Generating Time Series by Matching Random Convolutional Features

Nikita Zozoulenko*
Department of Mathematics
Imperial College London
n.zozoulenko23@imperial.ac.uk

Konrad Jakob Mueller*
Department of Mathematics
Imperial College London
k.mueller23@imperial.ac.uk

Thomas Cass
Department of Mathematics
Imperial College London

Lukas Gonon
Department of Mathematics/School of Computer Science
Imperial College London/University of St. Gallen

Abstract

Generating realistic financial time series is challenging as training data is typically limited to a single historical path, making adversarial training prone to discriminator overfitting. Instead, recent work replaces the trained discriminator with a fixed feature map, and trains to generate paths whose features match those of real time series. As features, most prior work chooses path statistics based on signatures from rough path theory. Motivated by strong empirical results of random convolutional features on time series classification, we train generative models with **SOCK** (***S**oftmax-Variance **O**f **C**ompeting **K**ernels*), a novel random convolutional feature map that is fully differentiable, highly scalable, and simple to implement. SOCK consistently outperforms other convolutional or signature-based feature maps on hypothesis testing benchmarks and on training generative models.

1 Introduction

Training generative time series models with limited real-world data is a fundamental challenge across many domains. In finance, synthetic time series are commonly used to train neural policies for hedging or investment problems [4, 31, 5], as policy optimization requires far more sample paths than historical data provides. Training generative models adversarially with parametric discriminators, tends to fail with such scarce training data: the discriminator is likely to overfit and drive the generator to memorize training samples [17, 16]. Instead of training both a generator and a discriminator, recent work avoids the mini-max optimization and trains to generate paths that have similar *features* as the real time series [26, 20]. Concretely, the generator g_θ minimizes

$$\|\mathbb{E}_{X\sim p}[f(X)] - \mathbb{E}_{X\sim g_\theta}[f(X)]\|_2^2, \tag{1}$$

where $X = \{X_1, \dots, X_T\}$ is the $\mathbb{R}^d$-valued time series of length T, and p denotes the training data distribution. The feature map $f : \mathbb{R}^{T\times d} \to \mathbb{R}^n$ is not trained and therefore not overfitted to p. This makes training generative models via feature-matching [27] suited to the scarce-data regime in finance, where data is often limited to a few years of daily observations ($\sim$252 data points per year).

Motivated by rough path theory [23], most prior work [16, 26, 20, 5, 1, 21, 30] chooses the path statistics $f(X)$ from the signature-based family of path-to-vector transformations [24, 18, 7, 22]. While the signature transform has strong theoretical properties, other, more heuristically motivated, unsupervised feature maps have recently been shown to empirically outperform the truncated signature in a large scale study on univariate time series classification [25]. Among the best performing feature

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Generative AI in Finance.

maps are ROCKET [10, 11, 29] and Hydra [12, 13], which are based on random convolutions of the input path, followed by non-differentiable temporal pooling operations.

The choice of the feature map f is important because it limits what the generator can learn: if f is 'blind' to certain properties of the true data-generating process, the generator will not be able to reproduce those qualities. In this work, we propose training generative models via a novel, highly expressive, random convolutional feature map. Our main contributions are:

1. **SOCK features.** We introduce ***S**oftmax-Variance **O**f **C**ompeting **K**ernels* (SOCK), a random convolutional feature map for multivariate time series. SOCK uses a novel differentiable pooling operator, is robust to its few hyperparameters, and is simple to implement.
2. **Discriminative & generative evaluation.** We compare SOCK to other feature maps on two tasks: two-sample hypothesis testing of stochastic processes, and generative training from a single historical path via feature-matching. Across these benchmarks, SOCK consistently outperforms other common convolutional and signature-based feature maps.
3. **Resampled features.** When supervising generators with random feature maps, we periodically *resample* the parameters. This effectively trains the generator on an ensemble of discriminators and improves the performance of SOCK and randomized-signature features.

2 Random Convolutional Features

The SOCK feature map consists of a preprocessing step, followed by convolutions with competing random kernels, and a final pooling layer. SOCK is based on Hydra [12], but has two important modifications. First, since training generative models through feature matching requires differentiating through the feature map, we replace Hydra's non-differentiable pooling with a novel differentiable pooling function. Second, to handle multivariate input sequences, we add a linear up-projection during preprocessing.

