FAST MULTIVARIATE TIME SERIES GENERATION VIA SAMPLING FROM A SPHERICAL MANIFOLD

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a multivariate time series generator that embeds the training data onto a uniformly populated spherical manifold, from which new points are sampled and decoded into high-fidelity, diverse samples. The approach uses no attention mechanisms, adversarial training, diffusion, flow matching, or vector quantization, yet achieves generation quality competitive with the state of the art while sampling one to four orders of magnitude faster. We identify which ingredients are essential to the success of the approach in time series generation, and which are not. A controlled ablation shows that replacing the Gaussian prior of a variational autoencoder with the bounded sphere is what makes the approach competitive, and we find that uniform coverage of the sphere is critical to generation quality, which we enforce with a KoLeo regularizer. Conversely, iterative refinement and consistency regularization prove unnecessary. The model is also readily made class-conditional, and its samples are effective for data augmentation in long-term (128-step) human activity recognition. Our results suggest that a bounded spherical manifold is a natural fit for time series generation, and they motivate further research in this direction.

1 INTRODUCTION

Generative time series models serve a range of downstream tasks, such as data augmentation Lee et al. (2023); Huang et al. (2024), anomaly detection Li et al. (2019); Wang et al. (2023), data imputation Yoon et al. (2018); Tashiro et al. (2021), privacy-preserving data sharing Jordon et al. (2019); Tian et al. (2024), and stress testing or scenario simulation Huang et al. (2024); Wiese et al. (2020). Modern approaches can be grouped by their generative mechanism: generative adversarial networks (GANs) Yoon et al. (2019), diffusion models Yuan & Qiao (2024), flow matching Kim et al. (2026), discrete-token models Chen et al. (2024), and latent variable models Desai et al. (2021). Simple latent variable models, such as variational autoencoders (VAEs), have largely been abandoned as standalone generators in favor of diffusion and flow matching, most commonly on the grounds of limited sample fidelity Zhang et al. (2026). A less explored alternative is to replace the Gaussian prior of a VAE with a non-variational spherical manifold Zhao et al. (2019), a modification that has recently been shown to counteract common failure modes of VAEs in computer vision Yue et al. (2026); Man et al. (2026).

Diffusion and flow matching models attain their fidelity at the cost of many network evaluations per sample. This makes generation slow and limits applications that require large volumes of fresh samples, such as on-line data augmentation. We ask whether a single-pass latent variable model can match their quality if its latent space is designed appropriately, and show that it can: an autoencoder with plain MLP or LSTM backbones, whose latent space is a uniformly populated sphere, is competitive with self-reported state-of-the-art results on six standard benchmarks while generating samples one to four orders of magnitude faster. Notably, the ingredients that make this work for time series differ from those reported for images. Uniform coverage of the sphere, which we enforce with a KoLeo regularizer, is critical, whereas iterative refinement and consistency regularization, both central to the image-domain Sphere Encoder Yue et al. (2026), are not needed.

Our contributions are as follows.

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.

- We introduce, to our knowledge, the first non-variational spherical latent variable generator for multivariate time series, which generates a sample in a single forward pass (Section 3).
- We show that it is competitive with self-reported state-of-the-art results on six standard benchmarks, while generating samples one to four orders of magnitude faster (Section 5).
- Through ablations, we identify which design choices matter: replacing the Gaussian prior with the sphere and enforcing uniform coverage are essential, while iterative refinement and consistency regularization are not (Section 6).
- We extend the approach to class-conditional, long-term (128-step) generation on human activity recognition, where a classifier trained only on our synthetic samples comes within 3.2 percentage points of one trained on real data, and, when real data is scarce (221 samples), adding just 110 synthetic samples raises accuracy from 0.841 to 0.906 (Section 5.2).
- We critically examine the standard evaluation protocol: we extend its reference scores to empirical floors and ceilings for every metric, and show that it is largely blind to memorization, that re-run baselines tend to score worse than self-reported ones, and that the Predictive score is uninformative on two of its six datasets (Sections 4.3 and 5, Appendix E.1).

2 Related work

Time series generation. A time series generator aims to produce synthetic samples that are faithful to a reference data distribution and as diverse as it Räisä et al. (2025). *GAN*-based generators, popularized by TimeGAN Yoon et al. (2019) and followed by others Xu et al. (2020); Jeon et al. (2022); Jeha et al. (2022), train a generator against a discriminator, often in a learned embedding space and with added supervision to capture temporal dynamics. They produce samples in a single forward pass, but adversarial training is known to be unstable. *Diffusion* models, such as Diffusion-TS Yuan & Qiao (2024) and PaD-TS Li et al. (2025), instead learn to reverse a gradual noising process, and currently set the standard for sample quality, at the cost of many network evaluations per sample. *Flow matching* approaches, such as FlowTS Hu et al. (2024) and SMF Kim et al. (2026), learn a continuous transport from noise to data, reducing but not removing the number of sampling steps. *Discrete-token* approaches Lee et al. (2023); Chen et al. (2024) quantize time series into codebook tokens whose distribution is modeled by a transformer.

Latent variable models. VAEs Kingma & Welling (2013) generate by decoding samples from a prior, typically $\mathcal{N}(0, I)$, towards which the encoder's posterior is regularized with a Kullback–Leibler (KL) loss. This makes them fast and simple to sample from, but the regularization trades off against reconstruction: a posterior that does not match the prior leaves "holes" Aneja et al. (2021) that decode to unrealistic samples, while matching it too closely causes posterior collapse Wang et al. (2021). VAE-based time series generators therefore add structure to work around these issues, e.g. interpretable trend and seasonality components in the decoder (TimeVAE Desai et al. (2021)) or a Koopman-inspired latent prior (KoVAE Naiman et al. (2024b)).

Spherical latent spaces. Zhao et al. (2019) proposed a non-variational alternative to the VAE that replaces its Gaussian prior with a hypersphere, the *Spherical Auto-Encoder* (SAE), for image generation. The approach gained little traction until Yue et al. (2026) scaled it up with transformer-based backbones and pixel- and latent-consistency losses, and showed it to be competitive with modern image generators, particularly when combined with iterative refinement. It has since also proven effective for image restoration Man et al. (2026).

Our position. We adopt the spherical latent space for time series generation and find that its success here depends on different ingredients than in images. (i) Uniform coverage of the sphere does not come for free: whereas Yue et al. (2026) report that latent noise alone yields a near-uniform latent distribution, our encodings cluster without explicit KoLeo repulsion Sablayrolles et al. (2019), and coverage turns out to be the key driver of sample quality (Section 6.2). (ii) Conversely, much of the image recipe is unnecessary: with a plain MSE loss, MLP or LSTM backbones, and a single decoding pass, we are competitive with state-of-the-art time series generators, while consistency losses and iterative refinement do not help (Section 6.3). (iii) Unlike prior VAE-based time series generators, we do not add structure to compensate for the Gaussian prior but replace it, and show in a controlled ablation that this replacement, not the backbone, accounts for the gap (Section 6.1).

3 METHOD

3.1 TIME SERIES GENERATION

We define a time series, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, as a sequence of n ordered observations drawn from a distribution of interest, $\mathbf{X} \sim \mathcal{D}$. Since $\mathbf{X}$ is a time series, we expect dependence between observations, such as autocorrelation, trends, and oscillations, and, if $\mathbf{X}$ is multivariate, cross-correlation between its components.

Since we rarely have access to $\mathcal{D}$ directly, we build a generator G from a finite training set D_{train} sampled from $\mathcal{D}$. The goal is for a synthetic set D_{synth} sampled from G to be distributed like $\mathcal{D}$, i.e. faithful to it without merely reproducing D_{train} (see Section 4.3 for how we evaluate this). In the conditional setting, $\mathbf{X}$ comes paired with a condition y, e.g. a class label (Section 5.2).

