Generating Financial Time Series by Matching Random Convolutional Features

Konrad J. Mueller[*†]
Imperial College London
JPMorgan Chase & Co.[‡]

Nikita Zozoulenko[*]
Imperial College London

Ben Wood
JPMorgan Chase & Co.[‡]

Thomas Cass
Imperial College London

Lukas Gonon
Imperial College London
University of St. Gallen

Abstract

Generating realistic financial time series is challenging as training data is often limited to a single historical path. With such scarce data, overfitting is hard to avoid, especially under adversarial training where a trained discriminator can memorize the training samples. To mitigate this, recent approaches train generators to minimize the discrepancy between untrained feature representations of real and generated time series. In these works, the feature maps are based on path signatures, which can fail to capture relevant time series properties at tractable truncation depths. In this work, we instead train generators by matching random convolutional features of real and generated time series. Existing random convolutional feature maps, such as Rocket and Hydra, have been shown to provide informative representations of real-world time series, but cannot supervise generative models because they are non-differentiable. We introduce **SOCK** (***SO**ft **C**ompeting **K**ernels*), a fully differentiable random convolutional feature map, suited to train generative time series models. We show that generators trained by matching random SOCK features outperform signature and diffusion baselines on most evaluation metrics across a wide range of small-sample financial datasets. We further demonstrate SOCK's expressiveness on two-sample hypothesis testing and time series classification tasks, where SOCK matches or outperforms existing unsupervised feature maps.

Code: https://github.com/konmue/sock

1 Introduction

Training generative time series models with limited real-world data is a fundamental challenge across many domains. In finance, time series datasets are often small in sample size: for instance, eight years of daily trading data provides about 2,000 data points. Because historical data is limited, synthetic time series are important for many applications, including estimation of risk measures [Cont et al., 2025] and policy optimization for investment [Mariani et al., 2019, Koshiyama et al., 2021] or hedging problems [Wiese et al., 2019, Buehler et al., 2020, Cont and Vuletić, 2025]. These applications require sampling future paths conditional on recent observations. We therefore study time series generation with two practical constraints: (i) only a single historical path is available for training, and (ii) generation is conditional on the q most recent observations.

[*]Equal contribution

[†]Corresponding author: k.mueller23@imperial.ac.uk

[‡]Opinions expressed in this paper are those of the authors, and do not necessarily reflect the view of JPMorgan Chase & Co.

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

Training generative models adversarially is often unstable with such limited data and can drive the generator to memorize training samples [Karras et al., 2020, Issa et al., 2023]. Instead, recent works propose training time series generators via feature matching under a fixed feature map [Ni et al., 2022, Biagini et al., 2024]. They fit a generative model p_θ to minimize the mean feature discrepancy

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

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. In contrast to (adversarial) feature-matching [Salimans et al., 2016, Mroueh et al., 2017], the feature map $f : \mathbb{R}^{T\times d} \to \mathbb{R}^n$ is fixed rather than learned, and thus cannot overfit to the training data. This makes such non-adversarial feature-matching well suited to the small-sample regime.

Motivated by rough path theory [Lyons, 1998], prior work chooses the path statistics $f(X)$ from the family of signature-based path-to-vector transformations. While the signature transform has strong theoretical properties, other, more heuristically motivated, unsupervised feature maps empirically outperform truncated signature features on small-sample time series classification problems [Middlehurst et al., 2024]. On these tasks, random convolutional feature maps such as Rocket [Dempster et al., 2020, Tan et al., 2022] and Hydra [Dempster et al., 2023] often yield the strongest classifiers.

In this paper, we show that the strong classification performance of random convolutional features transfers to generative modeling. Their randomness is especially useful for feature matching: the feature map can be resampled during generator training, so supervision does not rely on a single fixed representation. Existing random convolutional feature maps are not differentiable, so they cannot directly supervise generator training. We address this by introducing a new differentiable random convolutional feature map. Our contributions are:

1. **Random convolutional feature matching for conditional generation.** We train conditional time series generators by matching random convolutional feature statistics of real and generated paths. During training, we resample the random feature map, so the generator is supervised by a distribution of feature maps.
2. **Differentiable random convolutional features.** We introduce `SOCK`, a random convolutional feature map for multivariate time series that is fully differentiable w.r.t. the input path. `SOCK` builds on Hydra's random kernel competitions, but differs in how it preprocesses and convolves inputs, and pools the convolutional responses in a differentiable way.
3. **Evaluation under limited-data training.** We evaluate generator training techniques in a practical setting: training on a single historical path and evaluating out-of-sample. Across synthetic and real financial datasets, generators trained by matching random `SOCK` features achieve stronger overall performance than signature baselines and Diffusion-TS [Yuan and Qiao, 2024]. We further validate `SOCK`'s expressiveness on two-sample hypothesis testing and time-series classification tasks.

2 Background and related work

Signature-based path features The signature transform maps a continuous-time path to an infinite collection of iterated integrals, which can be viewed as statistics of the path. In practice, we can truncate the signature to compute a finite feature representation of a discrete-time path $X \in \mathbb{R}^{T\times d}$, which involves augmenting and interpolating X [McLeod and Lyons, 2025, Wu et al., 2021, Kiraly and Oberhauser, 2019]. Due to the polynomial growth of the signature in terms of the state-space dimension, the randomized signature is often preferred [Cuchiero et al., 2021, Cirone et al., 2023, Zozoulenko et al., 2025a].

Non-adversarial feature matching for time series Matching expected feature statistics is a common objective for training generative models across various data modalities. Existing methods either learn the feature map f (or a kernel) jointly with the generator [Salimans et al., 2016, Mroueh et al., 2017, Li et al., 2017, Sutherland et al., 2017], or fix f prior to training [Li et al., 2015, Dziugaite et al., 2015]. Ni et al. [2022] apply such non-adversarial feature matching to time series generation by using the truncated signature transform for f.

Ni et al. [2022] study unconditional generation, whereas many financial applications require generating future paths conditional on the recent past. Liao et al. [2024] train such conditional generators

with signature feature matching. Instead of directly applying eq. (1) to conditional samples, Liao et al. [2024] match conditional feature means. They estimate the conditional feature mean under the generated distribution by sampling multiple futures for each past segment, and approximate the real conditional mean with a linear predictor from the past segment's features (details in section C.4). Biagini et al. [2024] extend both the unconditional and conditional feature-matching objectives to randomized signature features. Empirically, this line of work reports strong performance of signature-based feature matching on small-sample financial datasets. For instance, Liao et al. [2024] report improvements over adversarial baselines [Esteban et al., 2017, Yoon et al., 2019] across several distributional metrics and financial datasets. These results motivate our work on training generators with untrained and randomized feature maps.

Random convolutional time series features In the small-sample time series classification literature [Middlehurst et al., 2024], the best performing classifiers most often take the form of a simple linear classifier on top of an untrained (unsupervised) high-dimensional feature map. For this purpose, random convolutional feature maps, such as the Rocket and Hydra families [Dempster et al., 2020, 2021, Tan et al., 2022, Dempster et al., 2023, 2024], perform particularly well. These compute features by convolving the input path with many random kernels and pooling the convolutional responses over time via non-differentiable operations such as argmax or indicator functions.

3 Random convolutional feature matching

3.1 Conditional time series generation from limited data

In financial applications of time series generative models, one often observes only a single realized path of the time series, but needs to simulate many plausible future scenarios conditional on the recent past. We study time series generation in this practically important setting. Concretely, our goal is to train a conditional generator that, given the most recent q observations of a time series, produces a distribution over the next T observations. Writing $x^- \in \mathbb{R}^{q\times d}$ for these recent observations, the generator defines a conditional distribution $p_\theta(\cdot \mid x^-)$ over length-T futures in $\mathbb{R}^{T\times d}$.

The practical constraints determine how we train and evaluate the generator. During training, only a single historical path $X_{1:H} \in \mathbb{R}^{H\times d}$ of length H is available. We evaluate the generator on the length-N out-of-sample continuation of this path, denoted $X_{H+1:H+N}$. At each $t \geq H$, we condition on the recent history x_t^- to generate a future $\hat{x}_t^+$ and compare it to the realized future x_t^+:

$$x_t^- := X_{t-q+1:t}, \qquad x_t^+ := X_{t+1:t+T}, \qquad \hat{x}_t^+ \sim p_\theta(\cdot \mid x_t^-).$$

We then form the joined past–future segments $x_t^- \oplus \hat{x}_t^+$ and $x_t^- \oplus x_t^+$ for each evaluation time, where $\oplus$ denotes concatenation along the time axis. We aggregate these segments over evaluation times t into empirical distributions of generated and realized segments. The generator is then evaluated by comparing these empirical path distributions; see table 4 for evaluation metrics.

On synthetic datasets, where the data-generating law is known, we sample J independent continuations $X^{(j)}_{H+1:H+N}$ for $j = 1, \ldots, J$. For each continuation, we form generated and realized joined segments as above and then aggregate those over both continuations j and times t. Multiple continuations increase the sample size of the empirical path distributions and reduce evaluation noise.

3.2 Training conditional time series generators with random feature matching

Training the conditional generator p_θ is challenging due to the limited training data. Adversarial methods are particularly prone to overfitting in this setting, since the training data contains only one realized future x_t^+ for each conditioning segment x_t^-. Instead, we train the generator p_θ by matching random convolutional features of real and generated paths. Our method builds on the signature matching approach of Ni et al. [2022], but differs in three respects. First, we replace signature features with random convolutional features. Second, we train against repeated draws of the random feature map, rather than a single fixed feature map. Third, we train a conditional generator by applying the feature map to joined past–future segments $x_t^- \oplus x_t^+$ and $x_t^- \oplus \hat{x}_t^+$.

Let $f_\psi : \mathbb{R}^{(q+T)\times d} \to \mathbb{R}^{d_f}$ be a feature map with random parameters $\psi \sim \Pi$. We define our choice of f_ψ, the SOCK feature map, in section 3.3. We train the generator parameters θ with the objective

$$\min_\theta \quad \mathbb{E}_{\psi\sim\Pi}\Big[\Big\|\mathbb{E}_t\big[f_\psi(x_t^- \oplus x_t^+)\big] - \mathbb{E}_{t,\,\hat{x}_t^+\sim p_\theta(\cdot|x_t^-)}\big[f_\psi(x_t^- \oplus \hat{x}_t^+)\big]\Big\|_2^2\Big], \tag{2}$$

where t is sampled uniformly from the valid training indices $\{q, \ldots, H-T\}$. The outer expectation over ψ trains the generator to match feature means across a distribution of feature maps.[4] This reduces the risk of supervisory blind spots of any single feature-map draw: discrepancies that are invisible to one draw of f_ψ may be detected by other independent draws.

In practice, we estimate the inner expectations by sampling B time indices $t_1, \ldots, t_B$ and generating a single continuation $\hat{x}_{t_i}^+ \sim p_\theta(\cdot \mid x_{t_i}^-)$ for each context $x_{t_i}^-$. We estimate the outer expectation over ψ with one feature-map draw at a time and keep this draw fixed for R optimization steps before resampling. The main additional cost of resampling is recomputing features of the real path segments, which could otherwise be cached for a fixed feature map. Resampling only every R steps amortizes this cost while still exposing the generator to many independent feature maps during training, which can substantially improve the performance of the trained generator (Appendix A.2).

The success of this approach depends on the random feature map f_ψ. Its feature means must reveal relevant discrepancies between real and generated paths, and the feature map must be differentiable with respect to its input so that gradients pass through generated paths to θ. Existing random convolutional features are highly expressive, but their pooling operations are typically non-differentiable. We therefore introduce a differentiable random convolutional feature map next.

3.3 Differentiable random convolutional features

We introduce SOCK, a differentiable random convolutional feature map for multivariate time series. SOCK maps a path to features in three steps: (i) preprocessing and augmentations, (ii) grouped random convolutions, and (iii) differentiable temporal pooling. The feature map is based on Hydra [Dempster et al., 2023], which first showed that competitions between random convolutional kernels provide useful features for classifying real-world time series. SOCK builds on the idea of kernel competitions, but differs from Hydra in several respects. Most importantly, SOCK supports multivariate input series and uses a differentiable temporal pooling.

Preprocessing. We augment, normalize, and randomly project the input path $X \in \mathbb{R}^{T\times d}$ to obtain

$$Y_t = P\,\mathcal{N}(\mathcal{A}(X))_t\,.$$

Here $\mathcal{A} : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d'}$ is an augmentation map, $\mathcal{N}$ denotes per-channel centering and scaling using statistics fitted on the augmented training paths, and $P \in \mathbb{R}^{M\times d'}$ is a random Gaussian matrix with ℓ_2-normalized rows. The augmentation map $\mathcal{A}$ preserves the original input path and appends derived channels. We make different augmentation choices in our generative and discriminative experiments. We find that appending either the integrated (`int`) or differenced (`diff`) path, as well as elementwise positive and negative parts of each channel (`posneg`) can sometimes improve performance. See Appendices B.1 and A.2 for details and ablations.

