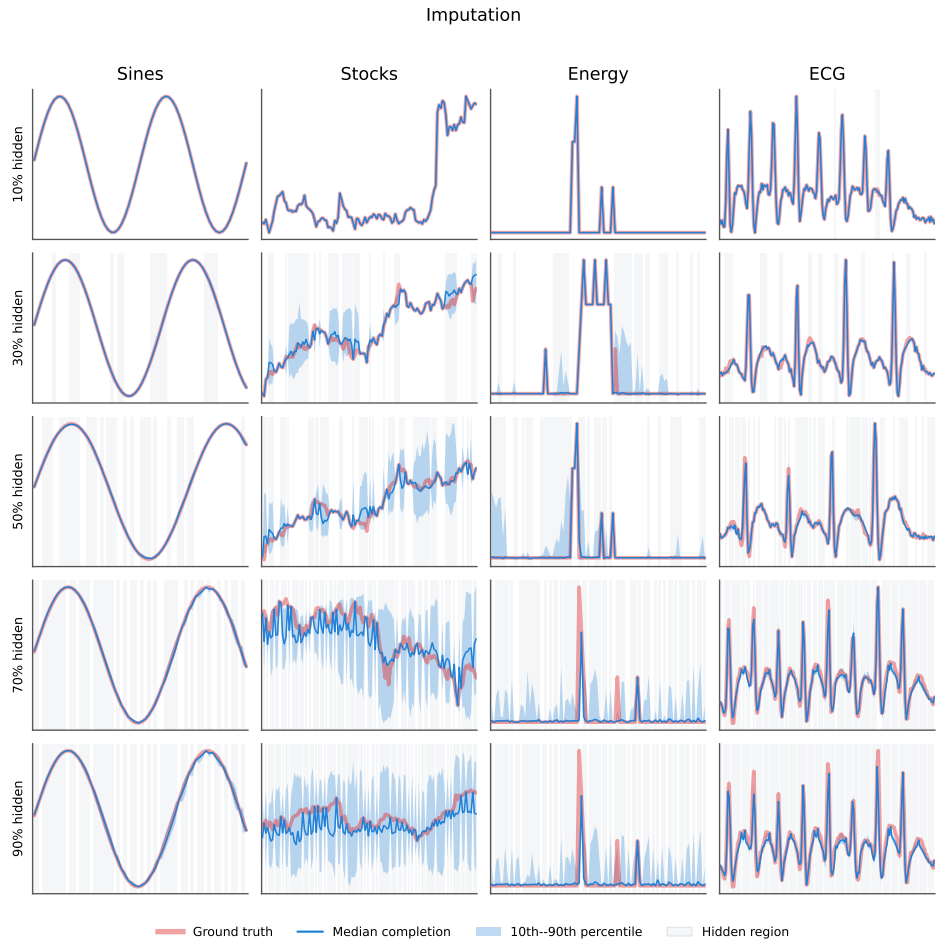

Figure 20: Imputation, all hidden fractions. Rows are the fraction hidden, columns datasets; shaded spans are hidden and one feature is shown per dataset. The band is the 10th to 90th percentile over sixteen completions of the same input and the line is their median, so each panel shows a predictive distribution rather than one draw. Averages over all sequences are in Table 12. The band widens with the hidden fraction on Stocks and Energy and stays narrow on Sines and ECG.

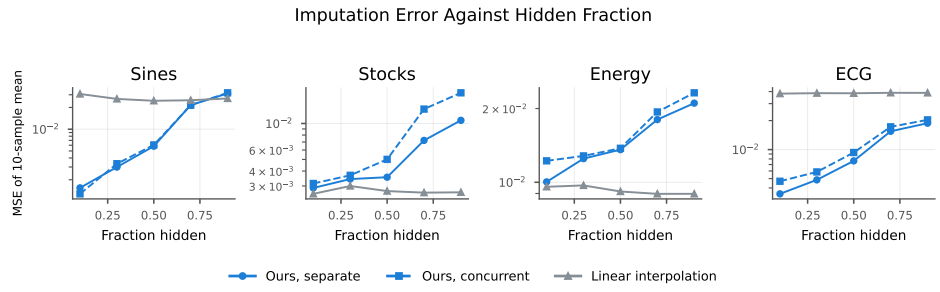

Figure 21: Imputation error against the fraction hidden, for both mask styles, with linear interpolation as a scale. The vertical axis is the squared error of the ten-sample mean, logarithmic.