3.2 BUILDING THE SPHERE

Our model is an autoencoder that reconstructs its input $\mathbf{X}$ through an intermediate projection onto a sphere:

$$\hat{\mathbf{X}} = D(M(E(\mathbf{X}))), \tag{1}$$

where E is an encoder, M is the spherical manifold projection, and D is a decoder. The projection normalizes each latent $\mathbf{Z} = E(\mathbf{X})$ by its root mean square,

$$\mathbf{V} = M(\mathbf{Z}) = \frac{\mathrm{vec}(\mathbf{Z})}{\sqrt{\mathrm{mean}(\mathbf{Z}^2)}}, \tag{2}$$

which places $\mathbf{V}$ on a hypersphere of radius $\sqrt{N}$, where N is the latent dimension. New samples are generated by decoding random points on this sphere:

$$G = D(M(\mathbf{Z}_{rand})), \qquad \mathbf{Z}_{rand} \sim \mathcal{N}(0, \mathbf{I}). \tag{3}$$

Since $\mathcal{N}(0, \mathbf{I})$ is isotropic, $M(\mathbf{Z}_{rand})$ is uniformly distributed on the sphere. For E and D, we use basic fully connected or recurrent networks (Appendix A). Because M only rescales each latent by a positive scalar[1], it discards its norm and nothing else. For the recurrent backbones, whose latent keeps a compressed time axis, this means that correlations and spectral shape along that axis are preserved.

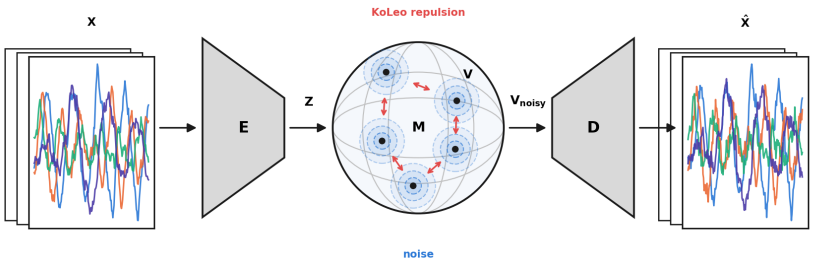


Figure 1: An encoder E maps the input $\mathbf{X}$ to a latent $\mathbf{Z}$, which the projection M places on a spherical manifold as $\mathbf{V}$, which is finally decoded by D into a reconstructed sample $\hat{\mathbf{X}}$. Latent noise ensures that nearby positions on the sphere are decoded to similar outputs, and *KoLeo* repulsion ensures that the sphere is uniformly populated during training.

3.3 POPULATING THE SPHERE

Since generation decodes uniformly distributed points on the sphere (Eq. 3), the encoded training data should cover the sphere uniformly as well. We train with the loss

$$L = L_{recon} + \beta_{\text{koleo}} L_{koleo}, \tag{4}$$

[1] In practice, we reshape the flattened vector back into its original shape after projection.

where L_{recon} is the point-wise MSE between $\mathbf{X}$ and $\hat{\mathbf{X}}$, and L_{koleo} is the ***KoLeo*** loss Sablayrolles et al. (2019), used in self-supervised pretraining, e.g. DINOv2 Oquab et al. (2024), to spread out embeddings:

$$L_{koleo} = -\frac{1}{b}\sum_{i=1}^{b} \log \min_{j \neq i} \|\mathbf{V}_i - \mathbf{V}_j\|, \tag{5}$$

where b is the batch size and $\mathbf{V}_i = M(E(\mathbf{X}_i))$[2]. Minimizing this loss pushes each training point away from its nearest neighbor on the sphere with a force inversely proportional to their distance. Empirically, uniform coverage is critical for generating high-quality samples through Eq. 3 (Section 6.2).

Finally, we inject latent noise during training, so that the decoder learns to produce consistent outputs for nearby points on the sphere:

$$\mathbf{V}_{noisy} = M(\mathbf{V} + i\,\boldsymbol{\Delta}), \qquad \boldsymbol{\Delta} \sim \mathcal{N}(0, \mathbf{I}), \qquad i = u_1 u_2,\ u_1 \sim \mathcal{U}(0, 0.5),\ u_2 \sim \mathcal{U}(0, \sigma_{max}), \tag{6}$$

where $\boldsymbol{\Delta}$ has the shape of $\mathbf{V}$ and sets the direction of the perturbation, and i is one scalar per sample that sets its intensity. As the product of two uniform variables, i favors small intensities. See Fig. 1 for an illustration of the approach as a whole.

4 Experimental Setup

Following Yuan & Qiao (2024); Kim et al. (2026); Zhang et al. (2026) and several other works, we evaluate our models on six datasets using four standard metrics to show that they are competitive in unconditional multivariate time series generation. We then perform targeted experiments on a seventh dataset to demonstrate their capabilities for conditional and long-term time series generation. All code will be made available on GitHub upon acceptance.

4.1 Datasets

Our core set of six multivariate time series benchmark datasets consists of *Sines*, *Stocks*, and *Energy* from Yoon et al. (2019), *MuJoCo* from Jeon et al. (2022), and *ETTh(1)* and *fMRI*, added by Yuan & Qiao (2024). Our seventh dataset, *HAR* Anguita et al. (2013), is class-labeled and is used for conditional generation. All datasets consist of 24-timestep windows, except *HAR*, which has 128 timesteps. See Appendix C for details.

4.2 Hyperparameters

Backbones. For Sines, Stocks, ETTh(1), and MuJoCo we use simple MLP-based encoders and decoders. For Energy, fMRI, and HAR we use LSTM-based encoders and decoders. See Appendix A for details on these backbones.

Training. Models are trained with Adam for 2000 epochs at a batch size of 512, with a learning rate of 10^{-3} that is cosine annealed to 10^{-6} at epoch 2000. L_{koleo} is weighted by $\beta_{\text{koleo}} = 0.01$ in Eq. 4 with no warm-up, and $\sigma_{max} = 0.25$. Both β_{koleo} and σ_{max} are shared by all datasets.

Tuning. We tune the training setup on Sines and apply it to the other datasets, with the latent size (i.e. the sphere dimensionality) swept per dataset. Two choices came from other datasets and were then adopted where needed: the cosine learning rate schedule (from MuJoCo, adopted for all datasets) and the LSTM backbones (from Energy, also used for fMRI and HAR). See Appendix B for details on our tuning protocol.

4.3 Evaluation

Our main result on unconditional multivariate time series generation uses the same evaluation suite as Yuan & Qiao (2024), which has since been adopted by others Hu et al. (2024); Kim et al. (2026);

[2]For efficiency, we select the nearest neighbor by cosine similarity and compute the loss with the Euclidean distance.

Zhang et al. (2026). Given a training set D_{train} and a generator G producing D_{synth}, the suite measures how closely D_{synth} matches D_{train} through four metrics (lower is better for all four):

- **Discriminative score**: $|\text{accuracy} - 0.5|$ of a simple gated recurrent unit (GRU) classifier trained to distinguish D_{synth} from D_{train} on a held-out split. A score of 0 means the classifier cannot tell the two apart.
- **Predictive score**: Train-on-synthetic, test-on-real: a simple GRU predictor is trained on D_{synth} to predict next-step values, and its mean absolute error is reported on D_{train}.
- **Context-FID**: Fréchet distance between real and synthetic sequence embeddings, computed with a TS2Vec encoder trained on D_{train}.
- **Correlational score**: Absolute difference between the pairwise feature-correlation matrices of D_{train} and D_{synth}, summed over feature pairs.

Each metric is reported as mean $\pm$ standard deviation over repeated evaluations. To keep the comparison conservative, we compare only against self-reported scores, which tend to be better than those obtained by re-running a method. For example, Li et al. (2025) and Kim et al. (2026) report Diffusion-TS's Yuan & Qiao (2024) discriminative score on Stocks as 0.082 and 0.085, compared with the self-reported 0.067; across all overlapping cells in these two papers, the re-run score is worse in 28 of 40 cases. See Appendix D for more detail on the evaluation suite and our chosen baselines.

5 Performance

In Fig. 2 we compare the performance of our method with self-reported baseline scores along with empirical estimates of performance ceiling (bad) and floor (good), defined as follows:

- *real-vs-noise ceiling*: the score of a 'generator' that outputs i.i.d. uniform noise over the data's $[0, 1]$ range, i.e. a model that has learned nothing about the data.
- *real-vs-real 50/50 floor*: the score obtained by comparing a random half of the real train set with the other half. This is a pessimistic estimate of the best possible performance, since each side has only half the data to train and estimate from.
- *real-vs-real floor*: the score obtained by comparing the whole train set with a fresh, same-size sample drawn from the true data-generating process. This score is averaged over 10 fresh draws with 10 evaluations each. We can only estimate this score for the two datasets for which the data generating process is known, i.e. *Sines* and *MuJoCo*.