Grouped random convolutions. We partition the M channels of Y into G groups of width W and write $Y^{(g)} \in \mathbb{R}^{T\times W}$ for group g. Then, we convolve each $Y^{(g)}$ with K random kernels $\{w^{(g,k)}\}_{k=1}^K$ of shape $W \times L$ at dilation $\delta \geq 1$. For odd L, the convolution response at time t is

$$Z_t^{(g,k)} \;=\; \sum_{r=-(L-1)/2}^{(L-1)/2} \big\langle w_r^{(g,k)},\, Y_{t+\delta r}^{(g)} \big\rangle,$$

with zero padding when $t + \delta r \notin \{1, \ldots, T\}$. The kernels are sampled with i.i.d. Gaussian entries, then centered and ℓ_1-normalized, with full details given in Appendix B.

This step largely follows Hydra, including the default choices $L = 9$ and $K = 8$, but differs in the group width. Hydra uses channel-wise kernels ($W = 1$), whereas SOCK uses $W = 2$, so that each $Z^{(g,k)}$ can combine two projected channels. One difference between these choices is how constant shifts in input channels affect Z. Because each kernel's weights sum to zero, $Y^{(g)}$ and $Y^{(g)} + c\mathbf{1}$

[4] Similar objectives using randomly initialized neural networks have been used to directly optimize synthetic image data [Zhao and Bilen, 2023].

give the same response $Z^{(g,k)}$ away from zero-padding boundaries. For $W = 1$, this implies that each $Z^{(g,k)}$ is invariant to constant shifts in any channel of the sequence $\mathcal{N}(\mathcal{A}(X))$. For $W = 2$, such per-channel shifts generally change $Z^{(g,k)}$, because kernel centering only cancels joint shifts of the two channels in a group.

Soft competing kernels. To map the response series $Z^{(g,\cdot)}$ into a feature vector, we follow Hydra's interpretation of the K parallel convolutions as a competition between kernels: if $Z_t^{(g,k)} > Z_t^{(g,k')}$, kernel k outcompetes kernel k' as it better matches the local path around time t. Hydra summarizes these competitions by counting how often each kernel 'won' and by accumulating the corresponding winning values $Z_t^{(g,k)}$. These features capture which random kernel directions best match the path over time, but they depend on a hard selection over kernels and are thus not differentiable in Z.[5] In SOCK, we represent the competition at each time step by win probabilities

$$P_t^{(g,k)} = \frac{\exp(Z_t^{(g,k)}/\tau)}{\sum_{j=1}^{K} \exp(Z_t^{(g,j)}/\tau)},$$

which are differentiable with respect to the responses $Z_t^{(g,k)}$ for any temperature $\tau > 0$. We find that several ways of aggregating the series $P^{(g,k)}$ over time produce useful features. In our experiments, we use the *soft-deviation* feature, the standard deviation of the win probabilities over time:

$$F^{(g,k)} = \texttt{std}_{t=1,\ldots,T}\big(P_t^{(g,k)}\big).$$

This statistic $F^{(g,k)}$ measures how much the win probability of kernel k in group g varies over time. We also consider differentiable generalizations of Hydra's count and value features, obtained by averaging $P_t^{(g,k)}$ or $P_t^{(g,k)} Z_t^{(g,k)}$ over time (Appendix B.3). Among individual pooling functions, soft-deviation performs best overall (see right panel of fig. 4). Concatenating multiple statistics can further improve performance in some settings (see discussion in Appendix D.2).

Following Hydra, we capture time series properties at multiple time scales by applying the pipeline $Y \mapsto Z \mapsto F$ to the same Y for different dilations $\delta \in \mathcal{D} := \{2^e : e = 0, 1, \ldots, e_{\max}\}$, with independently sampled kernels per δ.[6] We concatenate features across dilations into a feature vector of dimension $d_f = |\mathcal{D}|GK$. We provide additional details and pseudocode for SOCK in Appendix B.

4 Conditional time series generation experiments

We now consider the small-data, conditional generation task of section 3.1. We compare training generative models by matching random SOCK features to signature-matching and diffusion baselines.

Synthetic datasets We first evaluate the generative modeling techniques on five synthetic datasets spanning diverse time series characteristics (table 1). The datasets V1 and V10 are sampled from vector autoregressive processes, while TGH applies a Tukey g–h marginal transform to V10 to induce skewness and heavy tails [Yan and Genton, 2019]. The SV dataset consists of the log-return process and log-variance process, simulated under the stochastic volatility model of Heston [1993]. The dataset FGN is a multivariate fractional Gaussian noise process, with independent components and component-specific Hurst exponents. For each dataset, we sample a single training path of length $H = 2048$ (roughly eight years of trading days), and $J = 2048$ independent out-of-sample continuations of length $N = 2048$. We repeat this protocol for five seeds, resampling all paths each time. This reduces evaluation noise caused by the limited training data.

Table 1: Synthetic datasets and their key properties.

Dataset	d	Order	Key phenomena
V1	3	1	strong autocorrelation ($\phi = 0.99$)
V10	3	10	oscillatory autocorrelation
TGH	3	10	skewed + heavy tail marginals
SV	2	1	cross-correlation (spot-vol), skew
FGN	3	∞	long memory; $\mathcal{H} \in \{.05, .15, .25\}$

Real datasets We further evaluate on seven real financial datasets spanning multiple asset classes. We include four sector-based stock datasets (FI, PH, IT, ST), each a 4-dimensional time series of daily

[5] Hydra uses argmax for counts, argmin for value features.

[6] The maximum exponent $e_{\max}$ depends on the input length (Appendix B).

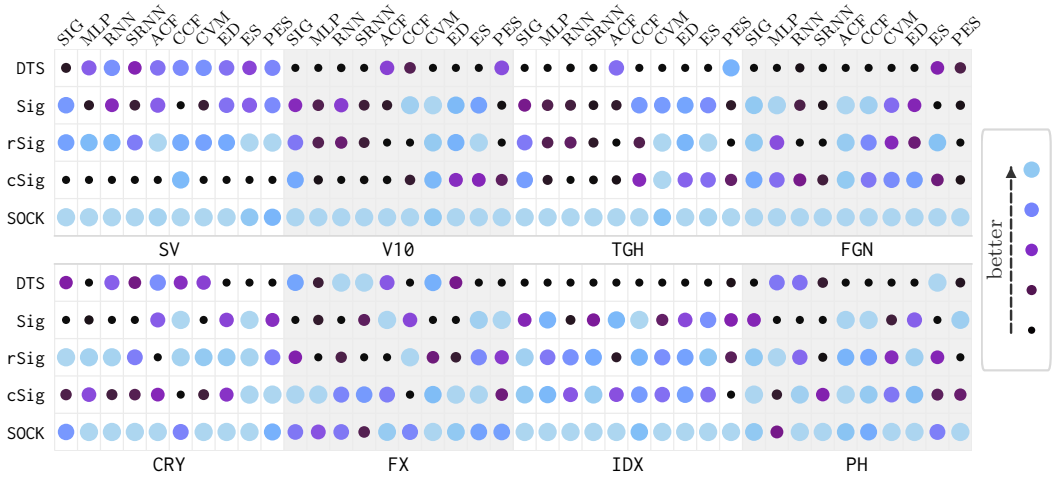

Figure 1: Summary of evaluation metrics across selected synthetic (SV, V10, TGH, FGN) and real datasets (CRY, FX, IDX, PH). Dot color and size encode the metric value: lighter and larger dots indicate a smaller discrepancy between real and generated data (hence better).

log-returns of four large-cap stocks from the same sector. We further include two 3-dimensional daily datasets: IDX, comprising S&P 500, VIX, and gold futures returns, and FX, comprising EUR/USD, JPY/USD, and GBP/USD returns. Finally, we include CRY, which consists of 5-minute returns of BTC and ETH. The daily time series cover 2009–2025. We use the first 8 years for training and the last 8 years for evaluation (details on datasets in Appendix C.1).

Rollout length We generate paths of length $T = 64$ (roughly 3 months of trading days), which is a relevant horizon for mid-term risk management applications. This is more challenging than the short horizons $T \in \{3, 5, 10\}$ used in prior work [Liao et al., 2024, Lou et al., 2024, Biagini et al., 2024], as even small modeling errors visibly compound over longer rollouts. We use a short context length $q = 5$ (one week of trading days), to limit overfitting.

Evaluation metrics To evaluate each generator, we compare the empirical distributions of real and generated out-of-sample paths (section 3.1). We report several discriminative and distributional metrics; all defined so that lower values indicate better fit. Discriminative metrics fit binary classifiers to distinguish real from generated paths and report $|\text{accuracy} - 0.5|$. We use linear classifiers on signature features (SIG) and random MLP features (MLP), and GRU classifiers applied either to individual paths (RNN; Yoon et al., 2019) or sets of paths (SRNN). Distributional metrics compare autocorrelation (ACF), cross-correlation (CCF), marginals (CVM) and their tails (ES), joint distributions of consecutive observations (ED), and trading-strategy PnL tails (PES; Cont et al., 2025). Definitions of all metrics are in section C.2.

4.1 Generative models

We compare several methods for training generative models and train separate models for each dataset under five random seeds. On synthetic datasets, each seed resamples the training and evaluation data, and also changes the training randomness. On real datasets, we cannot resample the data, so only the training randomness varies across seeds.

We train three generative models via feature matching on joined past–future segments (section 3.2) using three different feature maps: truncated signature Sig [Ni et al., 2022], randomized signature rSig [Biagini et al., 2024], and SOCK. For the random feature maps SOCK and rSig, we resample parameters every $R = 100$ steps; for Sig the feature map is fixed throughout training. Further, we benchmark against the Sig-Wasserstein GAN [Liao et al., 2024], which uses conditional feature matching with signature features (cSig).

For the truncated and randomized signature features, we augment paths before computing the signature, which significantly improves their performance. We first apply the Lead-Lag transform,

then augment the path with a time channel, and finally apply the I-visibility transformation [Wu et al., 2021]. For `Sig` and `cSig` we use a truncation depth of 3 [Liao et al., 2024] and for `rSig` we use a hidden dimension of 128 and the tanh activation function. For `SOCK`, we augment with `int` and `posneg` (Appendix B.1), and set $\tau = 0.1$, $M = 256$. With `SOCK`, we apply an additional componentwise feature scaling before computing feature means (Appendix C.3.1). For `Sig` and `rSig`, we found that such a standardization is not beneficial (table 5). We use the above choices on all datasets.

For these four training methods, we parameterize $p_\theta(\cdot \mid x^-)$ with the same GRU-based generator [Chung et al., 2014]. Given a context x^-, we draw $\varepsilon_0 \sim \mathcal{N}(0, I_d)$ and use an MLP on (x^-, ε_0) to produce the initial GRU state. We then draw an i.i.d. noise sequence $\varepsilon_{1:T} \sim \mathcal{N}(0, I_d)$ and map it directly to a generated sample path using a single-layer GRU-based decoder (Appendix C.3.2).

Additionally, we compare to Diffusion-TS (`DTS`) [Yuan and Qiao, 2024], a diffusion model for time series. Diffusion-TS supports conditional generation by keeping the conditioning segment x^- fixed and denoising the masked future segment of length T. We use the authors' implementation and recommended hyperparameters (Appendix C.4).

4.2 Generative Modeling Results

Figure 1 visualizes the evaluation metrics for the generated distributions on 8 of the 12 datasets. Lighter, larger dots indicate smaller metric values and thus smaller discrepancies between real and generated paths. Full numerical results for all datasets are provided in Figures 5-7 of Appendix A.

We find that **generators trained to match random `SOCK` features achieve the strongest overall performance** across datasets, outperforming the signature baselines and Diffusion-TS on most metrics. Quantitatively, `SOCK` attains the best average rank (1.49) over all dataset–metric pairs, followed by `rSig` (2.80). On synthetic datasets, `SOCK` outperforms all baselines by a wide margin on most metrics. On real datasets, `SOCK` also performs best overall, though not uniformly and with smaller margins. The smaller gap between methods on real datasets likely reflects a combination of typically weak autocorrelation in daily returns, noisier evaluation, and possible distributional drift over the 8-year horizon. In practice, such drift can be partially mitigated by regularly retraining the generator. We use the 8-year horizon to obtain sufficient samples for a stable distributional evaluation.

Because results across datasets and metrics are correlated, these aggregate ranks should be interpreted with care; standard significance tests assuming independence are not appropriate here. Instead, we devote the remainder of this section to analyzing the generative models at the level of individual metrics and datasets.

Marginal fit Across datasets, `SOCK` performs best overall on the marginal-fit metric CVM (Appendix A). As an example, the top left panel of fig. 2 shows KDEs for the two channels of the `SV` dataset (log-returns and log-variance). The log-variance marginal differs noticeably in-sample versus out-of-sample, illustrating how the limited training data provides a noisy view of the true data distribution. Despite this gap, the `SOCK`-trained generator matches the out-of-sample marginals well when generating conditionally from out-of-sample contexts x_t^-. This indicates that the generator captures the dependence structure of the process and does not merely resample the empirical marginal from the training path (see also the context-shuffling ablation in table 6). All methods reproduce the return density reasonably well, but differences are clearer for log-variance: `SOCK` is closest to the out-of-sample marginal, while `DTS` deviates substantially.

Autocorrelation Among the compared generators, `SOCK`-trained models most reliably reproduce autocorrelation structures. On the oscillatory datasets `V10` and `TGH`, only `SOCK` and `DTS` manage to reproduce the distinct autocorrelation pattern as shown in the top right panel of fig. 2. `SOCK` also best captures the anti-persistent dependencies of the fractional Gaussian noise process (bottom left of fig. 2). On the real daily-return datasets, autocorrelation is typically weak, making differences between methods less pronounced.