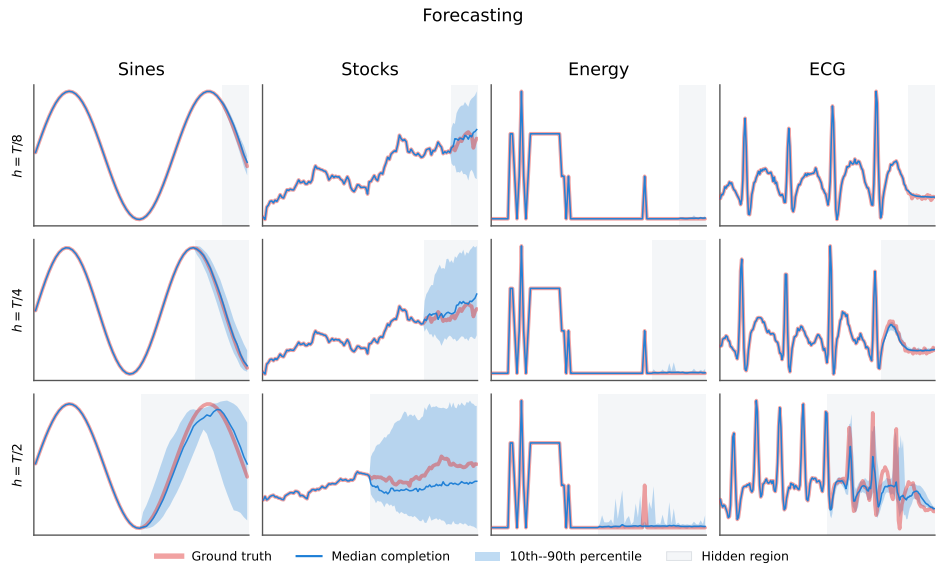

Figure 22: Forecasting, all horizons. Rows are the horizon as a fraction of T, which is 64, 128, 96 and 192 by column, so the absolute number of steps differs; the shaded span is the forecast region. Bands, medians and selection as in Figure 20, with averages in Table 13. The band opens with the horizon, most visibly on Stocks.

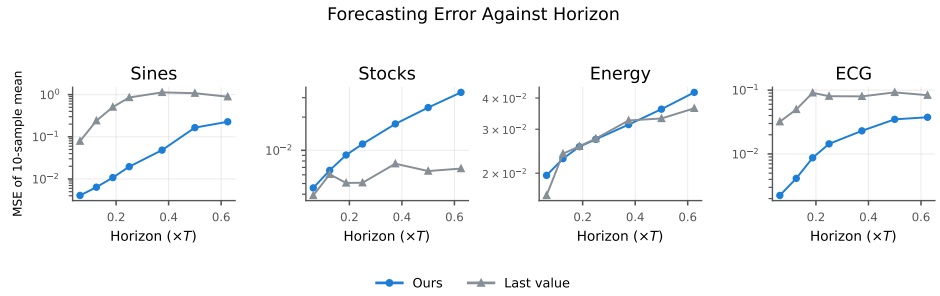

Figure 23: Forecasting error against the horizon, with repetition of the last observed value as a scale. The vertical axis is the squared error of the ten-sample mean, logarithmic, and the horizontal axis is the horizon as a fraction of T.

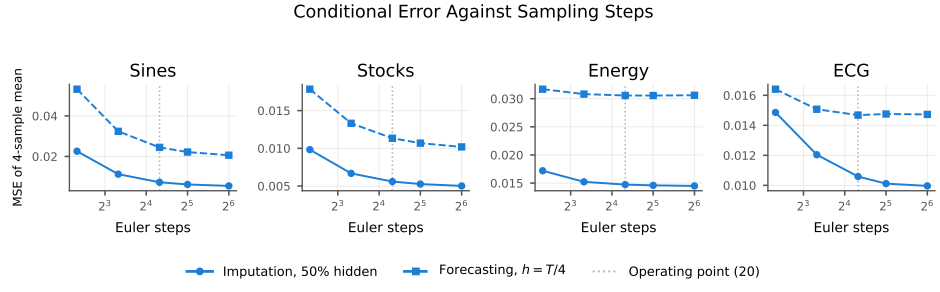

Figure 24: Conditional error against the number of Euler steps, at 50% hidden and $h = T/4$. The dotted line marks the 20-step operating point used elsewhere in the paper.