Reference points of this kind have precedent: since Yoon et al. (2019), several works report an 'Original' predictive score, obtained by training the predictor on real rather than synthetic data. We extend this idea to all four metrics and add a ceiling, so that every score can be read relative to both a best- and a worst-case reference.

Most methods score close to or below the estimated floors (Fig. 2). It is difficult to identify a clear front-runner, except perhaps SDformer Chen et al. (2024), which uses far more parameters (see Appendix D.3) and reports no Correlational scores. Several methods, including ours on the Correlational score, score below the real-vs-real (fresh) floor, i.e. better than a generator with oracle access to the data-generating process. This suggests some overfitting to the canonical train sets, consistent with the convention of tuning on the evaluation suite itself (Appendix B).

These results demonstrate that our approach performs competitively with current time series generation approaches. It scores worst of all methods in only 5 of the 24 cells, and even there remains closer to the floor than to the ceiling on a log scale. In comparison, Diffusion-TS Yuan & Qiao (2024), arguably the most widely used baseline, scores worst in 7 of the 24 cells.

Finally, we highlight the Predictive score, on which most methods nearly tie. In the reference implementation Yoon et al. (2019), the predictor forecasts only the *last* channel one step ahead from the other channels. On Sines, whose channels are independent sine waves, the last channel cannot be predicted from the others beyond its average trajectory, however good the generator; current methods appear to have reached this limit. On Energy, the last channel is uniform random noise,

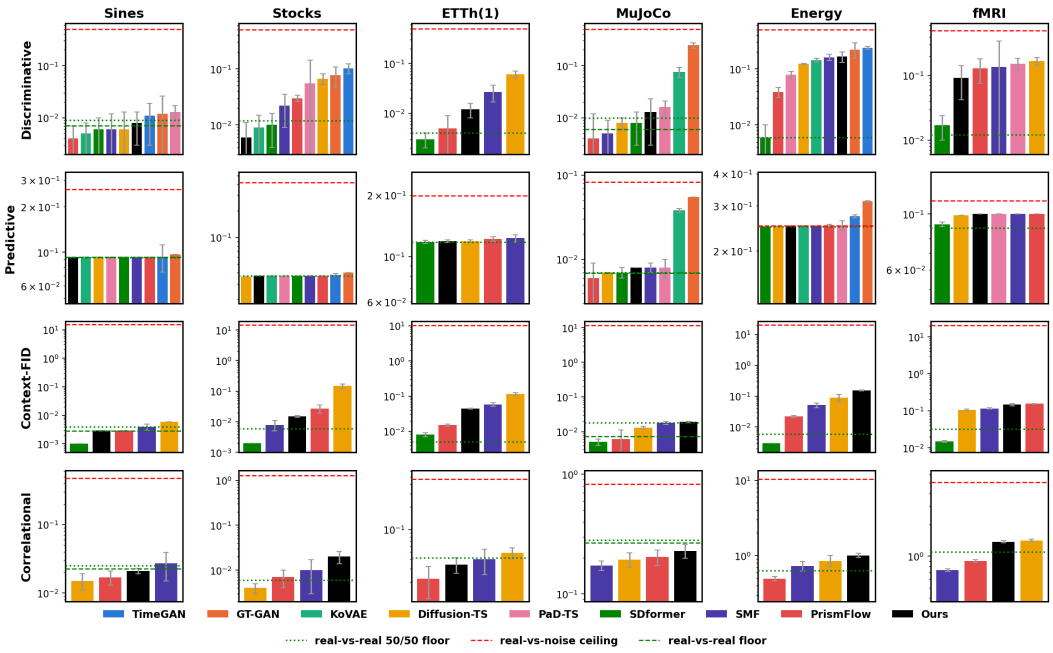


Figure 2: Unconditional time series generation performance (lower is better). Each row represents an evaluation score. Each column represents a dataset. The colored bars correspond to self-reported scores in the literature. Within each plot methods are sorted from best (lower score) to worst. See Appendix E.3 for actual numbers and an expanded analysis of out-of-sample performance on Sines and MuJoCo in Appendix E.1.

so no model can do better than chance, which is why its floor and ceiling coincide. The Predictive score is therefore uninformative on these two datasets. To our knowledge, this has gone unnoticed: TimeGAN Yoon et al. (2019) and Diffusion-TS Yuan & Qiao (2024) both describe the predictor as forecasting the full next-step vector, while their reference code forecasts only the last channel, and all baselines in Fig. 2 continue to report the Predictive score on both datasets.

5.1 Efficiency

A distinct advantage of our approach is the speed with which samples can be generated. In Fig. 3 we compare generation times against the baselines that self-report them. Our method is one to four orders of magnitude faster ($12\times$ to $7700\times$), despite running on less powerful hardware and generating the entire training set rather than 1000–2000 samples. This increase in generation speed makes "on-line" augmentation practical, i.e. generating fresh synthetic samples throughout downstream training; we present preliminary experiments on this in Section 5.2.

5.2 Conditional long-term generation

The main evaluations use short-term (24 timesteps) unconditional generation, as that is the literature convention for comparing time series generators. Here we evaluate our approach on long-term (128 timesteps) class-conditional generation on HAR Anguita et al. (2013) (7352 train and 2947 test windows). To make our models conditional we apply FiLM Perez et al. (2018) conditioning to the inputs of both the encoder and the decoder, each with its own set of learned class embeddings. We assess the generated samples by how useful they are for training a downstream classifier, evaluated on the real test set, in the spirit of the Classification Accuracy Score (CAS) Ravuri & Vinyals (2019). The classifier is a small three-block 1D CNN, see Appendix A for details.

First, we fit our generator to the full HAR train set. We then train classifiers on the train set with an increasing portion of its samples either removed (Fig. 4, left, blue) or replaced by class-conditional synthetic samples with the same labels, drawn anew every epoch (black). Up to 96% of the train set

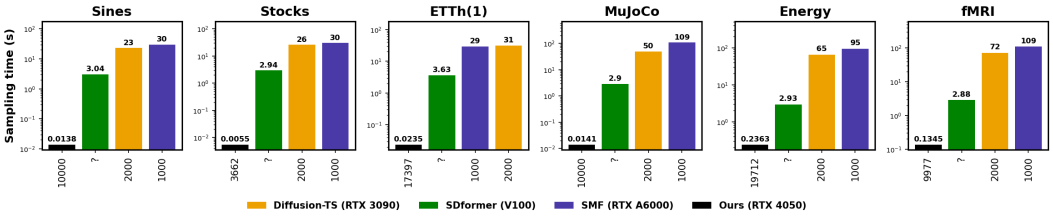


Figure 3: Wall-clock generation time per dataset (log scale). The label under each bar is the number of samples generated ("?": not reported). Baseline times and hardware are self-reported; ours are the mean of 10 runs on a laptop RTX 4050 GPU. SDformer Chen et al. (2024) does not report its sample count; its re-measured Diffusion-TS times are of the same order as those Diffusion-TS Yuan & Qiao (2024) reports.

can be removed (294 samples remaining) with only a small drop in accuracy (0.918 versus 0.945). Beyond that point accuracy falls sharply, reaching chance level once no real samples remain. When the removed samples are instead replaced by synthetic ones, accuracy is maintained throughout. A classifier trained entirely on synthetic samples (black star), i.e. the CAS proper, comes within 3.2 percentage points of one trained on the full real train set (blue star).

Second, we consider a low-data regime (Fig. 4, right). We train both the generator and the classifier on the same fixed 3% subset of the train set (221 samples), so no further real data is used, and then *add* an increasing number of generated samples to the classifier's train set, either generated once (static) or drawn anew every epoch (on-line). Adding only 110 generated samples raises accuracy from 0.841 to 0.906, after which it saturates at about 0.90–0.91, within roughly 3.5 percentage points of the full train set. Static and on-line augmentation perform equivalently.