Tail fit For financial applications, it is important that the generator produces realistic tail-risk scenarios. Which tail events matter most depends on the precise application. Following Cont et al. [2025], we compare the tails of profit-and-loss (PnL) distributions obtained by applying simple trading strategies to real and generated data. Such PnL tails are sensitive to a range of distributional

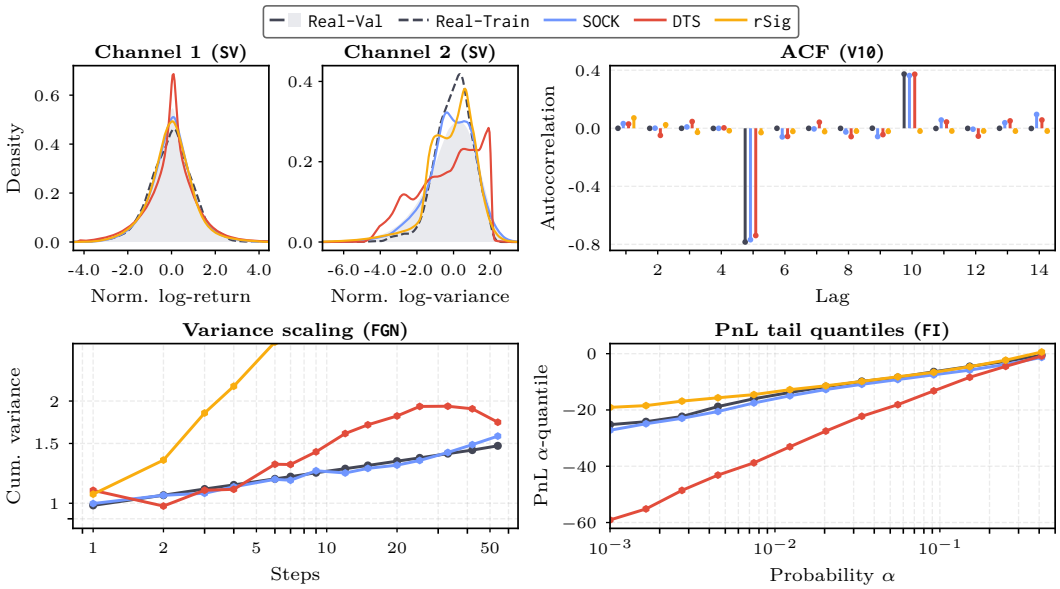

Figure 2: Comparison of real and generated distributions; visual similarity to real distributions indicates high generative fidelity. Top-left: marginal KDEs for the two SV channels. Top-right: ACF of the first V10 channel; the real process has nonzero autocorrelation at every fifth lag. Bottom-left: variance of cumulative FGN process $\sum_{i=1}^{\tau} X_i$ (log–log scale); slope on real data is $2\mathcal{H}$, where $\mathcal{H} = 0.05$. Bottom-right: left-tail PnL quantiles from a momentum trading strategy on FI.

path properties, including auto- and cross-correlation structure. The PES metric summarizes such tail discrepancies (Appendix C.2) and the bottom right panel of fig. 2 shows the PnL tail of a momentum strategy on the FI dataset. Both SOCK and rSig match the PnL tails closely, while DTS overestimates the tail heaviness.

Ablations We study how sensitive generator training by random SOCK feature matching is to feature-map resampling and to architectural choices in SOCK (augmentations, kernel width, and poolings). We run these ablations on the TGH dataset and report results in Appendix A.2. We find that resampling the random parameters of the feature map consistently improves performance for both SOCK and rSig. The gains from resampling are especially large for the weaker feature map rSig, while SOCK feature matching already trains good generative models without resampling. Regarding SOCK's architectural choices, such as augmentations, kernel width, and pooling, our default choices give the best RNN score, but the gaps to alternative choices are small and the ranking varies across metrics. This shows that random convolutional feature matching is robust to these choices. We next benchmark the SOCK feature map and its architectural choices on purely discriminative tasks.

5 Discriminative evaluation

5.1 Hypothesis testing

Following prior work [Chevyrev and Oberhauser, 2022, Salvi et al., 2021, Lou et al., 2023], we perform a series of two-sample hypothesis tests for stochastic processes, testing the null hypothesis that two stochastic processes, X_{β_0} and X_{β_1}, have the same law. Specifically, we use the maximum mean discrepancy (MMD) [Gretton et al., 2012] based on either explicit feature vectors or time series kernels, within a permutation test (see section D.1). Such hypothesis tests serve as a diagnostic for feature matching with the same feature map (or kernel): if the induced MMD is insensitive to a given mismatch between X_{β_0} and X_{β_1}, then minimizing eq. (1) cannot correct that mismatch.

We consider three tests that probe different time series characteristics, using the same parametric processes as in section 4. For each process, we vary a single parameter: TGH (tail heaviness h), SV (correlation ρ), and FBM[7] (Hurst $\mathcal{H}$). As explicit feature baselines we include truncated signatures

[7]For consistency with prior work, we use fractional Brownian motion (FBM) rather than FGN.

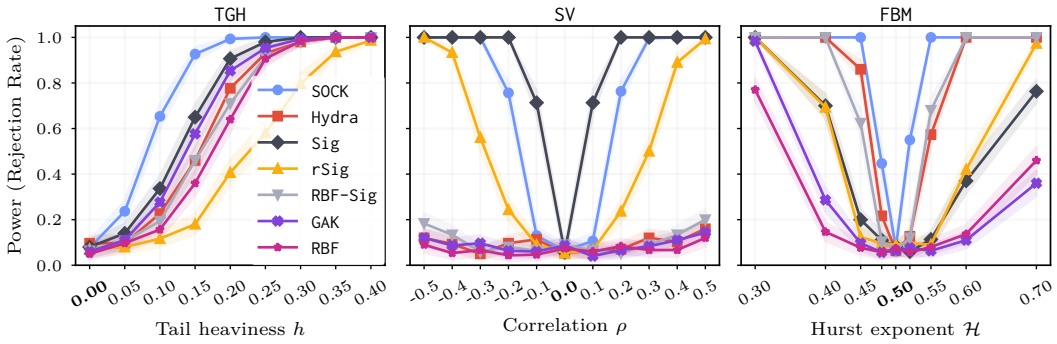

Figure 3: Permutation test power at significance level $\alpha = 0.05$ for feature- and kernel-based MMDs. Bold x-axis values indicate the parameter value under the null hypothesis H_0. Away from H_0, higher power means that the MMD more reliably detects the parameter change (hence better).

[Ni et al., 2022], randomized signatures [Cuchiero et al., 2021], and Hydra [Dempster et al., 2023]. Kernel baselines include the truncated RBF signature kernel [Kiraly and Oberhauser, 2019], the Global Alignment Kernel (GAK) [Cuturi et al., 2007, Cuturi, 2011], and the classical RBF kernel. Results are shown in fig. 3, with further details in section D.1. SOCK achieves the highest test power on FBM and TGH, and ranks second on SV. Notably, SOCK performs well on SV, a task where most baselines, including Hydra, barely outperform random guessing. Additionally, SOCK is robust to hyperparameter tuning, in comparison to signatures which are very sensitive to hyperparameters.

5.2 Time series classification

To further assess the expressivity of SOCK features, we evaluate them on the 112 univariate time series classification datasets of the UCR repository [Middlehurst et al., 2024]. Following prior works [Dempster et al., 2020, Tan et al., 2022, Dempster et al., 2023], we fix the classification pipeline to a feature extraction step followed by a RidgeCV classifier. First, we compare the poolings from Rocket, MultiRocket, and Hydra to three differentiable poolings: soft-deviation, the SOCK default, and soft-count and soft-value, differentiable generalizations of Hydra's count and value poolings (Appendix B.3). This is done by keeping the preprocessing pipeline and random kernels fixed, while varying the temporal pooling operator. The right panel of fig. 4 shows that classifiers trained with soft-deviation features consistently outperform other popular poolings across feature dimensions. Comprehensive experimental details, including setup, baselines, and robustness ablations (on random bias, temperature τ, number of kernels k), are provided in Appendix D.2.

Second, we compare SOCK against state-of-the-art models identified in Middlehurst et al. [2024] via a critical difference diagram in the left panel of fig. 4. We find that SOCK achieves the best average rank among the compared models. SOCK achieves this using an order of magnitude fewer random features and only a single pooling operation, compared to MultiRocket's four (details in Appendix D.2). These results show that SOCK performs strongly in purely discriminative settings and support its use as a feature map for generative modeling.

6 Conclusions and limitations

In this work, we studied time series generation in a limited-data setting relevant to many practical use cases in finance. We showed that conditional generative models trained by random convolutional feature matching achieve high generative fidelity, and outperform signature-based feature matching and Diffusion-TS on most metrics across small-sample synthetic and real financial datasets. To enable this approach, we introduced SOCK, a differentiable random convolutional feature map based on soft kernel competitions. We showed that SOCK also performs well on discriminative tasks, matching or outperforming existing random convolutional feature maps.

Motivated by our target applications, we focus on low-dimensional datasets with a single historical path. While random convolutional feature matching can be applied to larger datasets, we do not study that regime. Learned discriminators and diffusion models are likely to benefit more from additional

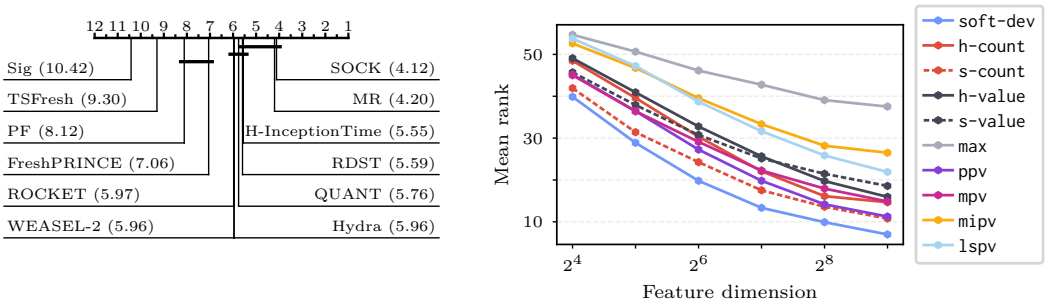

Figure 4: Results on UCR classification tasks. Left: critical difference diagram for classifiers trained on different feature maps. Right: mean rank for classifiers using the `SOCK` backbone with different poolings. We call Hydra's poolings `h-count` and `h-value`; `s-count` and `s-value` are differentiable versions of those; `soft-dev` (ours); `max`, `ppv` (Rocket); `ppv`, `mpv`, `mipv`, `lspv` (MultiRocket).

data than random feature maps. Moreover, we use a deliberately simple generator architecture for all feature matching methods. Stronger architectures could improve performance independently of feature-map improvements. Finally, the strong results on the UCR datasets motivate future work on `SOCK` for time series classification.

Acknowledgements

KM is supported by JPMorgan Chase & Co. through the EPSRC Centre for Doctoral Training in Mathematics of Random Systems: Analysis, Modelling and Simulation (EPSRC Grant EP/S023925/1). NZ has been supported by the Roth Scholarship at Imperial College London. The work of TC was supported in part by UK Research and Innovation (UKRI) through the Engineering and Physical Sciences Research Council (EPSRC) via Programme Grants [Grant No. UKRI1010: High order mathematical and computational infrastructure for streamed data that enhance contemporary generative and large language models]. We acknowledge computational resources and support provided by the Imperial College Research Computing Service (DOI: `10.14469/hpc/2232`). For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

A preliminary version of this work appeared at the NeurIPS 2025 Workshop on Generative AI in Finance [Zozoulenko et al., 2025b].

References

Erdinc Akyildirim, Matteo Gambara, Josef Teichmann, and Syang Zhou. Applications of signature methods to market anomaly detection. *Preprint, arXiv 2201.02441*, 2022.

Erdinç Akyildirim, Matteo Gambara, Josef Teichmann, and Syang Zhou. Randomized signature methods in optimal portfolio selection. *Quantitative Finance*, 25(2):197–216, 2025.

Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, 17(5):1–10, 2016.

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

Mark Broadie and Özgür Kaya. Exact Simulation of Stochastic Volatility and Other Affine Jump Diffusion Processes. *Operations Research*, 54(2):217–231, 2006.

Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, 2016.

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

Hans Buehler, Blanka Horvath, Terry J. Lyons, Imanol Pérez Arribas, and Ben Wood. A Data-driven Market Simulator for Small Data Environments. *ArXiv preprint arXiv:2006.14498*, 2020.

Thomas Cass and William F. Turner. Free probability, path developments and signature kernels as universal scaling limits. *The Annals of Applied Probability*, 36(2):1082–1109, 2026.

Minshuo Chen, Renyuan Xu, Yumin Xu, and Ruixun Zhang. Diffusion factor models: Generating high-dimensional returns with factor structure. *Preprint, arXiv 2504.06566*, 2025.

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

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555, 2014.