Let us first describe the pooled convolutional features using a univariate input path $X \in \mathbb{R}^T$. We sample random kernels $w_i \sim N(0, \sigma^2 I_k)$ of length k and compute the cross-correlation

$$Y_{t,i} = \sum\nolimits_{r=-(k-1)/2}^{(k-1)/2} w_{i,r}\, X_{t+r}, \qquad t = 1, \ldots, T,\ i = 1, \ldots, K,$$

with the convention $X_t = 0$ for $t \notin \{1, \ldots, T\}$. Hydra treats the random convolutions as a competition between kernels at each time step, where the winning kernel $w^*(t) = \arg\max_{w_i} Y_{t,i}$, achieves the largest alignment with the local X-window. Hydra then pools the local competitions into global features via simple statistics (counts and conditional sums), that summarize how often and strongly each kernel wins and loses. These features use both $\arg\max$ and indicator functions, and are therefore not differentiable. We replace the non-differentiable 'hard winner selection' with win probabilities

$$p_{t,i} = \frac{\exp(Y_{t,i}/\tau)}{\sum_{j=1}^{K} \exp(Y_{t,j}/\tau)}, \qquad t = 1, \ldots, T,\ i = 1, \ldots, K,$$

computed at temperature $\tau > 0$. We pool the probabilities across time via their standard deviation

$$\sigma_i = \operatorname{std}\left(p_{1,i}, \ldots, p_{T,i}\right), \qquad i = 1, \ldots, K,$$

and stack $\{\sigma_i\}_{i=1}^K$ to a feature vector of size K. This std-pooling is simple, differentiable, and reduces to a single feature per kernel. We found in preliminary experiments that std-pooling yields highly expressive features and outperforms other pooling operators.

The pooled random convolutions form the final layer of our feature map. The d-dimensional input path is first preprocessed by a pointwise linear up-projection into an M-dimensional path. The up-projected path is then further augmented with its first order difference across time. On each of the $2M$ channels of the augmented path, we apply the described conv-to-pool transformation to obtain a $2MK$-dimensional feature vector. As in Hydra, we repeat the convolutions at different dilations. The SOCK feature map can be implemented in only 10 lines of PyTorch code (see appendix A).

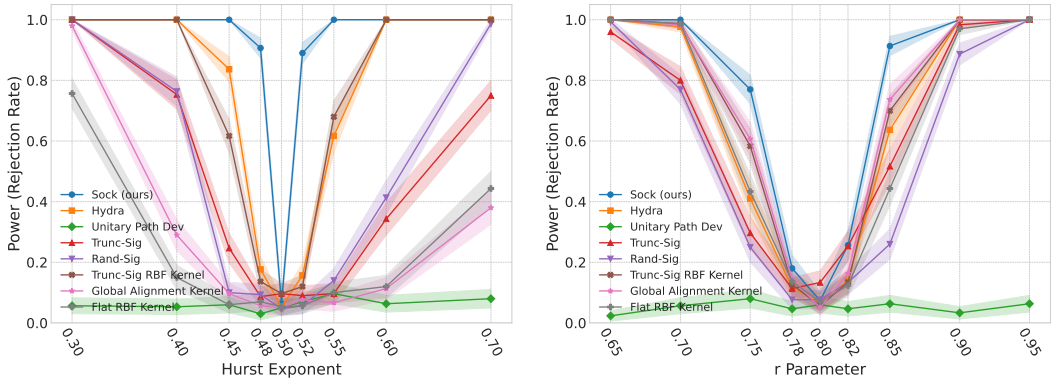

Figure 1: Permutation test power at significance level $\alpha = 0.05$ for feature- and kernel-based MMDs. Results are shown for Fractional Brownian Motions (left) and VAR(10) processes (right).

3 Hypothesis Testing

We illustrate the expressive power of SOCK features on a two-sample hypothesis test for stochastic processes. Specifically, we use the maximum mean discrepancy (MMD) [14] based on either explicit feature vectors (eq. 1) or time series kernels, within a permutation test as in [6, 28, 21, 19, 30]. We test the null hypothesis $H_0 : \mu = \nu$ against the alternative $H_1 : \mu \neq \nu$, where μ and ν denote laws of stochastic processes.

Let $\mathcal{X} = (X^{(1)}, \dots, X^{(N)}) \sim \mu$ and $\mathcal{Y} = (Y^{(1)}, \dots, Y^{(M)}) \sim \nu$, be i.i.d. samples of size N and M, respectively. The test proceeds as follows:

1. Let $\mathcal{Z} = (X^{(1)}, \dots, X^{(N)}, Y^{(1)}, \dots, Y^{(M)})$. Randomly partition $\mathcal{Z}$ into sets $\widetilde{\mathcal{X}}$ and $\widetilde{\mathcal{Y}}$ of sizes N and M respectively. Compute and record $\mathrm{MMD}(\widetilde{\mathcal{X}}, \widetilde{\mathcal{Y}})$. Repeat 100 times.
2. Compute the empirical $(1 - \alpha)$ quantile of the permuted MMD values, which under H_0 has the same distribution as the unpermuted statistic $\mathrm{MMD}(\mathcal{X}, \mathcal{Y})$.
3. Reject H_0 if $\mathrm{MMD}(\mathcal{X}, \mathcal{Y})$ exceeds this threshold.