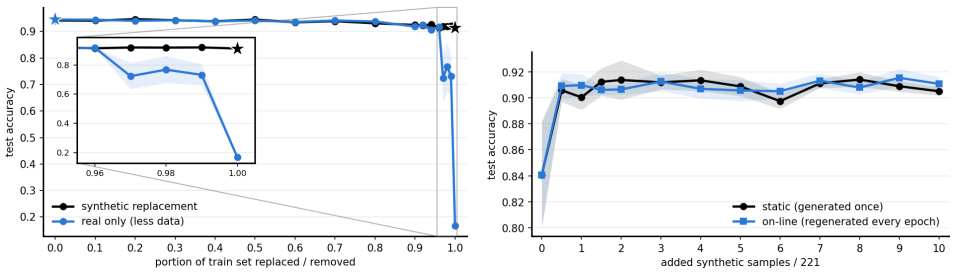


Figure 4: Real test-set accuracy of HAR classifiers trained with synthetic samples from our conditional generator. Left: a portion of the train set is removed (blue) or replaced by synthetic samples (black); stars mark the full real train set and the fully synthetic train set (CAS). The right-most blue point (no real samples) is chance level ($1/6$), not a trained classifier. Right: the fixed 221-sample real subset plus an increasing number of synthetic samples, in multiples of 221. Accuracy is the best test accuracy over 100 training epochs, applied identically to all conditions. Mean $\pm$ standard deviation over three (left) and five (right) classifier initialization seeds.

6 Ablations

6.1 Gaussian vs. spherical latent space

To isolate the effect of the spherical latent space, we replace it with a standard Gaussian (VAE) latent, i.e. reparameterization and a KL penalty towards $\mathcal{N}(0, I)$ instead of the spherical projection with noise and KoLeo regularization, keeping the backbone and training setup of our Sines and Stocks models unchanged. We tune the KL weight over four orders of magnitude and its warmup schedule. Performance generally improves as the KL weight decreases, but even the best setting, with the KL term nearly switched off, falls far short of the spherical model on Discriminative and Context-FID (Fig. 5). This holds even though we compare against the worst of our five training seeds, on two of

the simplest benchmark datasets, where the VAE should have its best chance. Predictive scores are tied, and on Stocks the VAE's Correlational score is slightly better.

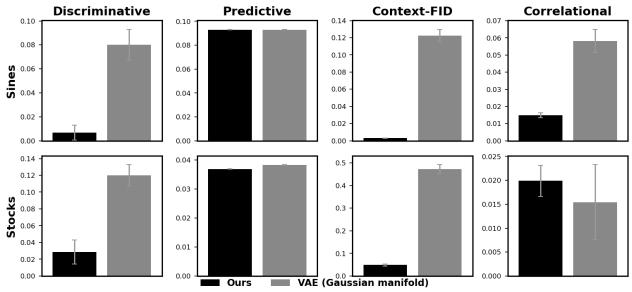


Figure 5: Our model (worst of five training seeds) versus the same model with a Gaussian (VAE) latent space (best of the tuned KL settings). Mean ± standard deviation over 5 evaluation repeats; lower is better for all metrics.

6.2 Sphere uniformity

Fig. 6 shows the effect of uniform coverage on Sines by varying the weight of the KoLeo regularizer, β_{koleo}. Without it, the encodings cluster together (short nearest-neighbour distances and a low intrinsic dimension), and uniformly drawn points land far from any encoded training sample. Increasing β_{koleo} spreads the encodings towards a uniform distribution, until uniform draws lie as close to the encoded data as to each other, and improves the Discriminative and Context-FID scores by up to 75% and 45%, respectively. The scores improve mostly over the same range of β_{koleo} in which uniform draws move closer to the data, suggesting that what matters most is that the prior covers the data.

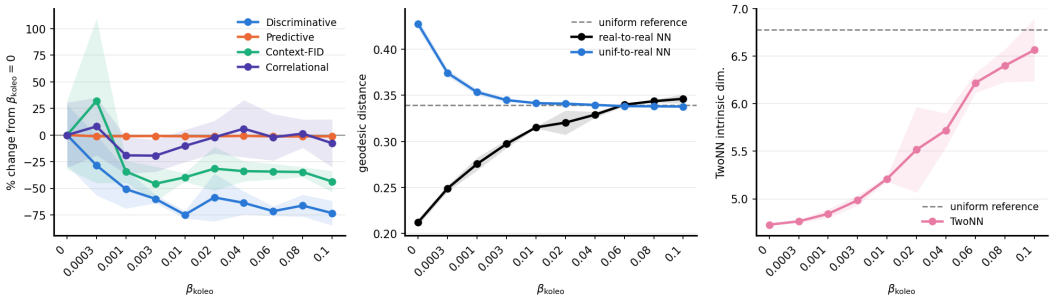


Figure 6: Effect of the KoLeo weight β_{koleo} on Sines. Left: change in each evaluation score relative to $\beta_{\text{koleo}} = 0$ (lower is better). Middle: median geodesic nearest-neighbour distance among encoded training samples (black), and from uniformly drawn sphere points to the nearest encoded training sample (blue). Right: TwoNN intrinsic dimension Facco et al. (2017) of the encoded training samples. Dashed lines show the values for a uniform distribution on the same sphere. Mean ± standard deviation over 3 training seeds. These models use an earlier architecture and training setup than our final Sines model, so the sweep illustrates the general effect of uniformity rather than tuning our final model.

6.3 Iterative refinement and consistency regularization

Yue et al. (2026) report that the quality of their generated images improves with *iterative refinement*: a generated sample is passed back through the encoder and decoded again, which is intended to move it progressively closer to the data manifold. We apply the same encode–project–decode cycle to our models and evaluate the generated samples after 0, 5, 10, 15 and 20 refinement steps (Fig. 7, top). Unlike Yue et al. (2026), we find no consistent benefit: refinement improves Context-FID and Correlational scores on Energy and, over the first few steps, Context-FID on ETTh(1), but

mostly degrades scores on the other four datasets. This holds even when we add the pixel and latent consistency losses of Yue et al. (2026), which are designed to support iterative refinement, and tune their weights and σ_{max} in our setting. Nevertheless, refinement does move generated samples closer to real samples on average, as measured by the geodesic distance between their encodings on the sphere (Fig. 7, bottom), on every dataset except fMRI, although a substantial share of individual samples move further away. See Appendix E.2 for the full results.

A key difference from Yue et al. (2026) is that we train with KoLeo repulsion. Unrefined samples drawn uniformly on our sphere already lie about as close to real encodings as uniformly drawn points lie to each other, suggesting that the real encodings already cover the sphere well, which leaves little gap for refinement to close. Alternatively, the small movement towards real samples may pull some samples into densely populated regions of the sphere without converging on any specific real sample, resulting in oversmoothed, "average" samples upon decoding. We leave disentangling these explanations to future work.

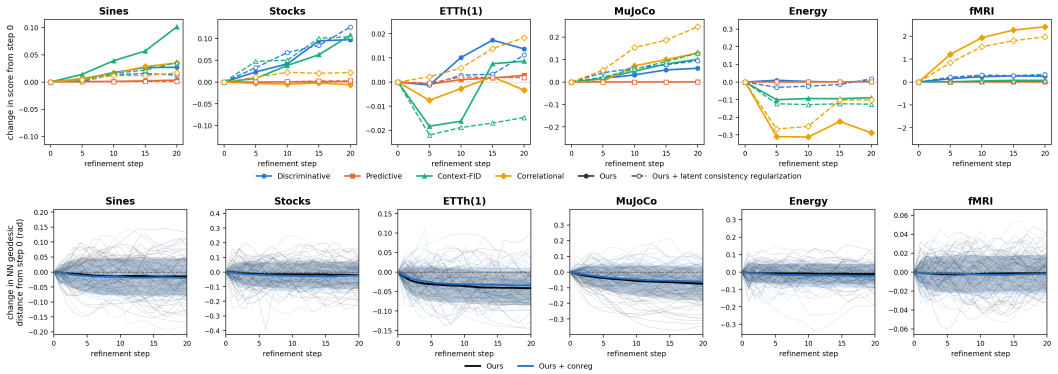


Figure 7: Effect of iterative refinement, for the best seed of our model (solid) and of our model with consistency regularization (dashed). Top: change in each evaluation score from its unrefined (step 0) value, in that metric's own units, evaluated every 5 steps. Lower is better for all metrics, so values below zero indicate improvement. Bottom: change from step 0 in the geodesic distance between each generated sample's latent and the nearest real sample's encoding on the sphere, for 1000 generated samples (mean ± 1 standard deviation; faint lines show 40 individual samples).