Nicola Muca Cirone, Antonio Orvieto, Benjamin Walker, Cristopher Salvi, and Terry Lyons. Theoretical foundations of deep selective state-space models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Nicola Muça Cirone, Maud Lemercier, and Cristopher Salvi. Neural signature kernels as infinite-width-depth-limits of controlled resnets. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Samuel N. Cohen, Derek Snow, and Lukasz Szpruch. *Black-Box Model Risk in Finance*, pages 687–717. Cambridge University Press, 2023.

Enea Monzio Compagnoni, Anna Scampicchio, Luca Biggio, Antonio Orvieto, Thomas Hofmann, and Josef Teichmann. On the effectiveness of randomized signatures as reservoir for learning rough dynamics. In *IJCNN*, pages 1–8, 2023.

Rama Cont and Milena Vuletić. Data-driven hedging with generative models. *Annals of Operations Research*, 2025.

Rama Cont, Mihai Cucuringu, Renyuan Xu, and Chao Zhang. Tail-GAN: Learning to Simulate Tail Risk Scenarios. *Management Science*, 2025.

Christa Cuchiero and Janka Möller. Signature methods in stochastic portfolio theory. *SIAM Journal on Financial Mathematics*, 16(4):1239–1303, 2025.

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

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

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.

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.

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. Association for Computing Machinery, 2021.

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, 2023.

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. Springer-Verlag, 2024.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. Training generative neural networks via Maximum Mean Discrepancy optimization. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pages 258–267. AUAI Press, 2015.

Holger Englisch, Thomas Krabichler, Konrad J. Müller, and Marc Schwarz. Deep treasury management for banks. *Frontiers in Artificial Intelligence*, 6, 2023.

Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. *CoRR*, abs/1706.02633, 2017.

Xuefeng Gao, Mengying He, Xuedong He, and Jiale Zha. Factor-based conditional diffusion model for contextual portfolio optimization. *Preprint, arXiv 2509.22088*, 2025.

Salvador García and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9(89):2677–2694, 2008.

Michael B. Gordy and Sandeep Juneja. Nested Simulation in Portfolio Risk Measurement. *Manag. Sci.*, 56(10): 1833–1848, 2010.

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.

Guangyi He, Tobias Sutter, and Lukas Gonon. Distributional adversarial attacks and training in deep hedging. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 131893–131925, 2025.

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.

Zacharia Issa, Blanka Horvath, Maud Lemercier, and Cristopher Salvi. Non-adversarial training of Neural SDEs with signature kernel scores. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Adam C. Jones, Blanka Horvath, Christoph Reisinger, Ben Wood, Lianjun Bai, and Amira Akkari. Ambiguity-Averse Deep Hedging with Feature Clustering. *SSRN preprint*, 2025. URL https://papers.ssrn.com/abstract=5390563.

Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 12104–12114, 2020.

Patrick Kidger and Terry Lyons. Signatory: Differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations (ICLR)*, 2021.

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

Adriano Koshiyama, Nick Firoozye, and Philip Treleaven. Generative adversarial networks for financial trading strategies fine-tuning and combination. *Quantitative Finance*, 21(5):797–813, 2021.

Thomas Krabichler and Josef Teichmann. A case study for unlocking the potential of deep learning in asset-liability-management. *Frontiers in Artificial Intelligence*, 6, 2023.

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards Deeper Understanding of Moment Matching Network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2203–2213, 2017.

Siran Li, Zijiu Lyu, Hao Ni, and Jiajie Tao. Restricted path characteristic function determines the law of stochastic processes. *ArXiv preprint arXiv:2404.18661*, 2024.

Yujia Li, Kevin Swersky, and Richard S. Zemel. Generative Moment Matching Networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1718–1727. JMLR.org, 2015.

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.

Yannick Limmer and Blanka Horvath. Robust Hedging GANs: Towards Automated Robustification of Hedging Strategies. *Applied Mathematical Finance*, 31(3):164–201, 2024.

Hang Lou, Siran Li, and Hao Ni. Pcf-gan: generating sequential data via the characteristic function of measures on the path space. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 39755–39781, 2023.

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

Eva Lütkebohmert, Thorsten Schmidt, and Julian Sester. Robust deep hedging. *Quantitative Finance*, 22(8): 1465–1480, 2022.

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

Benoit B. Mandelbrot and John W. Van Ness. Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review*, 10(4):422–437, 1968.

Giovanni Mariani, Yada Zhu, Jianbo Li, Florian Scheidegger, Roxana Istrate, Costas Bekas, and A. Cristiano I. Malossi. PAGAN: Portfolio Analysis with Generative Adversarial Networks, 2019.

Andrew McLeod and Terry Lyons. Signature methods in machine learning. *EMS Surveys in Mathematical Sciences*, 2025. Published online first.

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.

Youssef Mroueh, Tom Sercu, and Vaibhava Goel. McGan: Mean and Covariance Feature Matching GAN. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2527–2535. PMLR, 2017.

Konrad Mueller, Amira Akkari, Lukas Gonon, and Ben Wood. Fast Deep Hedging with Second-Order Optimization. In *Proceedings of the 5th ACM International Conference on AI in Finance*, ICAIF '24, pages 319–327. Association for Computing Machinery, 2024.

Phillip Murray, Ben Wood, Hans Buehler, Magnus Wiese, and Mikko Pakkanen. Deep Hedging: Continuous Reinforcement Learning for Hedging of General Portfolios across Multiple Risk Aversions. In *ACM International Conference on AI in Finance (ICAIF)*, pages 361–368. ACM, 2022.

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. Association for Computing Machinery, 2022.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2226–2234, 2016.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Bernhard Schäfl, Lukas Gruber, Johannes Brandstetter, and Sepp Hochreiter. G-signatures: Global graph propagation with randomized signatures. *Preprint, arXiv 2302.08811*, 2023.

Danica J. Sutherland, Hsiao-Yu Tung, Heiko Strathmann, Soumyajit De, Aaditya Ramdas, Alexander J. Smola, and Arthur Gretton. Generative Models and Model Criticism via Optimized Maximum Mean Discrepancy. In *International Conference on Learning Representations (ICLR)*, 2017.

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, 2022.

Jiajie Tao, Hao Ni, and Chong Liu. High rank path development: an approach to learning the filtration of stochastic processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 115309–115350, 2024.

John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley Series in Behavioral Science : Quantitative Methods. Addison-Wesley, 1977.

Magnus Wiese and Phillip Murray. Risk-Neutral Market Simulation. *ArXiv preprint arXiv:2202.13996*, 2022.

Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. Deep Hedging: Learning to Simulate Equity Option Markets, 2019.

Magnus Wiese, Ben Wood, Alexandre Pachoud, Ralf Korn, Hans Buehler, Phillip Murray, and Lianjun Bai. Multi-Asset Spot and Option Market Simulation, 2021.

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.

Yuan Yan and Marc G. Genton. Non-Gaussian autoregressive processes with Tukey g-and-h transformations. *Environmetrics*, 30(2):e2503, 2019.

Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series Generative Adversarial Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5509–5519, 2019.

Xinyu Yuan and Yan Qiao. Diffusion-TS: Interpretable Diffusion for General Time Series Generation. In *International Conference on Learning Representations (ICLR)*, 2024.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3391–3401, 2017.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2023, Waikoloa, HI, USA, January 2-7, 2023*, pages 6503–6512. IEEE, 2023.

Nikita Zozoulenko, Thomas Cass, and Lukas Gonon. Infinite-dimensional mahalanobis distance with applications to kernelized novelty detection. *Journal of Machine Learning Research*, 26(247):1–47, 2025a.

Nikita Zozoulenko, Konrad Jakob Mueller, Thomas Cass, and Lukas Gonon. Generating time series by matching random convolutional features. In *NeurIPS 2025 Workshop on Generative AI in Finance*, 2025b.

Appendix

A Detailed results and ablations for generative experiments **16**

A.1 Detailed numerical results 16

A.2 Ablations 19

A.3 Additional visualizations 20

B Additional details and pseudocode for SOCK **21**

B.1 Augmentations and normalization 21

B.2 Random parameters and dilations 21

B.3 Pooling operators 22

B.4 Pseudocode 23

C Additional details on time series generation experiments **24**

C.1 Datasets 24

C.2 Evaluation 25

C.2.1 Evaluation protocol 25

C.2.2 Discriminative evaluation metrics 26

C.2.3 Distributional evaluation metrics 27

C.3 Additional training details 28

C.3.1 Data and feature scaling 28

C.3.2 Generator architecture 28

C.3.3 Optimization 29

C.4 Additional details for the baselines 30

C.4.1 Truncated signature (Sig) 30

C.4.2 Randomized signature (rSig) 30

C.4.3 Conditional Sig-Wasserstein GAN (cSig) 31

C.4.4 Diffusion-TS (DTS) 31

D Additional details and ablations on discriminative experiments **33**

D.1 Hypothesis testing 33

D.2 Time series classification 33

A Detailed results and ablations for generative experiments

A.1 Detailed numerical results

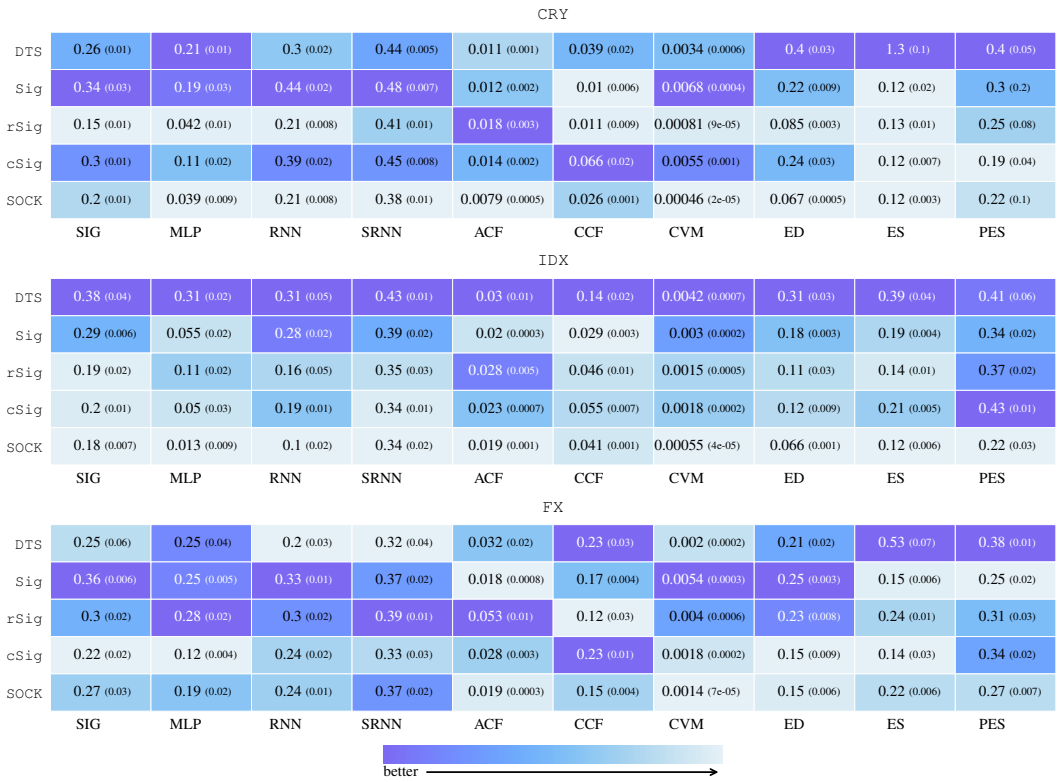

Figure 5: Detailed results on the real datasets CRY, IDX, and FX. Each cell reports the mean metric value across 5 seeds (standard deviation in parentheses). Here, seeds vary only the training randomness. Cell color encodes the mean value (lighter is better). The color is based on the averaged metrics.



Figure 6: Detailed results on the real baskets-of-stocks datasets. Each cell reports the mean metric value across 5 seeds (standard deviation in parentheses). Here, seeds vary only the training randomness. Cell color encodes the mean value (lighter is better). The color is based on the averaged metrics.