This procedure is repeated for 300 trials to estimate the test power. We consider two scenarios: 1.) Fractional Brownian Motions (fBm) with Hurst parameter $H \in (0, 1)$. We set $H = 0.5$ under H_0, and vary H under H_1. 2.) A VAR(10) process with oscillating autocorrelation [3], defined by

$$X_t = \phi_1 X_{t-5} + \phi_2 X_{t-10} + \varepsilon_t, \quad \phi_1 = 2r\cos(\frac{4\pi}{5}), \quad \phi_2 = -r^2, \quad \varepsilon_t \sim N(0, \Sigma),$$

with $\Sigma = \frac{1}{2}J + \frac{1}{2}I$ as in [20]. We set $r = 0.85$ under H_0, and vary r under H_1.

We benchmark SOCK by comparing the power of the permutation test against common feature- and kernel-based MMDs. Feature-based baselines include truncated signatures [26], randomized signatures [7], unitary path development (PCF) [22, 21], and Hydra [12]. Kernel baselines include the truncated RBF signature kernel [18], the Global Alignment Kernel [9, 8], and the classical flattened RBF kernel. Results are shown in fig. 1. SOCK achieves the highest power across both settings by a wide margin, and outperforms Hydra features. For details on hyperparameter tuning, see appendix B.

4 Time Series Generation

We now compare the feature maps for training generative models through feature-matching. As in practice, we fit generative models using training data from a single historical path of length L to generate new paths of length $T < L$. We estimate the target vector $\mathbb{E}_{X\sim p}[f(X)]$ by averaging features computed on length-T windows, cut from the historical path.

We train generators on three simulated time series (VAR(1), VAR(10), Heston) and two real financial time series (Crypto, 5-Tickers). We limit the simulated training data to $L = 2^{11}$ observations, which

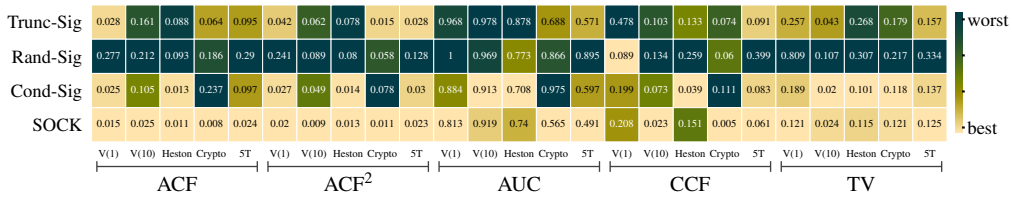

Figure 2: Results for generative modeling tasks, evaluated on synthetic and real financial time series. Lighter colors indicate a smaller (better) difference between real and generated distributions.

corresponds to approximately 8 years of daily observations. For validation, we simulate 2^{16} new paths of length T. Following [20], we consider the 3-dimensional VAR(1) process

$$X_{t+1} = \phi X_t + \varepsilon_t, \quad \varepsilon_t \sim N(0, \Sigma),$$

where Σ is as in section 3. We vary the scalar $\phi \in \{0.5, 0.9, 0.99\}$ and thus study processes with much stronger autocorrelation compared to previous studies [20, 1]. For the 3-dimensional VAR(10), we use the same coefficients as in section 3. As a final synthetic benchmark, we include the joint process of log-returns and log-variance in the Heston model [15, 2]. The Crypto time series contains the 1-min log-returns of `BTC` and `ETH` over the first two months of 2025. The 5-tickers dataset consists of the daily log-returns of `MSFT`, `JPM`, `PG`, `^VIX`, and `GC=F` (Gold futures) between 2009 and 2025. We use the first half of the real time series for training and the second half for evaluation.

For each dataset and feature map, we train an autoregressive generator $g_\theta : \mathbb{R}^{qd} \times \mathbb{R}^d \to \mathbb{R}^d$ [20]. Given a context length q and an initial path segment $x_{-q:0}$, we generate a path $x_{1:T}$ by iterating

$$x_{t+1} = g_\theta(x_{t-q:t}, z_{t+1}), \quad t = 0, \dots, T-1,$$

where $z_{t+1} \overset{\text{iid}}{\sim} N(0, I_d)$. We set $q = 1$ for VAR(1) and Heston, and $q = 5$ for VAR(10) and the real datasets. The mismatch between $q = 5$ and the VAR order $p = 10$ mimics a challenge with real datasets: the true dynamics may lie outside the generator's model class. For all datasets, we train on generated paths of length $T = 64$. This rollout length is larger than the lengths of 3-10 used in prior works [20, 22, 1], which we found insufficient to train generators that can be unrolled to length 64 at test time. With the long rollout length, each feature map truly computes *path*-level statistics.