7 Conclusion

In this paper, we have proposed a simple latent variable approach to multivariate time series generation: an autoencoder with basic MLP or LSTM backbones whose latent space is a sphere, populated uniformly through KoLeo repulsion and smoothed with latent noise, so that new samples are generated by decoding points drawn uniformly from the sphere. The same model is readily made class-conditional through FiLM conditioning, which we apply to long-term generation and data augmentation on HAR. Our experiments show that, without attention, adversarial training, diffusion, or flow matching, this approach is competitive with self-reported state-of-the-art results while generating samples one to four orders of magnitude faster. Our ablations on Sines and Stocks indicate that the bounded spherical latent and its uniform coverage are what make this work where a Gaussian VAE fails, while iterative refinement and consistency regularization are not needed. We also extend the standard evaluation suite with empirical floors and ceilings, and show that its Predictive score is uninformative on two of its six datasets. One notable limitation, shared with the methods we compare against, is that this suite scores generators against their own training data and is largely blind to memorization; our out-of-sample results on Sines and MuJoCo show that tuning on it did not compromise generalization, but cannot rule out memorization, so more principled evaluation is needed. Future work includes enforcing temporal structure in the latent space more directly, e.g. through contrastive objectives, characterizing how much data the approach requires, and extending it to data that is itself spherical.

AI USE STATEMENT

In this work, we used generative AI tools for retrieval and discovery, to aid or polish writing, and code execution. We have not used generative AI tools for generating sections, generating synthetic data (apart from the need for it in the subject matter of this work), or any mathematical claims. We have reviewed all AI-assisted work (including all generated code) and take full responsibility for the final content of this work. None of the main intellectual contributions of this work were AI generated as we found it much more useful for execution than ideation.

REFERENCES

Jyoti Aneja, Alex Schwing, Jan Kautz, and Arash Vahdat. A contrastive learning approach for training variational autoencoder priors. In *Advances in Neural Information Processing Systems*, volume 34, pp. 480–493, 2021.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2013.

Zhicheng Chen, Shibo Feng, Zhong Zhang, Xi Xiao, Xingyu Gao, and Peilin Zhao. SDformer: Similarity-driven discrete transformer for time series generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 132179–132207, 2024.

Andrea Coletta, Sriram Gopalakrishnan, Daniel Borrajo, and Svitlana Vyetrenko. On the constrained time-series generation problem. In *Advances in Neural Information Processing Systems*, volume 36, pp. 61048–61059, 2023.

Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. TimeVAE: A variational auto-encoder for multivariate time series generation. *arXiv preprint arXiv:2111.08095*, 2021.

Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, 7(1):12140, 2017.

Shibo Feng, Zhicheng Chen, Xi Xiao, Zhong Zhang, Qing Li, Xingyu Gao, and Peilin Zhao. MSDformer: Multi-scale discrete transformer for time series generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

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.

Hongbin Huang, Minghua Chen, and Xiao Qiao. Generative learning for financial time series with irregular and scale-invariant patterns. In *International Conference on Learning Representations*, 2024.

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.

Jinsung Jeon, Jeonghak Kim, Haryong Song, Seunghyeon Cho, and Noseong Park. GT-GAN: General purpose time series synthesis with generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 35, pp. 36999–37010, 2022.

James Jordon, Jinsung Yoon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

Jinwoo Kim, Max Beier, Petar Bevanda, Nayun Kim, and Seunghoon Hong. Sequence modeling with spectral mean flows. In *Advances in Neural Information Processing Systems*, volume 38, pp. 132724–132760, 2026.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. In *International Conference on Artificial Intelligence and Statistics*, pp. 7665–7693, 2023.

Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, pp. 703–716, 2019.

Yang Li, Han Meng, Zhenyu Bi, Ingolv T. Urnes, and Haipeng Chen. Population aware diffusion for time series generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18520–18529, 2025.

Sean Man, Ron Raphaeli, Matan Kleiner, and Or Ronai. SP^3: Spherical priors for plug-and-play restoration. *arXiv preprint arXiv:2606.16396*, 2026.

Ilan Naiman, Nimrod Berman, Itai Pemper, Idan Arbiv, Gal Fadlon, and Omri Azencot. Utilizing image transforms and diffusion models for generative modeling of short and long time series. In *Advances in Neural Information Processing Systems*, volume 37, pp. 121699–121730, 2024a.

Ilan Naiman, N. Benjamin Erichson, Pu Ren, Michael W. Mahoney, and Omri Azencot. Generative modeling of regular and irregular time series data via Koopman VAEs. In *International Conference on Learning Representations*, 2024b.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ossi Räisä, Boris van Breugel, and Mihaela van der Schaar. Position: All current generative fidelity and diversity metrics are flawed. *arXiv preprint arXiv:2505.22450*, 2025.

Suman Ravuri and Oriol Vinyals. Classification accuracy score for conditional generative models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. In *International Conference on Learning Representations*, 2019.

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.

Muhang Tian, Bernie Chen, Allan Guo, Shiyi Jiang, and Anru R. Zhang. Reliable generation of privacy-preserving synthetic electronic health record time series via diffusion models. *Journal of the American Medical Informatics Association*, 31(11):2529–2539, 2024.

Chengsen Wang, Zirui Zhuang, Qi Qi, Jingyu Wang, Xingyu Wang, Haifeng Sun, and Jianxin Liao. Drift doesn't matter: Dynamic decomposition with diffusion reconstruction for unstable multivariate time series anomaly detection. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10758–10774, 2023.

Yixin Wang, David Blei, and John P. Cunningham. Posterior collapse and latent variable non-identifiability. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5443–5455, 2021.

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

Tianlin Xu, Li Kevin Wenliang, Michael Munn, and Beatrice Acciaio. COT-GAN: Generating sequential data via causal optimal transport. In *Advances in Neural Information Processing Systems*, volume 33, pp. 8798–8809, 2020.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. GAIN: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning*, pp. 5689–5698, 2018.

Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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

Kaiyu Yue, Menglin Jia, Ji Hou, and Tom Goldstein. Image generation with a sphere encoder. *arXiv preprint arXiv:2602.15030*, 2026.

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.

Junru Zhang, Lang Feng, Jinbo Wang, Xu Guo, Yucheng Wang, Han Yu, Min Wu, Yabo Dong, and Duanqing Xu. PrismFlow: Residual dynamics for flow matching in time-series generation. *arXiv preprint arXiv:2605.28867*, 2026.

Deli Zhao, Jiapeng Zhu, and Bo Zhang. Latent variables on spheres for autoencoders in high dimensions. *arXiv preprint arXiv:1912.10233*, 2019.

A BACKBONES

During training, the model performs the following operations. The raw input $\mathbf{X} \in \mathbb{R}^{b \times t \times c}$ goes through the encoder to produce a latent representation $\mathbf{Z} \in \mathbb{R}^{b \times t_h \times c_h}$. This is then projected onto the sphere to produce a projected latent representation $\mathbf{V} \in \mathbb{R}^{b \times t_h \times c_h}$. The decoder receives this and decodes it back to a reconstructed input $\hat{\mathbf{X}} \in \mathbb{R}^{b \times t \times c}$. In the above b, t, and c are the sizes of the batch, time, and channel indices, respectively.

The **MLP encoder** flattens $\mathbf{X}$ across the last two axes and passes it through three fully connected layers with 512 nodes each. Each of these layers is followed by a one-dimensional batch-norm layer and a $ReLU$ activation layer. A final fully connected linear layer produces $\mathbf{Z}$ by unflattening its output. There is no output activation and no additional regularizer (dropout or otherwise).

The **MLP Decoder** is identical to the MLP encoder, but it has seven hidden layers, takes $\mathbf{V}$ as input, and outputs $\hat{\mathbf{X}}$.