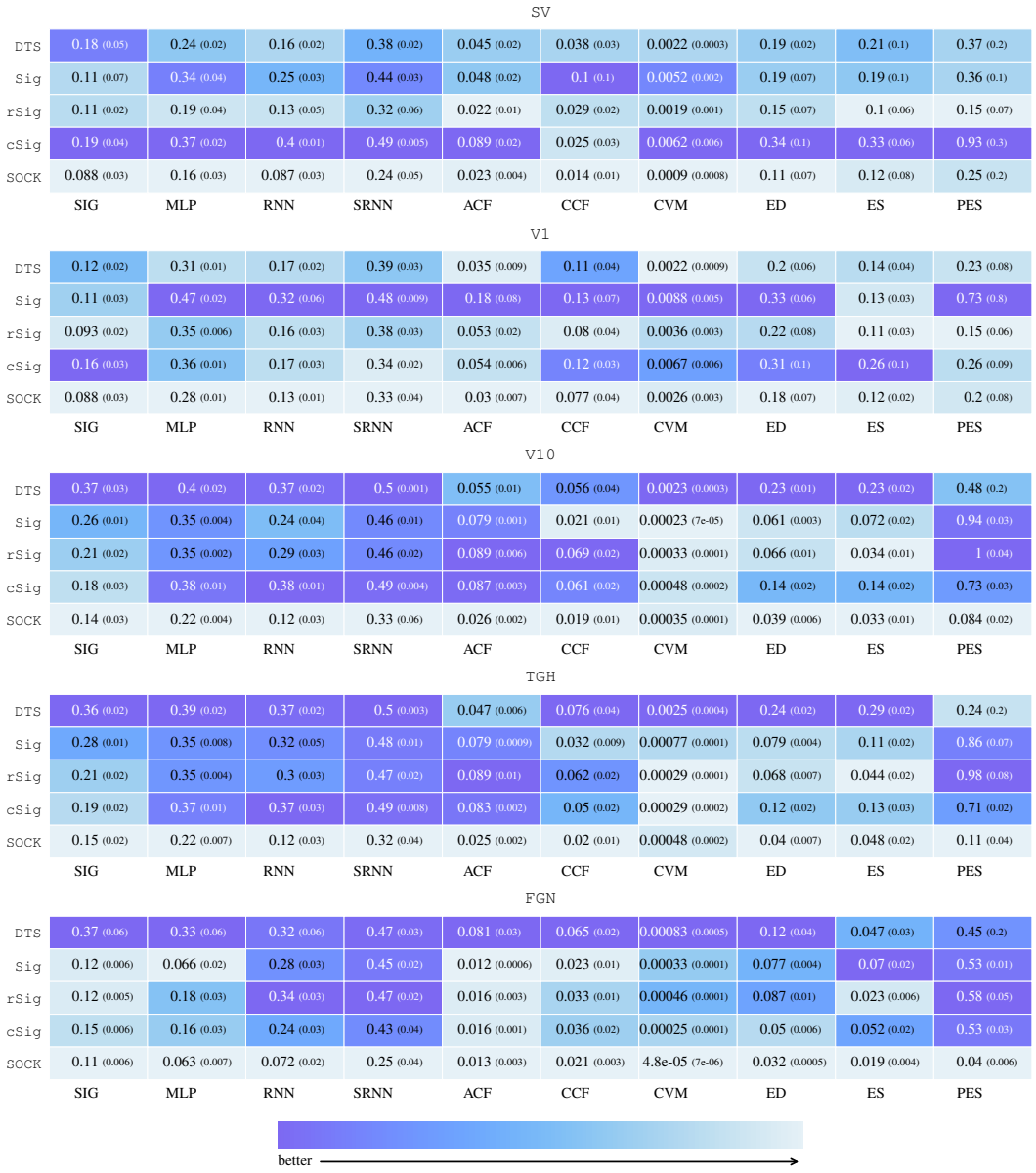

Figure 7: Detailed results on synthetic datasets. Each cell reports the mean metric value across 5 seeds (standard deviation in parentheses), where seeds vary both the data draw and training randomness. Cell color encodes the mean value (lighter is better). The color is based on the averaged metrics.

A.2 Ablations

Table 2: Ablations for the generative experiments. All ablations are run on the TGH dataset and repeated over five random seeds. Reported metrics are averaged over seeds; lower is better, and bold marks the best value within each block. Metric definitions are given in Appendix C.2. We report RNN and distributional metrics.

ABLATION 1: RESAMPLE METHOD	RNN	ACF	CCF	CVM	ED	ES
SOCK (RESAMPLE ×)	0.126	0.027	**0.019**	0.00145	0.041	0.051
SOCK (RESAMPLE ✓)	**0.117**	**0.025**	0.020	**0.00048**	**0.040**	**0.048**
rSig (RESAMPLE ×)	0.416	0.093	0.072	0.00249	0.172	0.080
rSig (RESAMPLE ✓)	**0.296**	**0.089**	**0.062**	**0.00029**	**0.068**	**0.044**
ABLATION 2: AUGMENTATIONS AUGMENTATIONS	RNN	ACF	CCF	CVM	ED	ES
$\mathcal{A}(X) = X$	0.127	**0.021**	**0.018**	0.00062	0.042	0.054
$\mathcal{A}(X) = \texttt{int}(X)$	0.130	0.023	**0.018**	**0.00043**	0.042	0.066
$\mathcal{A}(X) = \texttt{posneg}(\texttt{int}(X))$	**0.117**	0.025	0.020	0.00048	**0.040**	**0.048**
ABLATION 3: KERNEL WIDTH KERNEL WIDTH W	RNN	ACF	CCF	CVM	ED	ES
W=1	0.119	**0.022**	**0.017**	**0.00046**	0.042	0.050
W=2	**0.117**	0.025	0.020	0.00048	**0.040**	**0.048**
ABLATION 4: POOLING POOLING	RNN	ACF	CCF	CVM	ED	ES
SOFT-DEV	**0.117**	0.025	**0.020**	0.00048	**0.040**	0.048
SOFT-COUNT	0.132	**0.024**	0.024	**0.00027**	0.041	0.048
SOFT-VALUE	0.119	**0.024**	0.021	0.00050	**0.040**	**0.043**

Remarks. Ablation 1 compares training with a fixed random feature map against resampling the feature map every $R = 100$ steps; resampling improves both SOCK and rSig, with larger gains for rSig. Ablation 2 varies the input augmentation $\mathcal{A}$, defined in Appendix B.1; adding posneg improves RNN, but weakens some other metrics. Ablation 3 varies the kernel width W of the grouped random convolutions; both settings give good results, with $W = 2$ best on RNN. Ablation 4 varies the pooling applied to the soft kernel competitions, with poolings defined in Appendix B.3; all three poolings are competitive, with soft-dev best on RNN.

Limitations. These ablations are limited to the TGH dataset, which has a complex autocorrelation structure and heavy-tailed and skewed marginals, but its cross-correlation structure is rather simple. Results may therefore look different on other datasets.

Additional ablations. We additionally study choices of the SOCK architecture on purely discriminative tasks in section 5 and Appendix D.2. These also allow us to benchmark SOCK against other random convolutional feature maps that are non-differentiable and that therefore cannot be studied in the generative setting.

A.3 Additional visualizations

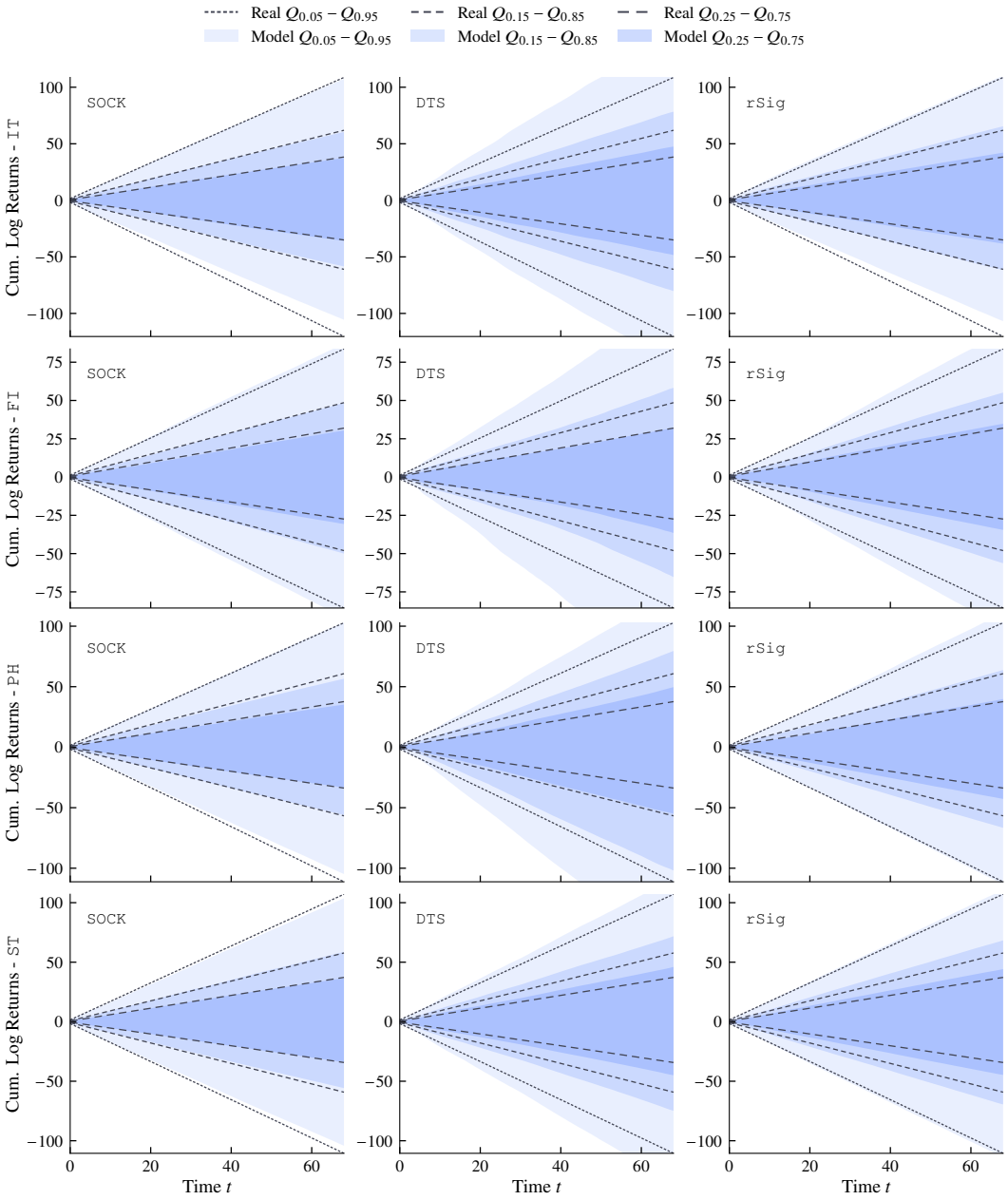

Figure 8: Real vs. model quantile bands over time for cumulative log returns on the first channel of each stock dataset. `SOCK` and `rSig` align more closely with the real quantiles than `DTS`.

B Additional details and pseudocode for SOCK

This appendix provides implementation details for the SOCK feature map from section 3.3. We discuss our implementation choices and define alternative choices used in our ablations.

B.1 Augmentations and normalization

In the first step of SOCK, we compute

$$Y_t = P\,\mathcal{N}(\mathcal{A}(X))_t,$$

where $\mathcal{A}$ is an augmentation map, $\mathcal{N}$ is a normalization function, and P is a random projection matrix. In this section, we provide details on the choice of $\mathcal{A}$ and $\mathcal{N}$.

Augmentations We make different augmentation choices between our generative and discriminative experiments. We provide an ablation in Appendix A.2 and use this section to define all the choices considered. When applying SOCK to a new task we recommend trying either $\mathcal{A}(X) = \texttt{int}(X)$ (when X is stationary) or $\mathcal{A}(X) = \texttt{diff}(X)$ (when X is non-stationary).

Let $U \in \mathbb{R}^{T \times d_U}$ be an input path. We use augmentation primitives that preserve U and append derived channels:

$$\begin{aligned}\texttt{diff}(U)_t &= [U_t,\ \Delta U_t],\\ \texttt{int}(U)_t &= [U_t,\ \textstyle\sum_{s=1}^{t} U_s],\\ \texttt{posneg}(U)_t &= [U_t,\ U_t^+,\ U_t^-].\end{aligned}$$

Here $\Delta U_1 = 0$, $\Delta U_t = U_t - U_{t-1}$ for $t > 1$, $U_t^+ = \max(U_t, 0)$, and $U_t^- = \min(U_t, 0)$, with maximum and minimum applied elementwise.

In the generative experiments, we use

$$\mathcal{A}(X) = \texttt{posneg}(\texttt{int}(X)).$$

We compare this choice to no augmentation, $\mathcal{A}(X) = X$, and to $\mathcal{A}(X) = \texttt{int}(X)$ in Appendix A.2.

Normalization For an augmented path $U = \mathcal{A}(X)$, the normalization $\mathcal{N}$ centers and scales each channel using statistics fitted on the augmented training paths:

$$\mathcal{N}(U)_t = (U_t - \mu) \odot \sigma^{-1},$$

where $\mu, \sigma \in \mathbb{R}^{d'}$ are computed over all training paths and time indices, and $\odot$ denotes elementwise multiplication.

B.2 Random parameters and dilations

The random parameters of SOCK consist of the projection matrix P and the convolutional kernels for each dilation. We draw a pre-projection matrix $\hat{P} \in \mathbb{R}^{M \times d'}$ with i.i.d. $N(0,1)$ entries and normalize each row:

$$P_{i,\cdot} = \frac{\hat{P}_{i,\cdot}}{\|\hat{P}_{i,\cdot}\|_2}.$$

We choose M divisible by the group width W and set $G = M/W$. For each dilation δ, group g, and kernel index k, we draw a pre-kernel $\hat{w}^{(g,k)} \in \mathbb{R}^{W \times L}$ with i.i.d. $N(0,1)$ entries, suppressing δ in the notation. We then center and normalize:

$$w^{(g,k)} = \frac{\hat{w}^{(g,k)} - \mathrm{mean}(\hat{w}^{(g,k)})}{\|\mathrm{vec}\big(\hat{w}^{(g,k)} - \mathrm{mean}(\hat{w}^{(g,k)})\big)\|_1}.$$

Thus each kernel has zero sum and unit ℓ_1 norm. This scheme is a straightforward extension of Hydra's [Dempster et al., 2023] kernel sampling to $W > 1$ and ensures that, a priori, each kernel has the same chance at winning the local competition.