We train generators via feature-matching using three fully untrained feature maps: the truncated signature, the randomized signature, and SOCK. At each training step, we generate a batch of 2^{10} paths and compute the gradient of the feature-matching loss (eq. 1) with respect to θ. We update θ using Adam and a linearly annealed learning rate, starting at 3×10^{-4}. We resample the random feature maps (Rand-Sig and SOCK) every 10 training iterations. We also compare to the conditional signature [20], which fits a matrix parameter prior to training the generator and optimizes a loss, that requires nested sampling. All details can be found in appendix C. In preliminary experiments, we also included the signature-inspired path development as in PCF-GAN [21], but we failed to obtain a convergent model, and hence do not report any generative results using the PCF feature map. We aim to investigate this further in the full release of our paper.

Results We generate a set of paths with each trained generator and compare it to the validation set of real paths. On both the real and generated paths, we estimate the autocorrelation function and report the *difference* between the two autocorrelation functions as ACF. Similarly, we report the difference of the autocorrelation functions of the squared processes (ACF^2), the difference in cross-correlations (CCF), and the total variation distance of the marginal distributions (TV). We also report the ROC-AUC of an MLP discriminator, whose final layer is trained to distinguish between real and fake paths.

We provide all results in appendix D and visualize most of them in fig. 2. We observe that on most datasets and metrics, the SOCK-trained generator either matches or beats the performance of all other generators. SOCK often outperforms the other untrained feature maps (Trunc-Sig, Rand-Sig) by a wide margin and even performs typically better than Cond-Sig, whose features are specialized on each dataset.

On the real datasets, the SOCK-trained generators clearly outperform all others. In particular, SOCK performs much stronger than Cond-Sig, despite Cond-Sig being a strong baseline on the synthetic

data. On VAR(1) with $\phi = 0.5$, Rand-Sig, Cond-Sig, and SOCK all perform very well, which matches positive results in the literature [20, 1]. Note that with $\phi = 0.5$, autocorrelation decays fast (half-life of a single time step). When $\phi = 0.99$, the generator trained with Rand-Sig fails to accurately reproduce the autocorrelation, while the SOCK generator is still accurate. That SOCK is strong at capturing autocorrelations is further apparent from the strong performance on VAR(10) (see fig. 3), where the SOCK generator far outperforms all others. On Heston, the SOCK-trained generator is slightly weaker than the conditional signature in matching the distribution of individual points X_t. We aim to investigate this in future work.

Finally, we highlight a unique advantage of random feature maps: we can resample parameters, like SOCK's convolutional kernels, during generator training. The generator faces different discriminators during training, which improves performance for SOCK and Rand-Sig (table 3).

References

[1] Francesca Biagini, Lukas Gonon, and Niklas Walter. Universal randomised signatures for generative time series modelling. *Preprint, arXiv 2406.10214*, 2024.

[2] Mark Broadie and Özgür Kaya. Exact Simulation of Stochastic Volatility and Other Affine Jump Diffusion Processes. *Operations Research*, 54(2):217–231, 2006. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.1050.0247. URL https://pubsonline.informs.org/doi/10.1287/opre.1050.0247.

[3] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, Cham, 2016. ISBN 978-3-319-29852-8 978-3-319-29854-2. doi: 10.1007/978-3-319-29854-2. URL http://link.springer.com/10.1007/978-3-319-29854-2.

[4] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19 (8):1271–1291, 2019.

[5] Hans Buehler, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. Generating Financial Markets With Signatures. *SSRN Electronic Journal*, 2020. ISSN 1556-5068. doi: 10.2139/ssrn.3657366. URL https://www.ssrn.com/abstract=3657366.

[6] Ilya Chevyrev and Harald Oberhauser. Signature moments to characterize laws of stochastic processes. *Journal of Machine Learning Research*, 23(176):1–42, 2022.

[7] Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Juan-Pablo Ortega, and Josef Teichmann. Expressive power of randomized signature. In *Advances in Neural Information Processing Systems*, 2021.

[8] Marco Cuturi. Fast global alignment kernels. In *Proceedings of the 28th International Conference on Machine Learning*, page 929–936. Omnipress, 2011.

[9] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. A kernel for time series based on global alignments. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages II–413–II–416, 2007.

[10] Angus Dempster, François Petitjean, and Geoffrey I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5): 1454–1495, 2020. ISSN 1573-756X.

[11] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 248–257, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325.

[12] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Hydra: competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5):1779–1805, Sep 2023. ISSN 1573-756X.

[13] Angus Dempster, Chang Wei Tan, Lynn Miller, Navid Mohammadi Foumani, Daniel F. Schmidt, and Geoffrey I. Webb. Highly scalable time series classification for very large datasets. In *Advanced Analytics and Learning on Temporal Data: 9th ECML PKDD Workshop, AALTD 2024, Vilnius, Lithuania, September 9–13, 2024, Revised Selected Papers*, page 80–95, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-77065-4.