The **LSTM encoder** applies a basic LSTM (three layers of 256 nodes) to $\mathbf{X}$ to produce $\mathbf{H} \in \mathbb{R}^{b \times t \times h}$. A learned linear projection is used to project $\mathbf{H}$ to $\mathbf{Z}' \in \mathbb{R}^{b \times t \times c_h}$. In order to downsample the time axis we linearly interpolate the second axis to produce $\mathbf{Z} \in \mathbb{R}^{b \times t_h \times c_h}$. In order to provide the encoder with early context we seed the LSTM hidden and cell states with two separately learned linear projections (passed through a $tanh$ activation) from a tensor $S \in \mathbb{R}^{b \times 2c}$, where $2c$ is a concatenation of the channel-wise mean and standard deviation of $\mathbf{X}$.

The **LSTM decoder** applies a learned linear projection to $\mathbf{V}$ that results in $\mathbf{H}' \in \mathbb{R}^{b \times t_h \times h}$. Then a basic LSTM (again three layers of 256 nodes) is applied to $\mathbf{H}'$ to produce a tensor of the same shape. A final learned linear projection produces $\hat{\mathbf{X}}' \in \mathbb{R}^{b \times t_h \times c}$ with the same interpolation-based upsampling to produce $\hat{\mathbf{X}} \in \mathbb{R}^{b \times t \times c}$. The decoder LSTM is seeded with a $tanh$ activated linear projection of $\mathbf{V}$.

Downstream classifier. All classification experiments on HAR use the same fixed 1D convolutional network, which was not tuned to this dataset. It consists of three convolutional blocks over the time axis with 64, 128 and 256 channels, respectively. Each block applies a convolution (kernel size 5, same padding), batch normalization, a ReLU, max pooling (factor 2) and dropout ($p = 0.3$).

The blocks are followed by global average pooling over time and a linear layer to the 6 class logits ($\approx$ 211K parameters in total). The classifier is trained with a cross-entropy loss and Adam (learning rate 10^{-3}, constant) for 100 epochs with a batch size of 256. Test accuracy on the real HAR test set is recorded after every epoch, and we report the best value over training, applied identically to all conditions. Each configuration is repeated over several classifier initialization seeds (three or five, see Fig. 4).

B HYPERPARAMETER TUNING

Our framework is largely tuned to the *Sines* dataset with MLP backbones. This was done through a combination of random and grid searches to find a ballpark of good performance. The resulting model was then transplanted to the other datasets, with a sweep over the latent compression ratio, since this is expected to be dataset-specific. During these sweeps we found:

- Loss instability on the *MuJoCo* dataset required cosine annealing of the learning rate, with a final learning rate of 10^{-6} and no warmup. We then adopted this schedule for all other datasets.
- On the *Energy* dataset, the MLP decoder oversmoothed the reconstructions, prompting us to introduce the LSTM backbones. These proved necessary for the *fMRI* and *HAR* datasets as well.

Table 1: Best latent sizes for each dataset. Note that the distinction between the time axis t_h and channel axis c_h only matters for the last three datasets (which use LSTM backbones).

	Sines	**Stocks**	**ETTh(1)**	**MuJoCo**	**Energy**	**fMRI**	**HAR**
t_h	6	7	5	8	24	24	64
c_h	1	1	1	1	2	20	2

We tuned hyperparameters on a "combined score" of the four evaluation metrics. This score first rescales each of the four evaluation scores onto a common scale via a log-ratio between its own empirically estimated floor (real vs. real) and ceiling (real vs. noise), then takes an unweighted average of the four rescaled scores. Note that tuning on the evaluation suite itself is a silent convention in the literature[3]. That is, we tune on the same scores that we use to compare our generators. This is an uncomfortable convention, but one we need to follow in order to remain comparable with other self-reported scores. See Appendix E.1 for more principled results on the two datasets for which we can obtain them. Those results suggest that this type of tuning does not substantially affect out-of-sample performance.

C DATASETS

- *Sines* Yoon et al. (2019): Generated partial sine waves with random frequencies and phases
- *Stocks* Yoon et al. (2019): Daily Google stock price data, 2004–2019
- *ETTh(1)* Yuan & Qiao (2024): Oil temperature plus six power load features, recorded July 2016–July 2018
- *MuJoCo* Jeon et al. (2022): Physics-simulation trajectories from random-policy rollouts in DeepMind's dm_control Hopper environment
- *Energy* Yoon et al. (2019): UCI Appliances Energy Prediction dataset
- *fMRI* Yuan & Qiao (2024): Simulated blood-oxygen-level-dependent (BOLD) time series
- *HAR* Anguita et al. (2013): Smartphone accelerometer and gyroscope data (UCI) for human activity recognition. The data comprises 30 subjects and 6 activities (classes). Not part of the standard unconditional benchmark suite above; we use it here as a class-conditional generation benchmark.

[3]None of our baselines report selecting hyperparameters on held-out data; where a selection criterion is disclosed at all (TimeGAN, SDformer), it is one of the evaluation metrics themselves.

Table 2: Time series generation benchmark datasets. All six unconditional datasets are used by Yuan & Qiao (2024), Chen et al. (2024), Hu et al. (2024), Kim et al. (2026) and Zhang et al. (2026); the last column lists additional works using each dataset.

dataset	size	timesteps	dimensions (d)	also used by
Sines	10000	24	5	(Yoon et al., 2019; Desai et al., 2021; Naiman et al., 2024b;a; Li et al., 2025; Jeon et al., 2022; Coletta et al., 2023)
Stocks	3662	24	6	(Yoon et al., 2019; Desai et al., 2021; Naiman et al., 2024b;a; Li et al., 2025; Jeon et al., 2022; Coletta et al., 2023)
ETTh(1)	17397	24	7	(Naiman et al., 2024a)
MuJoCo	10000	24	14	(Naiman et al., 2024b;a; Li et al., 2025; Jeon et al., 2022)
Energy	19712	24	28	(Yoon et al., 2019; Desai et al., 2021; Naiman et al., 2024b;a; Li et al., 2025; Jeon et al., 2022; Coletta et al., 2023)
fMRI	9977	24	50	(Li et al., 2025)
HAR	7352 + 2947	128	9	-

D Evaluation and Baselines

D.1 Metrics

We follow the implementation of Yuan & Qiao (2024), whose discriminative and predictive scores are in turn taken directly from Yoon et al. (2019), re-implemented in PyTorch and verified against the reference code. All four are lower-is-better.

- **Discriminative score** Yoon et al. (2019). Real and synthetic samples are each split 80/20 at random. A single-layer GRU classifier (hidden size $d/2$) is trained for 2000 iterations (batch size 128) to label real vs. synthetic, and the score is $|\text{accuracy} - 0.5|$ on the held-out 20%.
- **Predictive score** Yoon et al. (2019). Train-on-synthetic, test-on-real. A single-layer GRU (hidden size $d/2$) is trained for 5000 iterations on D_{synth} to predict the last dimension one step ahead from the other $d-1$ dimensions, at every timestep. The score is its mean absolute error on all of D_{train}.
- **Context-FID** Yuan & Qiao (2024). A TS2Vec encoder Yue et al. (2022) is trained on D_{train}, both datasets are embedded with it, and the score is the Fréchet distance between the two embedding distributions.
- **Correlational score** Yuan & Qiao (2024). For random subsets of D_{train} and D_{synth} (each of size $|D_{train}|/5$), the lag-0 correlation between every pair of channels is computed and averaged over samples. The score is the summed absolute difference between the real and synthetic correlation matrices, divided by 10 (following the reference implementation).

D.2 Evaluation repeats and training seeds

Every metric is stochastic: the classifier and predictor are retrained from random initializations, TS2Vec is retrained each time, and the Correlational score uses random subsets. Following Yuan & Qiao (2024), we therefore run the full suite 5 times on the same set of generated samples and report mean $\pm$ standard deviation. (Yoon et al. (2019), Jeon et al. (2022) and Naiman et al. (2024b) use 10 repeats for the discriminative score.)