We follow Hydra's dilation scheme [Dempster et al., 2023]. For input length n and kernel length $L = 9$,

$$\mathcal{D} = \{2^e :\ e = 0, 1, \ldots, e_{\max}\}, \qquad e_{\max} = \left\lfloor \log_2\left(\frac{n-1}{L-1}\right) \right\rfloor.$$

For segments of length $n = q + T = 69$, this gives $\mathcal{D} = \{1, 2, 4, 8\}$.

B.3 Pooling operators

For each group g and time index t, the K convolution responses define a competition between kernels. For $z \in \mathbb{R}^K$, let

$$\pi_\tau(z)_k = \frac{\exp(z_k/\tau)}{\sum_{j=1}^{K} \exp(z_j/\tau)}$$

denote the $\mathrm{softmax}$ at temperature $\tau > 0$.

Writing $Z_t^{(g,\cdot)} = (Z_t^{(g,1)}, \ldots, Z_t^{(g,K)})$, we define the following differentiable pooling operators:

$$F^{(g,k)}_{\text{soft-dev}} = \Big[\frac{1}{T}\sum_{t=1}^{T}\big(P_t^{(g,k)} - \frac{1}{T}\sum_{s=1}^{T} P_s^{(g,k)}\big)^2\Big]^{1/2}, \qquad P_t^{(g,k)} = \pi_\tau(Z_t^{(g,\cdot)})_k,$$

$$F^{(g,k)}_{\text{soft-count}} = \frac{1}{T}\sum_{t=1}^{T} P_t^{(g,k)}, \qquad P_t^{(g,k)} = \pi_\tau(Z_t^{(g,\cdot)})_k,$$

$$F^{(g,k)}_{\text{soft-value}} = \frac{1}{T}\sum_{t=1}^{T} \tilde{P}_t^{(g,k)} Z_t^{(g,k)}, \qquad \tilde{P}_t^{(g,k)} = \pi_\tau(-Z_t^{(g,\cdot)})_k.$$

The first statistic is the softmax-deviation pooling, which is `SOCK`'s default pooling. The other two are differentiable analogues of Hydra's count and value features; the value feature uses $\pi_\tau(-Z_t^{(g,\cdot)})$ to match Hydra's minimum-response convention.

B.4 Pseudocode

```python
import math
import torch
from torch import Tensor, nn

# the Aug modules map an input tensor x to an augmented version:
#      (B, n_steps, n_channels) -> (B, n_steps, n_channels + n_add_channels)
AUGMENTATIONS: dict[nn.Module] = {"cumsum": CumSumAug, "diff": DiffAug, "posneg": PosNegAug}

def normalize_kernel_(w: Tensor, eps: float = 1e-6) -> None:
    w.sub_(w.mean(dim=(-2, -1), keepdim=True)) # dim -2 is kernel width and dim -1 is kernel length
    w.div_(w.abs().sum(dim=(-2, -1), keepdim=True).clamp_min(eps))

def normalize_proj_(w: torch.Tensor, eps: float = 1e-6) -> None:
    w.div_(w.norm(p=2, dim=1, keepdim=True).clamp_min(eps))

class SOCK(nn.Module):
    def __init__(self,
        n_steps: int,  # corresponds to T in the paper
        n_channels: int,  # corresponds to d in the paper
        tau: float = 0.1,  # softmax temperature
        k: int = 8,  # number of kernels per group
        mix_dim: int = 256,  # corresponds to M in the paper
        kernel_len: int = 9,
        augs: tuple[str, ...] = ("cumsum",),
    ) -> None:
        super().__init__()
        self.tau, self.k = tau, k

        self.augs = nn.Sequential(*[AUGMENTATIONS[aug]() for aug in augs])
        n_channels += sum(aug.n_add_channels for aug in self.augs)
        self.proj = nn.Linear(n_channels, mix_dim, bias=False)

        emax = math.log2((n_steps - 1) / (kernel_len - 1))
        self.dilations = (2 ** torch.arange(int(emax) + 1)).int()
        kernel_width = 2
        self.convs = nn.ModuleList()
        for d in self.dilations:
            self.convs.append(
                nn.Conv1d(
                    in_channels=mix_dim,
                    out_channels=k * (mix_dim // kernel_width),
                    kernel_size=kernel_len,
                    padding="same",
                    dilation=d,
                    groups=mix_dim // kernel_width,
                    bias=False,
                )
            )

        for p in self.parameters():  # SOCK's parameters are untrained, only need gradient w.r.t input x
            p.requires_grad = False
        self.resample()

    def resample(self) -> None:
        self.proj.weight.normal_()
        normalize_proj_(self.proj.weight)
        for conv in self.convs:
            conv.weight.normal_()
            normalize_kernel_(conv.weight)

    def fit_input_scales(self, x: Tensor) -> None: ...  # fits (input_mean, input_scl)
    def fit_ft_scales(self, x: Tensor) -> None: ...  # fits (ft_mean, ft_scl); call after every resample

    def pool(self, z: Tensor) -> Tensor:  # soft-deviation pooling
        return torch.std(torch.softmax(z / self.tau, dim=2), dim=-1)

    def forward(self, x: Tensor) -> Tensor:  # x.shape = (B, n_steps, n_channels)
        x = self.augs(x)
        x = (x - self.input_mean) / self.input_scl
        x = self.proj(x).permute(0, 2, 1)  # (B, mix_dim, n_steps)

        feats = []
        for conv in self.convs:  # loop over dilations; for T=64, use dilations 1,2,4
            z = conv(x)  # (B, n_groups * k, n_steps), where n_groups = mix_dim // kernel_width
            z = z.view(x.size(0), -1, self.k, x.size(-1))  # (B, n_groups, k, n_steps)
            f = self.pool(z)  # (B, n_groups, k)
            feats.append(f.view(x.size(0), -1))  # (B, n_groups * k)
        feats = torch.cat(feats, dim=-1)  # (B, n_dilations * n_groups * k)
        return (feats - self.ft_mean) / self.ft_scl
```

C Additional details on time series generation experiments

C.1 Datasets

Vector Autoregressive Process - `V1`. The process `V1` is a 3-dimensional vector autoregressive process

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

where $\Sigma = \frac{1}{2}J + \frac{1}{2}I$ [Liao et al., 2024] and J is a matrix where each entry is equal to 1. We set $\phi = 0.99$, meaning the process exhibits strong autocorrelation, which should be simple to detect and generate. We include it as a first benchmark, following prior work [Liao et al., 2024, Biagini et al., 2024].

Vector Autoregressive Process - `V10`. The process `V10` is a 3-dimensional vector autoregressive process with oscillating autocorrelation presented in Brockwell and Davis [2016], where

$$X_t = \phi_1 X_{t-5} + \phi_2 X_{t-10} + \varepsilon_t, \quad \varepsilon_t \sim N(0, \Sigma), \tag{3}$$

with coefficients $\phi_1 = 2r\cos(\omega\pi)$ and $\phi_2 = -r^2$. We set $r = \omega = 0.8$ and use innovation covariance $\Sigma = \frac{1}{15}(\frac{1}{2}J + \frac{1}{2}I)$, which has trace 0.2. With this process, we can test the generative model's ability to reproduce complex autocorrelation patterns.

Tukey g and h - `TGH`. The process `TGH` is an instance of the class of processes studied by Yan and Genton [2019]. The process is generated by first simulating the `V10` process from eq. (3) and then applying pointwise the Tukey g and h transform [Tukey, 1977], which is defined as

$$\tau_{g,h}(x) = \begin{cases} \dfrac{e^{gx}-1}{g}\exp\left(\dfrac{hx^2}{2}\right), & g \neq 0, \\ x\exp\left(\dfrac{hx^2}{2}\right), & g = 0. \end{cases}$$

We set $g = -0.2$ and $h = 0.2$ to induce a negative skew and heavy tails on the marginals of the process. We apply the transform at the Gaussian process's simulation scale, then center and scale each transformed channel using statistics fitted on the training path.

Stochastic volatility - `SV`. The process `SV` is a 2-dimensional process based on the stochastic volatility model by Heston [1993]:

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

We set $\kappa = 1.0$, $\theta = 0.04$, $\xi = 0.25$, $\rho = -0.7$, and simulate (S, V) paths using the scheme of Broadie and Kaya [2006] at a time discretization of $\Delta t = 1/250$. We then transform (S, V) into

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

which is a stationary Markov process. We include this process in our study due to its practical relevancy in finance and because accurately generating the process requires correctly reproducing the spot-vol correlation ρ, as well as the skewed marginals of $\log V$.

Fractional Gaussian Noise - `FGN`. The process `FGN` is a 3-dimensional Gaussian process with independent channels. For each $k \in \{1, 2, 3\}$, let $(B_t^{\mathcal{H}_k})_{t\in\mathbb{R}}$ be a fractional Brownian motion with Hurst parameter $\mathcal{H}_k$ [Mandelbrot and Van Ness, 1968]. We generate sequences of length $T = 64$ by discretizing the time interval $[0, 1]$ using the uniform grid $(t_i = i/T)$, and defining the k-th channel as the increment process

$$X_i^{(k)} := B_{t_i}^{\mathcal{H}_k} - B_{t_{i-1}}^{\mathcal{H}_k}, \qquad i = 1, \ldots, T.$$

We set $X_i := (X_i^{(1)}, X_i^{(2)}, X_i^{(3)}) \in \mathbb{R}^3$. We use distinct Hurst parameters $\mathcal{H}_1 = 0.05$, $\mathcal{H}_2 = 0.15$, and $\mathcal{H}_3 = 0.25$, so each channel exhibits antipersistent, slowly decaying correlations, making the temporal dependence non-Markovian and challenging to reproduce.

Real datasets (daily). We construct six daily datasets from Yahoo Finance.[8] We download daily close prices over 2009–2025, computing log-returns $r_t = \log(P_t/P_{t-1})$, and splitting the resulting

Table 3: Daily real datasets

Dataset	d	Tickers
FI	4	JPM, DB, UBS, HSBC
PH	4	PFE, JNJ, AZN, NVS
IT	4	MSFT, GOOGL, NVDA, AAPL
ST	4	PG, NSRGY, KO, UL
IDX	3	^GSPC, ^VIX, GC=F
FX	3	EURUSD=X, JPYUSD=X, GBPUSD=X

series into a training period (2009–2017) and a validation period (2017–2025). The datasets and their tickers are listed in table 3.

Cryptocurrency dataset (CRY, intraday). The dataset CRY is constructed from Binance public market data.[9] We use BTC/USD and ETH/USD close prices, extract the period 2025-01-01 to 2025-03-01, resample to 5-minute frequency by last price, compute log-returns, and split into a training period (2025-01-01 to 2025-02-01) and a validation period (2025-02-01 to 2025-03-01).

Higher-dimensional datasets. Note that all our experiments focus on time series with a moderate number of channels. This is a deliberate choice since learning a flexible conditional simulator from a single historical path is already statistically challenging, and increasing the channel dimension without more data makes the problem more underdetermined. High-dimensional return generation is commonly studied under different simplifying assumptions: Chen et al. [2025] impose a low-dimensional factor structure, while Gao et al. [2025] generate next-day returns conditional on asset-specific factors. These settings differ from generating future paths conditional on a recent observed path. Motivated by these works, our generative modelling approach can be extended to higher-dimensional financial data by estimating a low-dimensional factor representation, for example $X_t \approx BF_t + \varepsilon_t$ with $F_t \in \mathbb{R}^k$ and $k \ll d$, and applying random convolutional feature matching to the factor process. We leave this extension to future work.

C.2 Evaluation

C.2.1 Evaluation protocol

We evaluate generators by comparing empirical distributions of joined past–future segments, as defined in section 3.1. For each evaluation time t, we compare the realized segment $x_t^- \oplus x_t^+$ to the generated segment $x_t^- \oplus \hat{x}_t^+$, where both segments share the same length-q context x_t^- and differ only in their length-T continuation.

On real datasets, we use evaluation times $\mathcal{T}_{\text{eval}} = \{H, H+1, \dots, H+N-T\}$ along the single out-of-sample trajectory. Using overlapping segments is necessary to obtain enough evaluation samples, but reduces the effective sample size. The resulting empirical collections are

$$\{x_t^- \oplus x_t^+\}_{t \in \mathcal{T}_{\text{eval}}} \qquad \text{and} \qquad \{x_t^- \oplus \hat{x}_t^+\}_{t \in \mathcal{T}_{\text{eval}}}.$$

On synthetic datasets, we sample J independent continuations and use the non-overlapping grid $\mathcal{T}_{\text{eval}}^{\text{syn}} = \{H,\ H+T,\ H+2T,\ \dots,\ H+N-T\}$. We then pool segments over continuation index $j = 1, \dots, J$ and evaluation times $t \in \mathcal{T}_{\text{eval}}^{\text{syn}}$, flattening (j, t) into a single empirical collection.

We assess the similarity between these empirical path distributions using a large suite of metrics defined below. Many of these metrics have been proposed and used in prior work [Yoon et al., 2019, Ni et al., 2022, Wiese et al., 2021, Liao et al., 2024, Cont et al., 2025].