[14] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. URL http://jmlr.org/papers/v13/gretton12a.html.

[15] Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343, 1993. URL https://academic.oup.com/rfs/article-abstract/6/2/327/1574747.

[16] Zacharia Issa, Blanka Horvath, Maud Lemercier, and Cristopher Salvi. Non-adversarial training of Neural SDEs with signature kernel scores.

[17] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training Generative Adversarial Networks with Limited Data, June 2020. URL https://arxiv.org/abs/2006.06676v2.

[18] Franz J. Kiraly and Harald Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 20(31):1–45, 2019.

[19] Siran Li, Zijiu Lyu, Hao Ni, and Jiajie Tao. Restricted path characteristic function determines the law of stochastic processes, 2024. URL https://arxiv.org/abs/2404.18661.

[20] Shujian Liao, Hao Ni, Marc Sabate-Vidales, Lukasz Szpruch, Magnus Wiese, and Baoren Xiao. Sig-Wasserstein GANs for conditional time series generation. *Mathematical Finance*, 34(2):622–670, 2024. ISSN 1467-9965. doi: 10.1111/mafi.12423. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/mafi.12423.

[21] Hang Lou, Siran Li, and Hao Ni. Pcf-gan: generating sequential data via the characteristic function of measures on the path space. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 39755–39781. Curran Associates, Inc., 2023.

[22] Hang Lou, Siran Li, and Hao Ni. Path development network with finite-dimensional lie group. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

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

[24] Andrew McLeod and Terry Lyons. Signature methods in machine learning. *EMS Surveys in Mathematical Sciences*, 2025. doi: 10.4171/EMSS/95. Published online first.

[25] Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 38(4):1958–2031, 2024. ISSN 1573-756X.

[26] Hao Ni, Lukasz Szpruch, Marc Sabate-Vidales, Baoren Xiao, Magnus Wiese, and Shujian Liao. Sig-wasserstein GANs for time series generation. In *Proceedings of the Second ACM International Conference on AI in Finance*, ICAIF '21, pages 1–8, New York, NY, USA, May 2022. Association for Computing Machinery. ISBN 978-1-4503-9148-1. doi: 10.1145/3490354.3494393. URL https://doi.org/10.1145/3490354.3494393.

[27] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved Techniques for Training GANs.

[28] Cristopher Salvi, Maud Lemercier, Chong Liu, Blanka Horvath, Theodoros Damoulas, and Terry Lyons. Higher order kernel mean embeddings to capture filtrations of stochastic processes. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

[29] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I. Webb. Multirocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5):1623–1646, Sep 2022. ISSN 1573-756X.

[30] Jiajie Tao, Hao Ni, and Chong Liu. High rank path development: an approach to learning the filtration of stochastic processes. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 115309–115350. Curran Associates, Inc., 2024.

[31] Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. Deep Hedging: Learning to Simulate Equity Option Markets, November 2019. URL http://arxiv.org/abs/1911.01700.

[32] Yue Wu, Hao Ni, Terence J. Lyons, and Robin L. Hudson. Signature features with the visibility transformation. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4665–4672, 2021. doi: 10.1109/ICPR48806.2021.9412642.

A Code

```python
from typing import List
from torch import Tensor
import torch
import torch.nn.functional as F

TAU = 1e-2  # softmax temperature

def sock(x: Tensor, w_mix: Tensor, w_conv: Tensor, dilations: List[int]) -> Tensor:
    """
    Args:
        x (Tensor): shape `(B, T, C)`,
            where B = batch size, T = num time steps, C = num channels.
        w_mix (Tensor): shape `(M, C)`, where M = mixing dimension.
        w_conv (Tensor): shape `(2KM, 1, k)`,
            where K = num kernels and k = kernel length.
        dilations (List[int]): list of dilation factors.
    Returns:
        Tensor: shape `(B, 2MK * len(dilations))`.
    """
    x = F.linear(x, w_mix).permute(0, 2, 1)  # (B, M, T)
    x = torch.cat([x, F.pad(torch.diff(x, dim=-1), (1, 0))], dim=1)  # (B, 2M, T)
    g, K = x.size(1), w_conv.size(0) // x.size(1)
    f = []
    for d in dilations:
        y = F.conv1d(x, w_conv, dilation=d, padding="same", groups=g)
        y = y.view(y.size(0), -1, K,  y.size(-1)) # (B, 2M, K, T)
        y = torch.std(F.softmax(y / TAU, dim=2), dim=3)  # (B, 2M, K)
        f.append(y.reshape(y.size(0), -1))
    return torch.cat(f, dim=-1)
```

B Additional Details on Hypothesis Tests

In the hypothesis testing experiments, we tune the hyperparameters of each MMD model via independent trials with different seeds from the reported results. Hydra uses its default hyperparameters and requires no tuning as per the original paper [12]. For PCF, we use the optimized hyperparameters from [21, 30], which are based on the same hypothesis testing experiment. We use a grid search to tune the σ parameter for all RBF-based models, including GAK, and to determine the optimal truncation depth for all truncated signature MMDs. For the randomized signature, we tune the path dimension and the variance of the random maps. For SOCK we set the number of competing kernels to $k = 3$ and tuned the temperature, but we note that this only had a small increase in performance. We stress that SOCK, like its predecessor Hydra, is robust to changes in hyperparameters and we recommend using the defaults for new problems. Additionally, for all models we preprocess all paths with the standard augmentations from the signature literature, namely Lead-Lag, Add-time, and I-visibility transformations [24, 32]. We note however that this has little effect on SOCK, but is required to make signatures competitive.