These repeats capture only the variance of the evaluation, not that of training the generator. Each trained generator corresponds to one seed; we train 5 seeds per benchmark dataset (a single seed for HAR) and report the best of the five, as judged by the combined score (Appendix B). None of our

baselines state how many training seeds were run or how the reported model was selected; as far as we can trace in their code, their $\pm$ reflects evaluation repeats on a single trained generator. Note that baselines are inconsistent in what $\pm$ denotes: Yuan & Qiao (2024), Li et al. (2025) and Kim et al. (2026) report 95% confidence intervals, Jeon et al. (2022) and Naiman et al. (2024b) standard deviations, and Chen et al. (2024) a mix; we report standard deviations.

D.3 Choice of baselines

We aimed to cover each family of generative models with both a well-established and a recent method: GANs (TimeGAN Yoon et al. (2019), GT-GAN Jeon et al. (2022)), VAEs (KoVAE Naiman et al. (2024b)), diffusion (Diffusion-TS Yuan & Qiao (2024), PaD-TS Li et al. (2025)), flow matching (SMF Kim et al. (2026), PrismFlow Zhang et al. (2026)) and discrete-token transformers (SDformer Chen et al. (2024)). For every baseline except Zhang et al. (2026), we checked the public codebase to confirm that its self-reported scores were produced with the same datasets and a compatible evaluation protocol. Zhang et al. (2026) has no public code; we include it because it is the most recent method in our comparison and the current state of the art in the line of work we position ourselves against, but its scores could only be checked against its paper.

TimeVAE. We do not compare against Desai et al. (2021), even though it is widely used as a baseline. Its "sine" dataset uses a different generating process from the standard Sines, and its "stock" dataset is a different stock from the standard Stocks, despite the shared names; only Energy matches. Its discriminative and predictive scores also come from a custom re-implementation rather than the reference code. Its self-reported scores are therefore not comparable with ours.

SDformer. According to Feng et al. (2026), SDformer Chen et al. (2024) has 36.9M–95.3M parameters depending on the dataset, roughly 30–$150\times$ more than the other baselines. For this reason Kim et al. (2026) compares against it only in a separate "larger model" table, and Zhang et al. (2026) mentions it only in the text. We include it in our main comparison regardless. Note that it does not report the Correlational score.

D.4 Minor protocol differences between baselines

Checking the baselines' codebases revealed several small inconsistencies. We judged none of them large enough to exclude a baseline, but list them for transparency:

- **Repeat count:** 10 evaluation repeats for Yoon et al. (2019), Jeon et al. (2022) and Naiman et al. (2024b) (discriminative score), 5 for the rest.
- **Energy discriminative score:** Jeon et al. (2022) uses a smaller classifier and fewer iterations for Energy only; Naiman et al. (2024b) and Chen et al. (2024) reuse Jeon et al. (2022)'s metric code but without this Energy-specific setting.
- **Windowing:** Jeon et al. (2022) produces one extra window on Stocks and Energy (an off-by-one difference from Yoon et al. (2019)).
- **Dataset size:** Yuan & Qiao (2024) reports 3773 Stocks samples, which we could not reproduce from the data and code (3662 windows).
- **Missing metrics:** Li et al. (2025) reports neither Context-FID nor the Correlational score; Chen et al. (2024) does not report the Correlational score.

E Additional results

E.1 Out-of-sample performance

Because Sines and MuJoCo are generated from known processes, we can evaluate against fresh real data never seen during training or tuning. Table 3 shows no large drop from in-sample to out-of-sample scores; the largest change is a small increase in Context-FID (e.g. .003 to .007 on Sines, versus a real-vs-noise value of 15.17). Tuning on the in-sample training set therefore did not noticeably compromise out-of-sample performance.

The reference rows also expose a limitation of the evaluation suite: a perfect memorizer (the training set scored against a copy of itself) scores close to the real-vs-real floor, and Context-FID even assigns it a perfect score. Note that out-of-sample evaluation cannot detect memorization on these datasets either, since the training set is itself a true sample of the generating process.

Table 3: Our approach evaluated against several in- and out-of-sample references (Sines and MuJoCo only). Best seed rows: the best of 5 training seeds under that evaluation, mean ± standard deviation over 5 evaluation repeats. Seed average rows: mean ± standard deviation over the 5 seeds' per-seed means. Reference rows: real-vs-real (fresh) is pooled over 10 fresh draws × 10 repeats; real-vs-copy compares the training set against an exact copy of itself (a perfect memorizer), over 10 repeats. Lower is better for all metrics.

		Sines				MuJoCo			
Evaluation	Model	Disc.	Pred.	C-FID	Corr.	Disc.	Pred.	C-FID	Corr.
In-sample	Best seed (Ours)	.008±.005	.093±.000	.003±.000	.021±.002	.013±.010	.008±.000	.019±.001	.229±.030
	Seed average	.006±.002	.093±.000	.003±.000	.016±.002	.017±.003	.008±.000	.020±.001	.225±.005
Out-of-sample	Best seed	.007±.001	.093±.000	.007±.001	.018±.005	.016±.012	.008±.000	.022±.002	.234±.019
	Seed average	.006±.001	.093±.000	.007±.000	.019±.001	.021±.006	.008±.001	.026±.003	.230±.008
Reference	*real-vs-real (fresh)*	.007±.005	.093±.000	.003±.001	.023±.006	.006±.005	.007±.001	.007±.001	.267±.038
	real-vs-copy (memorizer)	.006±.005	.093±.000	.000±.000	.019±.005	.016±.006	.007±.001	.000±.000	.243±.027

E.2 Iterative refinement and consistency regularization

Refinement procedure. A sample is first generated as usual, by decoding a point drawn uniformly on the sphere (Eq. 3). One refinement step then encodes the generated sample, projects the encoding onto the sphere, and decodes it again. In our main refinement experiments (Section 6.3) no noise is added during this cycle. At each refinement level we generate as many samples as there are in the training set and run the full evaluation suite (5 repeats), exactly as for our main results.

Models. We refine two versions of each dataset's model. The first is our final model ("locked"), i.e. the one behind our main results (Fig. 2). The second adds the pixel and latent consistency losses of Yue et al. (2026): the pixel consistency loss compares the decodings of a strongly and a weakly perturbed latent (with a stop-gradient on the latter), and the latent consistency loss is the cosine distance between a latent and the re-encoded decoding of its strongly perturbed version. For each dataset we tuned the two loss weights and σ_{max} with a star-shaped sweep around a reference setting and trained 5 seeds of the selected setting. The selected weights are small (10^{-3} to 2×10^{-3} for the latent and 5×10^{-3} to 0.15 for the pixel consistency loss), with $\sigma_{max} = 0.1$ on ETTh(1) and fMRI and 0.25 elsewhere.

Noisy refinement. Yue et al. (2026) additionally perturb the latent at every refinement step before decoding it, with maximum-magnitude noise ($\sigma = \sigma_{max}$), and report their best results when the noise direction is drawn once per sample and shared across steps. We ran this variant as a dedicated experiment on the best seed of each model, at 1, 3, 5 and 9 refinement steps, which correspond to the 2, 4, 6 and 10 sampling steps in their ablation (they count the initial decoding as a step), and cover the range in which they report their largest gains.

Results. Fig. 8 shows all results. Without noise, refinement improves Context-FID and Correlational scores on Energy and, over the first few steps, Context-FID on ETTh(1), and mostly degrades scores elsewhere, with or without consistency regularization. Noise changes where refinement helps or hurts, but not the overall picture. It is less damaging than noiseless refinement on fMRI, but removes the Context-FID improvement on ETTh(1), and on Energy it yields a similar Context-FID improvement to noiseless refinement. On Sines, Stocks and MuJoCo, noisy refinement still degrades scores, though typically less than noiseless refinement over the same number of steps. None of the settings yields a consistent improvement across datasets.