Motivation of our training & evaluation protocol Our single-path training and out-of-sample evaluation protocol is designed to mimic how generative time series models are trained and deployed in financial applications, where generative time series models are used as *market simulators* [Wiese et al., 2019, Buehler et al., 2020, Cohen et al., 2023, Lütkebohmert et al., 2022, Wiese and Murray,

[8] https://finance.yahoo.com/. Yahoo Finance data are subject to Yahoo's terms of use and are not redistributed.

[9] https://data.binance.vision/. The accompanying Binance public-data repository is MIT licensed: https://github.com/binance/binance-public-data.

Table 4: Evaluation metrics. Discriminative metrics report $|\text{accuracy} - 0.5|$ for classifiers trained to distinguish real from generated paths. All metrics are defined so that lower values indicate better fit.

Metric	Description
Discriminative metrics	
SIG	Linear classifier on truncated signature features.
MLP	Linear classifier on random one-hidden-layer MLP features of flattened paths.
RNN	GRU classifier applied to individual paths, following Yoon et al. [2019].
SRNN	Permutation-invariant GRU classifier applied to sets of 8 paths.
Distributional metrics	
ACF	Average absolute discrepancy between channel-wise autocorrelation functions.
CCF	Average absolute discrepancy between Pearson cross-correlation matrices.
CVM	Average Cramér–von Mises distance between one-dimensional channel marginals.
ES	Average relative discrepancy in per-channel expected shortfall at level 5%.
ED	Energy distance between empirical distributions of consecutive observation pairs $(X_t, X_{t+1}) \in \mathbb{R}^{2d}$.
PES	Average relative discrepancy in expected shortfall of trading-strategy PnL distributions, following Cont et al. [2025].

2022, Limmer and Horvath, 2024, Cont and Vuletić, 2025, Jones et al., 2025, He et al., 2025]. These market simulators are often used to optimize and stress-test decision rules (e.g., via neural network policies) for general portfolio management and hedging problems [Buehler et al., 2019, Mariani et al., 2019, Murray et al., 2022, Krabichler and Teichmann, 2023, Englisch et al., 2023, Mueller et al., 2024]. Across all these applications, the core generative problem is the same: given a single observed historical path, generate future trajectories conditional on the recent past (section 3.1). To obtain sufficient out-of-sample samples for the distributional evaluation, we train the generator once and keep it fixed throughout evaluation, rolling it forward across evaluation times. Such simulator reuse can also be useful in practice, since it amortizes generator training cost and because incremental new data often has limited effects on the optimal generator. In practice, the generator may nonetheless be retrained once enough new data is available or after detecting a distributional shift. For simplicity and to control compute, we do not perform such generator refitting.

Broader impacts Improved financial time series generation from small datasets can support risk analysis and stress testing for many financial applications when historical data is limited. The same capability also creates a risk of overreliance if uncertainty about the generative model is ignored.

C.2.2 Discriminative evaluation metrics

We report four discriminative metrics: SIG, MLP, RNN, and SRNN. For each metric, we train a binary classifier to distinguish real paths from fake paths. We train on half of the validation set, evaluate accuracy on the other half, and report the discriminative score $|\text{accuracy} - 0.5|$.

For the metrics SIG and MLP, we fit linear ridge classifiers on top of fixed feature maps. SIG uses truncated signature features (degree 3), computed on augmented versions of the paths (same augmentations as described in section C.4). MLP uses random features obtained by passing the flattened path $\text{vec}(X) \in \mathbb{R}^{Td}$ through a randomly initialized one-hidden-layer MLP (hidden dimension 512, ReLU). To reduce evaluation variance due to the MLP initialization, we repeat sampling the MLP and fitting the linear classifier for 5 times and report the averaged discriminative score.

Following Yoon et al. [2019], we further report RNN, which is the discriminative score obtained by a fully trained GRU-based classifier [Chung et al., 2014]. Note that these three discriminative scores (SIG, MLP, RNN) classify individual paths as real or generated. To better capture distributional properties, we also report SRNN, which classifies *sets* of paths rather than individual paths. Concretely, given a set of $B_S = 8$ paths $\{X^i\}_{i=1}^{B_S}$, we compute a GRU embedding for each path and then apply a Deep Sets aggregator [Zaheer et al., 2017]:

$$\ell = \rho\Big(\frac{1}{B_S}\sum\nolimits_{i=1}^{B_S} \phi(\text{GRU}(X^i))\Big), \qquad p(\text{real} \mid \{X^i\}_{i=1}^{B_S}) = \sigma(\ell).$$

Here ϕ and ρ are MLPs, σ is the sigmoid function, and the mean-pooling makes the score permutation-invariant over the set. To reduce training noise, we train two GRU-based classifiers (RNN and SRNN) ten times and average the resulting scores.

C.2.3 Distributional evaluation metrics

Below, superscripts r and g denote quantities computed on the real and generated segment collections, respectively. We report three metrics (CVM, CCF, ES) that ignore temporal ordering and for which we aggregate all observations across segments and time. The other three metrics (ACF, ED, PES) capture temporal dependencies.

Cramér–von Mises distance (CVM). We measure similarity of the one-dimensional marginals using the average Cramér–von Mises distance:

$$\mathrm{CVM} = \frac{1}{d}\sum_{j=1}^{d}\int_{-\infty}^{\infty}\left(\hat{F}^{(r,j)}(u) - \hat{F}^{(g,j)}(u)\right)^2 \mathrm{d}\hat{H}^{(j)}(u),$$

where $\hat{F}^{(r,j)}$ and $\hat{F}^{(g,j)}$ are the ECDFs of channel j computed from the aggregated real and generated observations, and $\hat{H}^{(j)}$ is the ECDF of the pooled real+generated sample in channel j.

Cross-correlation difference (CCF). We measure cross-sectional dependence via the average absolute difference between Pearson cross-correlation matrices:

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

where $\chi^{(r)}, \chi^{(g)} \in \mathbb{R}^{d\times d}$ are the Pearson correlation matrices computed from the aggregated real and generated observations, respectively.

Relative expected shortfall difference (ES). We compare tail risk using the average relative absolute error in Expected Shortfall (ES) at level $\alpha = 0.05$:

$$\mathrm{ES} = \frac{1}{d}\sum_{j=1}^{d}\frac{\left|\mathrm{ES}_{\alpha,j}^{(r)} - \mathrm{ES}_{\alpha,j}^{(g)}\right|}{\left|\mathrm{ES}_{\alpha,j}^{(r)}\right|}.$$

Here $\mathrm{ES}_{\alpha,j}^{(\cdot)}$ denotes the empirical expected shortfall of channel j at tail level α, computed as the mean of observations below the empirical α-quantile (after aggregating over segments and time).

Autocorrelation difference (ACF). We measure temporal dependence via the average absolute difference between channel-wise autocorrelation functions:

$$\mathrm{ACF} = \frac{1}{d\,\ell_{\max}}\sum_{j=1}^{d}\sum_{k=1}^{\ell_{\max}}\left|\rho_j^{(r)}(k) - \rho_j^{(g)}(k)\right|, \qquad \ell_{\max} = \lfloor T/3 \rfloor,$$

where $\rho_j^{(r)}(k)$ and $\rho_j^{(g)}(k)$ denote the empirical autocorrelation at lag k for channel j, computed from the real and generated segment collections, respectively.

Energy distance (ED). We compute the Energy Distance between the empirical distributions of consecutive time-step pairs. Let $(x, y) \in \mathbb{R}^{2d}$ denote two *consecutive* observations from the same segment, i.e., y is the next time step after x:

$$Z := (x, y) \in \mathbb{R}^{2d}, \qquad \mathrm{ED} = \sqrt{2\,\mathbb{E}\|Z^{(r)} - Z^{(g)}\|_2 - \mathbb{E}\|Z^{(r)} - Z'^{(r)}\|_2 - \mathbb{E}\|Z^{(g)} - Z'^{(g)}\|_2},$$

where the expectations are taken with respect to the empirical distributions of real and generated consecutive pairs (aggregated over segments and valid time steps).

PnL expected shortfall difference (PES). Following Cont et al. [2025], we assess tail fit by comparing the tails of profit-and-loss (PnL) distributions induced by simple trading strategies applied to real versus generated segments. For each strategy $j \in \{1, \ldots, N_{\text{strategies}}\}$, we compute a PnL time series on the real and generated data, denoted $\mathrm{PnL}_j^{(r)}$ and $\mathrm{PnL}_j^{(g)}$, and compare their expected shortfall at tail level $\alpha = 0.05$. We define

$$\mathrm{PES} = \frac{1}{N_{\text{strategies}}}\sum_{j=1}^{N_{\text{strategies}}}\frac{\left|\mathrm{ES}_\alpha\left(\mathrm{PnL}_j^{(r)}\right) - \mathrm{ES}_\alpha\left(\mathrm{PnL}_j^{(g)}\right)\right|}{\left|\mathrm{ES}_\alpha\left(\mathrm{PnL}_j^{(r)}\right)\right|}.$$

Table 5: Feature-scaling ablation for `Sig` and `rSig` on the `TGH` dataset; metrics are averaged over five random seeds.

METHOD	SCALING	RNN	ACF	CCF	CVM	ED	ES
`Sig`	×	**0.319**	**0.079**	0.032	**0.0008**	**0.079**	**0.109**
`Sig`	✓	0.368	0.084	**0.025**	0.0010	0.101	0.110
`rSig`	×	0.296	0.089	0.062	**0.0003**	**0.068**	**0.044**
`rSig`	✓	**0.288**	**0.085**	**0.049**	0.0005	0.084	0.047

We use the following strategies:

- *Buy-and-hold (single channel):* d strategies, each taking a unit position $+1$ in a single channel and 0 in all others.
- *Buy-and-hold* $(1/d)$*:* one strategy taking an equal-weight position $1/d$ in each channel.
- *Momentum:* d strategies, where the position in channel j at time t is $\operatorname{sign}(\bar{X}_t^{(j)})$, with $\bar{X}_t^{(j)}$ the 10-step moving average of channel j up to time t.

C.3 Additional training details

C.3.1 Data and feature scaling

Data scaling. On each dataset, we compute the per-channel means and standard deviations on the raw training path and use those vectors to standardize the training and validation data.

Feature scaling. We optimize the objective in eq. (2) using minibatches of size B for the inner expectations and one draw of the feature-map parameters ψ at a time. At each step, we sample indices $t_1, \dots, t_B$ uniformly from the valid training indices and generate $\hat{x}^+_{t_i} \sim p_\theta(\cdot \mid x^-_{t_i})$. We then compute

$$F_i = f_\psi(x^-_{t_i} \oplus x^+_{t_i}), \qquad \hat{F}_i = f_\psi(x^-_{t_i} \oplus \hat{x}^+_{t_i}).$$

The corresponding minibatch version of eq. (2) is therefore

$$\mathcal{L}_B(\theta; \psi) = \left\| \frac{1}{B} \sum_{i=1}^{B} F_i - \frac{1}{B} \sum_{i=1}^{B} \hat{F}_i \right\|_2^2.$$

For all feature maps (`SOCK`, `rSig`, and `Sig`), we also considered the scaled loss

$$\mathcal{L}_B^{\text{scaled}}(\theta; \psi) = \left\| \boldsymbol{\sigma}_\psi^{-1} \odot \left(\frac{1}{B} \sum_{i=1}^{B} F_i - \frac{1}{B} \sum_{i=1}^{B} \hat{F}_i \right) \right\|_2^2,$$

where $\boldsymbol{\sigma}_{\psi,j}$ is the empirical standard deviation of feature coordinate j over all real training segments for the current feature-map draw, and $\odot$ denotes componentwise multiplication. This scaling is related to feature matching methods that use higher moments [Mroueh et al., 2017]. We found in preliminary experiments that this scaling improved generator training with SOCK features. However, this scaling does not generally improve generator training with `Sig` or `rSig` features (table 5). In all experiments, we therefore apply this scaling to `SOCK` features, but do not apply it to `Sig` or `rSig` features, which is consistent with prior works Liao et al. [2024], Biagini et al. [2024].