C Additional Details on Generative Modeling

C.1 Datasets

Heston We consider the Heston model [15]

$$dS_t = S_t\sqrt{V_t}dW_t,$$
$$dV_t = k(\theta - V_t)dt + \xi\sqrt{V_t}dB_t,$$

and take $\kappa = 1.0$, $\theta = 0.04$, $\xi = 0.25$, $\rho = -0.7$. We simulate (S, V) paths using the scheme of Broadie and Kaya [2] at a time discretization of $\Delta t = 1/250$. We then transform (S, V) into

$$X_t := (\log S_t - \log S_{t-\Delta t}, \log V_t).$$

The generative models are trained to generate this two-dimensional process X.

Crypto The Crypto dataset is a two month long 2-dimensional path of BTCUSD and ETHUSD, consisting of 1min log-returns from the Binance exchange. We use the first month for training, and the second month for evaluation.

5-Tickers The 5-Tickers dataset is 5-dimensional and consists of the daily log returns of `MSFT` (Microsoft), `JPM` (JP Morgan), `PG` (Proctor and Gable), `^VIX` (VIX), `GC=F` (Gold Futures), between the years 2009 and 2025. We obtained the data from Yahoo Finance.

C.2 Training generators via feature-matching

We train generators using the untrained feature maps (Trunc-Sig, Rand-Sig, SOCK) via feature matching (eq. 1) as in [26, 1] and train the conditional signature, via the algorithm of Liao et al. [20]. We find that many training choices, such as the bootstrapping schemes for computing $\mathbb{E}_{X\sim p}[f(X)]$ or sampling $x_{-q:0}$, are not sensitive to the choice of the feature map, so we keep them constant across all feature maps. We found in preliminary experiments that all feature maps have similar optimal learning rates. For simplicity, we therefore take the same learning rate schedule for each method (linear warm up for 5% of iterations to 3×10^{-4}, and linear decay for 70% of iterations). We note that for the full version of this paper, we aim to tune learning rates granularly and individually for each feature map. We train each generator at batch size 1024 for 10,000 iterations.

We use Trunc-Sig, Rand-Sig, and Cond-Sig with standard augmentations (Lead-Lag, Add-time, I-visibility). We also use Lead-Lag and Add-time with SOCK, but find that this affects performance only marginally. For Trunc-Sig and Cond-Sig we use truncation depth 3 of the signature, which we found to be optimal in preliminary runs and is in line with prior work. We highlight that we don't find an increase in performance when increasing the truncation depth much beyond this depth. For Rand-Sig, we use a hidden dimension of size 128. We use SOCK with mixing dimension $M = 64$ and reuse the Hydra defaults [12] for all other hyperparameters (kernel length of $k = 9$ and number of concurrent kernels $K = 8$). The generator architecture is the same for all feature maps and consists of an initial projection layer to the hidden dimension of 256, two gated residual blocks, and a linear readout.

C.3 Evaluation metrics

Each evaluation metric is computed on N 'real' paths and N 'fake' paths of length $T = 64$. On the synthetic datasets, we sample those real paths from the true known data generating process. There, we use 65,536 paths of length T. On the real datasets, the real paths are cut from the longer real data path. If not specified otherwise, we report performance on the out-of-sample part of the real data set.

In the following we use r for quantities that belong to the real sample, and g for quantities that belong to the sample generated by the trained generator.

We denote by

$$\mathrm{TV} = \frac{1}{d}\sum_{i=1}^{d} \left\| p_i^{(r)} - p_i^{(g)} \right\|_{\mathrm{TV}},$$

the average total variation of the channel-wise marginal distributions $p_i^{(\cdot)}$,

$$\mathrm{ACF} = \frac{1}{ds}\sum_{i=1}^{d}\sum_{k=1}^{s} \left| \rho_i^{(r)}(k) - \rho_i^{(g)}(k) \right|,$$

the average difference in the channel-wise autocorrelation functions $\rho_i^{(\cdot)}$, up to a max lag of $s = \lfloor T/3 \rfloor$,

$$\mathrm{ACF}^2 = \frac{1}{ds}\sum_{i=1}^{d}\sum_{k=1}^{s} \left| \varrho_i^{(r)}(k) - \varrho_i^{(g)}(k) \right|,$$

the average difference in the channel-wise autocorrelation functions $\varrho_i^{(\cdot)}$ of the squared process $(X_1^2, \ldots, X_T^2)$,

$$\mathrm{CCF} = \frac{1}{d(d-1)} \sum_{i \neq j} \left| \chi_{ij}^{(r)} - \chi_{ij}^{(g)} \right|,$$

where $\chi_{ij}^{(\cdot)}$ is the Pearson cross-correlation between channels i and j.