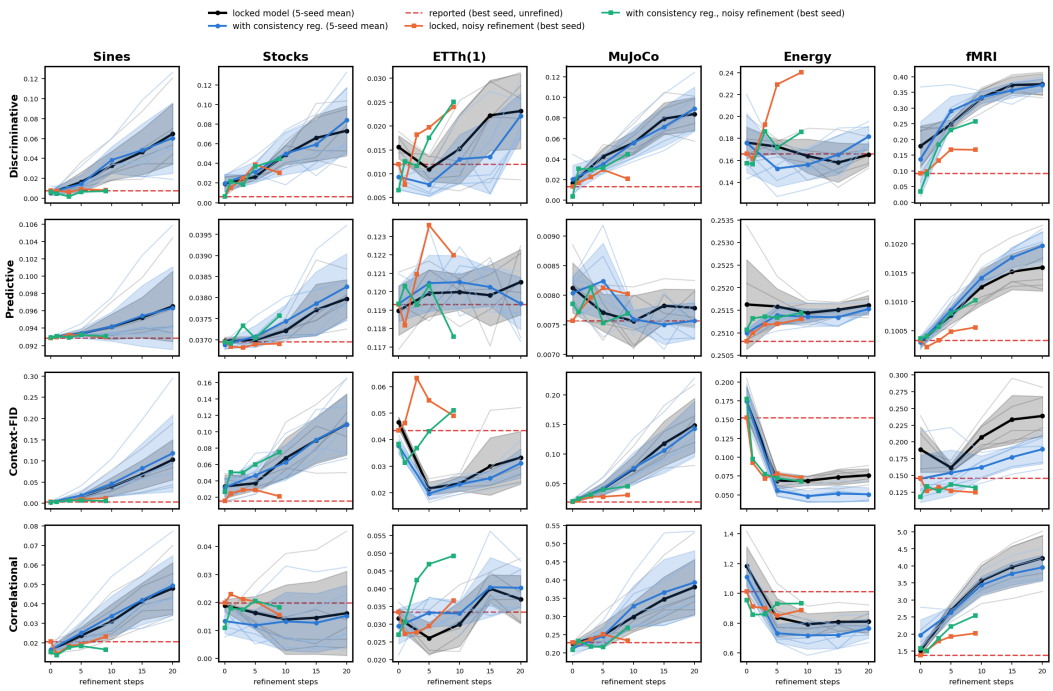


Figure 8: Evaluation scores under iterative refinement for all six datasets. Black and blue: noiseless refinement of our final model and of the model with consistency regularization, respectively. Thick lines show the mean over 5 training seeds, shading ± 1 standard deviation, and thin lines individual seeds, evaluated every 5 steps. Orange and green: noisy refinement as in Yue et al. (2026) (maximum noise magnitude, noise direction shared across steps) of the best seed of each model, evaluated at 0, 1, 3, 5 and 9 steps. Red dashed line: our reported score (best seed, unrefined). Each point is the mean over 5 evaluation repeats; lower is better for all metrics.

E.3 Full performance table

Table 4: Numerical values behind Fig. 2. Baseline scores are self-reported (see Appendix D for what their ± denotes); ours are the best of 5 training seeds, mean ± standard deviation over 5 evaluation repeats. Reference rows: real-vs-real (fresh) is pooled over 10 fresh draws × 10 repeats (Sines and MuJoCo only); real-vs-real 50/50 and real-vs-noise are over 5 repeats of a single split/draw. A dash means not reported. Lower is better for all metrics.

Metric	Method	Sines	Stocks	ETTh(1)	MuJoCo	Energy	fMRI
Discriminative	TimeGAN Yoon et al. (2019)	.011±.008	.102±.021	–	–	.236±.012	–
	GT-GAN Jeon et al. (2022)	.012±.014	.077±.031	–	.245±.029	.221±.068	–
	KoVAE Naiman et al. (2024b)	.005±.003	.009±.006	–	.076±.017	.143±.011	–
	Diffusion-TS Yuan & Qiao (2024)	.006±.007	.067±.015	.061±.009	.008±.002	.122±.003	.167±.023
	PaD-TS Li et al. (2025)	.013±.004	.055±.087	–	.016±.005	.078±.011	.153±.032
	SDformer Chen et al. (2024)	.006±.004	.010±.006	.003±.001	.008±.005	.006±.004	.017±.007
	SMF Kim et al. (2026)	.006±.006	.022±.013	.027±.010	.005±.004	.161±.021	.136±.207
	PrismFlow Zhang et al. (2026)	.004±.003	.030±.004	.005±.004	.004±.008	.039±.008	.129±.053
	Ours	.008±.005	.006±.005	.012±.004	.013±.010	.166±.035	.092±.050
	real-vs-real (fresh)	.007±.005	–	–	.006±.005	–	–
	real-vs-real 50/50	.009±.006	.012±.006	.004±.001	.010±.005	.006±.005	.012±.007
	real-vs-noise	.499±.000	.500±.000	.498±.001	.500±.000	.500±.000	.500±.000
Predictive	TimeGAN Yoon et al. (2019)	.093±.019	.038±.001	–	–	.273±.004	–
	GT-GAN Jeon et al. (2022)	.097±.000	.040±.000	–	.055±.000	.312±.002	–
	KoVAE Naiman et al. (2024b)	.093±.000	.037±.000	–	.038±.002	.251±.000	–
	Diffusion-TS Yuan & Qiao (2024)	.093±.000	.036±.000	.119±.002	.007±.000	.250±.000	.099±.000
	PaD-TS Li et al. (2025)	.093±.000	.037±.000	–	.008±.002	.251±.011	.100±.000
	SDformer Chen et al. (2024)	.093±.000	.037±.000	.118±.002	.007±.001	.249±.000	.091±.002
	SMF Kim et al. (2026)	.093±.000	.037±.000	.123±.005	.008±.001	.251±.000	.100±.000
	PrismFlow Zhang et al. (2026)	.093±.000	.037±.000	.122±.003	.006±.003	.251±.003	.100±.000
	Ours	.093±.000	.037±.000	.119±.002	.008±.000	.251±.000	.100±.000
	real-vs-real (fresh)	.093±.000	–	–	.007±.001	–	–
	real-vs-real 50/50	.092±.000	.037±.000	.118±.003	.007±.001	.249±.000	.088±.002
	real-vs-noise	.260±.003	.420±.007	.199±.001	.083±.001	.251±.000	.113±.004
Context-FID	TimeGAN Yoon et al. (2019)	–	–	–	–	–	–
	GT-GAN Jeon et al. (2022)	–	–	–	–	–	–
	KoVAE Naiman et al. (2024b)	–	–	–	–	–	–
	Diffusion-TS Yuan & Qiao (2024)	.006±.000	.147±.025	.116±.010	.013±.001	.089±.024	.105±.006
	PaD-TS Li et al. (2025)	–	–	–	–	–	–
	SDformer Chen et al. (2024)	.001±.000	.002±.000	.008±.001	.005±.001	.003±.000	.015±.001
	SMF Kim et al. (2026)	.004±.001	.008±.003	.058±.007	.018±.002	.051±.009	.116±.004
	PrismFlow Zhang et al. (2026)	.003±.000	.027±.008	.015±.001	.006±.005	.022±.002	.155±.002
	Ours	.003±.000	.015±.001	.044±.001	.019±.001	.153±.007	.146±.011
	real-vs-real (fresh)	.003±.001	–	–	.007±.001	–	–
	real-vs-real 50/50	.004±.000	.006±.001	.005±.000	.018±.001	.006±.000	.032±.001
	real-vs-noise	15.17±.88	14.76±2.26	10.31±.34	11.74±.62	20.46±1.08	20.10±1.16
Correlational	TimeGAN Yoon et al. (2019)	–	–	–	–	–	–
	GT-GAN Jeon et al. (2022)	–	–	–	–	–	–
	KoVAE Naiman et al. (2024b)	–	–	–	–	–	–
	Diffusion-TS Yuan & Qiao (2024)	.015±.004	.004±.001	.049±.008	.193±.027	.856±.147	1.411±.042
	PaD-TS Li et al. (2025)	–	–	–	–	–	–
	SDformer Chen et al. (2024)	–	–	–	–	–	–
	SMF Kim et al. (2026)	.027±.012	.010±.007	.040±.015	.173±.016	.732±.107	.737±.021
	PrismFlow Zhang et al. (2026)	.017±.004	.007±.003	.022±.010	.203±.030	.495±.033	.898±.025
	Ours	.021±.002	.020±.006	.034±.008	.229±.030	1.012±.060	1.374±.028
	real-vs-real (fresh)	.023±.006	–	–	.267±.038	–	–
	real-vs-real 50/50	.025±.006	.006±.004	.042±.007	.282±.038	.632±.150	1.092±.031
	real-vs-noise	.476±.004	1.278±.005	.471±.009	.827±.011	10.398±.024	5.070±.045