As a final evaluation metric, we report ROC-AUC of a discriminator, trained to distinguish between real and fake paths. We use $N/2$ real and $N/2$ fake paths to train the discriminator and evaluate it on the other half of evaluation paths. The discriminator uses a non-trainable one-layer random MLP feature map. We train a linear readout mapping the random MLP features to a scalar y, such that $\sigma(y)$ is the probability that the input path is real (where σ is the sigmoid function). A larger AUC indicates a stronger discriminator and weaker generator. If the generator perfectly replicates the true distribution, the AUC is 0.5.

D Results

Below, we report the results of all our experiments and models across 5 random seeds.

Table 1: Results on synthetic datasets.

		Trunc-Sig	Rand-Sig	Cond-Sig	SOCK
Heston	ACF	0.088 (0.040)	0.093 (0.079)	**0.013 (0.008)**	**0.011 (0.008)**
	ACF^2	0.078 (0.049)	0.080 (0.066)	**0.014 (0.004)**	**0.013 (0.013)**
	CCF	0.133 (0.050)	0.259 (0.214)	**0.039 (0.037)**	0.151 (0.181)
	TV	0.268 (0.124)	0.307 (0.230)	**0.101 (0.037)**	**0.115 (0.057)**
	AUC	0.878 (0.073)	**0.773 (0.142)**	**0.708 (0.118)**	**0.740 (0.088)**
VAR(1) - $\phi = 0.5$	ACF	0.015 (0.003)	**0.004 (0.003)**	**0.004 (0.002)**	0.006 (0.003)
	ACF^2	0.008 (0.000)	**0.002 (0.001)**	**0.002 (0.001)**	0.004 (0.003)
	CCF	0.190 (0.047)	**0.027 (0.007)**	**0.022 (0.011)**	**0.031 (0.012)**
	TV	0.069 (0.006)	**0.030 (0.003)**	**0.030 (0.003)**	**0.031 (0.008)**
	AUC	0.748 (0.021)	**0.630 (0.018)**	**0.631 (0.010)**	**0.624 (0.016)**
VAR(1) - $\phi = 0.9$	ACF	0.021 (0.007)	**0.009 (0.002)**	0.019 (0.010)	0.012 (0.006)
	ACF^2	0.021 (0.005)	**0.011 (0.002)**	0.016 (0.007)	**0.009 (0.005)**
	CCF	0.380 (0.121)	0.363 (0.124)	**0.045 (0.017)**	**0.048 (0.022)**
	TV	0.098 (0.016)	0.087 (0.030)	0.060 (0.024)	**0.050 (0.005)**
	AUC	0.916 (0.074)	0.892 (0.096)	0.678 (0.027)	**0.652 (0.026)**
VAR(1) - $\phi = 0.99$	ACF	0.028 (0.014)	0.277 (0.004)	0.025 (0.010)	**0.015 (0.011)**
	ACF^2	0.042 (0.015)	0.241 (0.001)	**0.027 (0.009)**	**0.020 (0.019)**
	CCF	0.478 (0.012)	**0.089 (0.047)**	0.199 (0.085)	0.208 (0.063)
	TV	0.257 (0.096)	0.809 (0.017)	0.189 (0.028)	**0.121 (0.045)**
	AUC	0.968 (0.013)	1.000 (0.000)	0.884 (0.061)	**0.813 (0.025)**
VAR(10)	ACF	0.161 (0.016)	0.212 (0.135)	0.105 (0.016)	**0.025 (0.004)**
	ACF^2	0.062 (0.005)	0.089 (0.087)	0.049 (0.005)	**0.009 (0.002)**
	CCF	0.103 (0.071)	0.134 (0.192)	0.073 (0.023)	**0.023 (0.007)**
	TV	0.043 (0.019)	0.107 (0.196)	**0.020 (0.006)**	**0.024 (0.002)**
	AUC	0.978 (0.020)	0.969 (0.030)	**0.913 (0.018)**	**0.919 (0.003)**

Table 2: Results on real datasets.

		Trunc-Sig	Rand-Sig	Cond-Sig	SOCK
Crypto	ACF (in)	0.070 (0.012)	0.187 (0.108)	0.237 (0.032)	**0.023 (0.000)**
	ACF^2 (in)	0.026 (0.002)	0.065 (0.061)	0.082 (0.017)	**0.025 (0.000)**
	CCF (in)	0.068 (0.022)	0.071 (0.062)	0.112 (0.106)	**0.046 (0.001)**
	TV (in)	0.207 (0.010)	0.252 (0.131)	0.142 (0.014)	**0.127 (0.004)**
	ACF (out)	0.064 (0.013)	0.186 (0.111)	0.237 (0.033)	**0.008 (0.001)**
	ACF^2 (out)	0.015 (0.002)	0.058 (0.064)	0.078 (0.016)	**0.011 (0.000)**
	TV (out)	0.179 (0.010)	0.217 (0.115)	**0.118 (0.016)**	**0.121 (0.002)**
	CCF (out)	0.074 (0.031)	0.060 (0.092)	0.111 (0.124)	**0.005 (0.004)**
	AUC (out)	0.688 (0.024)	0.866 (0.070)	0.975 (0.030)	**0.565 (0.009)**
5-Tickers	ACF (in)	0.095 (0.007)	0.286 (0.130)	0.098 (0.008)	**0.025 (0.000)**
	ACF^2 (in)	0.030 (0.001)	0.125 (0.066)	0.032 (0.003)	**0.025 (0.000)**
	CCF (in)	0.081 (0.008)	0.382 (0.210)	0.061 (0.008)	**0.040 (0.001)**
	TV (in)	0.147 (0.008)	0.336 (0.125)	0.134 (0.002)	**0.120 (0.001)**
	ACF (out)	0.095 (0.009)	0.290 (0.132)	0.097 (0.011)	**0.024 (0.001)**
	ACF^2 (out)	0.028 (0.002)	0.128 (0.068)	0.030 (0.004)	**0.023 (0.002)**
	TV (out)	0.157 (0.008)	0.334 (0.116)	0.137 (0.006)	**0.125 (0.007)**
	CCF (out)	0.091 (0.018)	0.399 (0.219)	0.083 (0.007)	**0.061 (0.003)**
	AUC (out)	0.571 (0.065)	0.895 (0.164)	0.597 (0.056)	**0.491 (0.086)**

Table 3: Comparison of resampling vs no resampling.

		Rand-Sig	Rand-Sig (resample)	SOCK	SOCK (resample)
VAR(10)	ACF	0.211 (0.132)	0.212 (0.135)	0.032 (0.006)	**0.025 (0.004)**
	ACF^2	0.086 (0.080)	0.089 (0.087)	0.013 (0.004)	**0.009 (0.002)**
	CCF	0.180 (0.169)	0.134 (0.192)	**0.025 (0.013)**	**0.023 (0.007)**
	TV	0.108 (0.167)	0.107 (0.196)	**0.024 (0.003)**	**0.024 (0.002)**
	AUC	0.972 (0.030)	0.969 (0.030)	0.936 (0.015)	**0.919 (0.003)**
Heston	ACF	0.180 (0.164)	0.093 (0.079)	**0.015 (0.011)**	**0.011 (0.008)**
	ACF^2	0.120 (0.072)	0.080 (0.066)	**0.016 (0.014)**	**0.013 (0.013)**
	CCF	0.521 (0.347)	**0.259 (0.214)**	**0.255 (0.417)**	**0.151 (0.181)**
	TV	0.360 (0.173)	0.307 (0.230)	**0.119 (0.066)**	**0.115 (0.057)**
	AUC	0.870 (0.114)	**0.773 (0.142)**	**0.744 (0.085)**	**0.740 (0.088)**

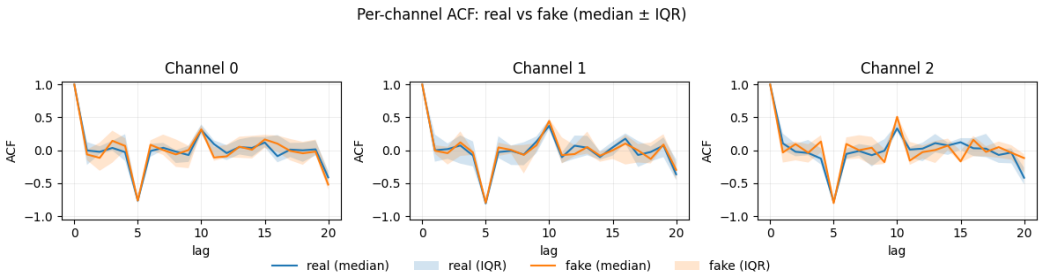

Figure 3: Comparison of the channel-wise autocorrelation of the real VAR(10) process and a generator trained with SOCK. Note that the generator is restricted to a $q = 5$-lagged Markov process, so even a perfectly trained generator could not exactly match the autocorrelation structure. Despite this limitation, the autocorrelation profile of the generated paths is very similar to the profile of the real paths.