C.3.2 Generator architecture

We use a simple conditional generator based on a single-layer GRU that decodes an i.i.d. noise sequence into an output path. We choose this architecture for simplicity and fast sampling. Improving the generator architecture is complementary to our approach and may yield further gains.

```python
class Generator(nn.Module):
    def __init__(self, d: int, hidden_dim: int = 128, q: int = 5) -> None:
        super().__init__()
        self.noise_dim = self.initial_noise_dim = d  # set noise dimension to data dimension
```

```python
        # maps (flattened context c, initial noise) -> initial hidden state h0
        self.initial_state_generator = nn.Sequential(
            nn.Linear(d * q + self.initial_noise_dim, hidden_dim),
            nn.SiLU(),
            nn.Linear(hidden_dim, hidden_dim),
        )

        self.proj_in = nn.Linear(self.noise_dim, 2 * hidden_dim)
        self.rnn = nn.GRU(hidden_dim, hidden_dim, num_layers=1, batch_first=True)

        # residual noise injection: add a gated noise stream after the GRU
        self.alpha = nn.Parameter(torch.tensor(0.1))
        self.gate = nn.Sequential(
            nn.LayerNorm(hidden_dim), nn.Linear(hidden_dim, hidden_dim), nn.Sigmoid()
        )

        self.proj_out = nn.Linear(hidden_dim, d)

    def reset_parameters(self) -> None: ...  # stabilizing initialization (e.g., small output/MLP scales)

    def forward(self, c: Tensor, n_steps: int = 64) -> Tensor:  # c.shape = (B, q, d)
        # sample initial hidden state h0 conditionally on context c
        initial_noise = torch.randn((c.size(0), self.initial_noise_dim), device=c.device)
        h0_in = torch.cat((c.flatten(start_dim=1), initial_noise), dim=-1)
        h0 = self.initial_state_generator(h0_in)

        # decode per-step noise with: linear proj. -> SiLU -> GRU
        z = torch.randn((c.size(0), n_steps, self.noise_dim), device=c.device)
        z, z_skip = self.proj_in(z).chunk(2, dim=-1)
        h, _ = self.rnn(F.silu(z), h0.unsqueeze(0))

        h = h + self.alpha * self.gate(h) * z_skip  # optional output noise injection
        return self.proj_out(h)
```

Do generators learn to condition on the past? To evaluate this architecture, we test whether generators trained with SOCK features use their conditioning context by comparing correct and randomly shuffled contexts. Let $\{x_t^-\}$ be the evaluation contexts and let π be a random permutation of their indices. Normally, we sample $\hat{x}_t^+ \sim p_\theta(\cdot \mid x_t^-)$ and evaluate the joined path $x_t^- \oplus \hat{x}_t^+$. For the shuffled evaluation, we instead sample $\widetilde{x}_t^+ \sim p_\theta(\cdot \mid x_{\pi(t)}^-)$ and evaluate $x_t^- \oplus \widetilde{x}_t^+$. This gives two collections of paths: $\mathcal{X}_{\text{correct}} = \{x_t^- \oplus \hat{x}_t^+\}_t$, which pairs each context with a continuation generated from that context, and $\mathcal{X}_{\text{shuffled}} = \{x_t^- \oplus \widetilde{x}_t^+\}_t$, which pairs each context with a continuation generated from the permuted context $x_{\pi(t)}^-$. We compute the ACF metric on each collection as it is sensitive to whether the generated continuation is temporally consistent with the real prefix. We perform this evaluation for SOCK generators trained on the TGH and SV datasets (five seeds each). For each generator, we average results over five draws of the permutation π. Supplying the wrong contexts to the generators results in an increase of the ACF metric by 18.2% on TGH and 99.1% on SV (table 6). This shows that the trained generators have learned to meaningfully use the conditioning context to produce future continuations.

Table 6: Context-shuffling ablation for generators trained with SOCK features. Results are averaged over five training seeds and five context-permutation and noise realizations per seed.

DATASET	ACF ON $\mathcal{X}_{\text{correct}}$	ACF ON $\mathcal{X}_{\text{shuffled}}$	RELATIVE INCREASE
TGH	**0.024578**	0.029062	+18.2%
SV	**0.023370**	0.046539	+99.1%

C.3.3 Optimization

For all the feature matching methods (Sig, rSig, cSig, SOCK), we use the same generator architecture and training protocol across all datasets. We tuned these choices on the synthetic datasets to avoid overfitting to the small real datasets. We train the generator for 100,000 optimization steps using the AdamW optimizer (weight decay of 0.01) and a linear learning rate schedule (linear warm up for first 5% of steps to 3×10^{-4} and linear decay for last 70% of steps).

Compute. All reported generative experiments fit on a single NVIDIA A40 GPU with 48GB memory. For one dataset and seed, training a GRU generator with feature matching for 100k steps

takes about 1 hour; sampling 65k paths takes less than 1 minute. Diffusion-TS trains faster, about 10 minutes, but sampling 65k paths takes more than 30 minutes. `SOCK` feature extraction is not the training bottleneck; the dominant cost is the backward pass through the GRU generator.

C.4 Additional details for the baselines

C.4.1 Truncated signature (`Sig`)

As discussed in section 4.1, we apply three path transforms prior to computing signature terms for all signature-based methods: Lead-Lag, Time, and I-visibility. To compute the truncated signature, we use Signatory Kidger and Lyons [2021]. We truncate the signature of the augmented path at degree 3, following Liao et al. [2024].

C.4.2 Randomized signature (`rSig`)

Due to the exponential $\mathcal{O}(d^m)$ time and space complexity of computing m-level truncated signatures for d-dimensional paths or time series, the randomized signature Cuchiero et al. [2021], Cirone et al. [2023] is often preferred, defined below.

Definition C.1. Let $M \geq 1$ be an integer. Fix an initial condition $z_0 \in \mathbb{R}^M$, random matrices $A_1, \cdots, A_d \in \mathbb{R}^{M \times M}$, random biases $b_1, \cdots, b_d \in \mathbb{R}^M$ and an activation function σ. The randomized signature Z of a path $t \mapsto x_t$ is defined as the solution of the controlled differential equation (CDE)

$$dZ_t = \sum_{i=1}^{d} \sigma(A_i Z_t + b_i) dx_t^{(i)}, \qquad Z_0 = z_0, \tag{4}$$

where $x^{(i)}$ denotes the i'th component of x.

The randomized signature was first constructed by Cuchiero et al. [2021] as a random projection of the signature, with an argument based on a non-trivial application of the Johnson-Lindenstrauss lemma. Randomized signatures have previously been used for various time series applications, for instance anomaly detection [Akyildirim et al., 2022, Zozoulenko et al., 2025a], graph conversion [Schäfl et al., 2023], optimal portfolio selection [Akyildirim et al., 2025, Cuchiero and Möller, 2025], generative modeling [Biagini et al., 2024], and for learning rough dynamical systems [Compagnoni et al., 2023]. The CDE (4) has since been studied from the perspective of randomly initialized ResNets [Cirone et al., 2023, 2024], and path developments on compact Lie groups [Lou et al., 2023, 2024, Cass and Turner, 2026]. We use the version of Cirone et al. [2023] with vector fields $A_i\sigma(Z_t) + b_i$ and $\sigma = \tanh$, implemented below. In preliminary experiments, we observed similar performance to an Euler discretization of (4). We expect differences in generator performance between the two versions to be small, particularly when the random parameters are resampled during generator training.

```python
def randomized_sig_tanh(X: Tensor, A: Tensor, b: Tensor, Y_0: Tensor) -> Tensor:
    """
    Randomized signature of a (batched) time series X, with tanh activation function.

    Args:
        X (Tensor): Input tensor of shape (N, T, D).
        A (Tensor): Tensor of shape (M, M, D). Random matrix.
        b (Tensor): Tensor of shape (M, D). Random bias.
        Y_0 (Tensor): Initial hidden state.
            Tensor of shape (M).
    """
    N, T, D = X.shape
    diff = X.diff(dim=1) # shape (N, T-1, D)
    Y_0 = torch.tile(Y_0, (N, 1)) # shape (N, M)
    Z = torch.tensordot(tanh(Y_0), A, dims=1) + b[None] # shape (N, M, D)
    Y = Y_0 + (Z * diff[:, 0:1, :]).sum(dim=-1) # shape (N, M)
    for t in range(1, T-1):
        Z = torch.tensordot(tanh(Y), A, dims=1) + b[None]
        Y = Y + (Z * diff[:, t:t+1, :]).sum(dim=-1)
    return Y
```

C.4.3 Conditional Sig-Wasserstein GAN (cSig)

Liao et al. [2024] train conditional generators by matching signature features of future segments. For each training time t, we distinguish three segments:

$$x_t^- := X_{t-q+1:t}, \qquad r_t^- := X_{t-T_-+1:t}, \qquad x_t^+ := X_{t+1:t+T_+}.$$

The generator receives the q observations in x_t^- and samples $\hat{x}_t^+ \sim p_\theta(\cdot \mid x_t^-)$. The longer segment r_t^- is used only to construct the target feature mean for the real future segment x_t^+.

Let f denote the truncated signature of an augmented input path. If the target feature mean of the real future were available, the cSig loss would compare it to the generator's conditional feature mean:

$$\mathbb{E}_t\Big[\big\|\mathbb{E}_{x_t^+\sim p(\cdot|r_t^-)}\big[f(x_t^+)\big] - \mathbb{E}_{\hat{x}_t^+\sim p_\theta(\cdot|x_t^-)}\big[f(\hat{x}_t^+)\big]\big\|_2^2\Big]. \tag{5}$$

The second expectation can be estimated by Monte Carlo, because we can sample many generated futures from $p_\theta(\cdot \mid x_t^-)$. The first expectation is not directly observable from a single historical path: for each r_t^- we observe only one realized future x_t^+.

Liao et al. [2024] therefore estimate the first expectation before training the generator. They fit a linear map from signature features of the longer past segment to signature features of the future segment,

$$\mathbb{E}_{x_t^+\sim p(\cdot|r_t^-)}\big[f(x_t^+)\big] \approx b + W f(r_t^-).$$

The coefficients (b, W) are fitted on the training path and then held fixed. The generator is then trained with the practical objective

$$\mathbb{E}_t\Big[\big\|b + W f(r_t^-) - \mathbb{E}_{\hat{x}_t^+\sim p_\theta(\cdot|x_t^-)}\big[f(\hat{x}_t^+)\big]\big\|_2^2\Big].$$

Thus the target in the cSig loss is predicted from r_t^-, while the generator itself is still conditioned only on x_t^-. This objective differs from ours in eq. (2): cSig matches conditional feature means of the future segment x_t^+, whereas our objective matches features of the joined segment $x_t^- \oplus x_t^+$.

Liao et al. [2024] choose leading and generated segments of equal length ($T_- = T_+$) in their experiments. We make the same choice for our cSig generators: $T_- = T_+ = T = 64$. Thus the regression map is fitted from T past observations, while the generator is still conditioned on the $q = 5$ observations in x_t^-.

All runs with cSig therefore use $T_- = T_+ = 64$. We implement the nested sampling scheme at batch size $B = 1000$: for each of the $B^{1/3} = 10$ outer samples we use $B^{2/3} = 100$ inner samples to approximate the inner expectations [Gordy and Juneja, 2010].

C.4.4 Diffusion-TS (DTS)

We train a separate Diffusion-TS model for each dataset and each seed. We adapt the authors' implementation [Yuan and Qiao, 2024] to run on our datasets.[10] Note that by default, Diffusion-TS uses the same stride-1 sampling scheme for training to extract smaller path segments from a single historical time series as we do. We followed Yuan and Qiao [2024]'s preprocessing choices for our datasets, which require scaling the training data to the unit interval $[0, 1]$. During evaluation, Diffusion-TS again produces only samples in this range, which may limit performance of Diffusion-TS on our out-of-sample evaluation.

To assess this support restriction, we compare the default sampling procedure with sampling that disables clipping of the predicted clean sample to the model range $[-1, 1]$ (by setting `clip_denoised=False`). We run this ablation on TGH using the same trained models for both sampling procedures.

Table 7: Effect of sample clipping for Diffusion-TS on the TGH dataset; metrics are averaged over two seeds.

METHOD	RNN	ACF	CCF	CVM	ED	ES	PES
DTS (CLIPPING)	**0.382**	0.051	0.077	**0.0026**	**0.256**	**0.304**	**0.162**
DTS (NO CLIPPING)	0.454	**0.037**	**0.060**	0.0072	0.386	0.409	1.055

[10] See https://github.com/Y-debug-sys/Diffusion-TS.

Disabling clipping improves ACF and CCF but worsens the other reported metrics (table 7). In particular, both tail metrics, ES and PES, deteriorate when clipping is disabled. A possible explanation for this is that the generator produces extreme outliers when clipping is disabled. Clipping then suppresses these samples, which improves the tail metrics despite biasing the generated tails.

D Additional details and ablations on discriminative experiments

D.1 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) [Gretton et al., 2012] based on either explicit feature vectors (as in eq. (1)) or time series kernels, within a permutation test as in Chevyrev and Oberhauser [2022], Salvi et al. [2021], Lou et al. [2023], Li et al. [2024], Tao et al. [2024]. 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.

Test Procedure Let $\mathcal{X} = (X^{(1)}, \dots, X^{(N)}) \sim \mu$ and $\mathcal{Y} = (Y^{(1)}, \dots, Y^{(M)}) \sim \nu$ be independent samples of sizes N and M, respectively, with i.i.d. elements. The permutation 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 three different stochastic processes that capture different time series characteristics: 1.) `FBM`: We vary the Hurst exponent $H \in (0, 1)$ under H_1, and set $H = 0.5$ under H_0. 2.) `TGH`: We vary the tail heaviness parameter h under H_1, and set $h = 0$ under H_0. 3.) `SV`: We vary the correlation $\rho \in (-1, 1)$ under H_1, and set $\rho = 0$ under H_0.

Hyperparameter Tuning For each stochastic process, we independently tune the hyperparameters of each MMD model to maximize the test power, using a hold-out data set generated using a separate random seed. Hydra uses its default hyperparameters and requires no tuning as per the original paper Dempster et al. [2023]. 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` and Hydra we set the number of competing kernels to $k = 3$, and for `SOCK` we only tune the softmax 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 signature-based models we preprocess all paths with the standard path augmentations from the signature literature, namely Lead-lag, Add-time, and I-visibility transformations McLeod and Lyons [2025], Wu et al. [2021]. We note however that this has little effect on `SOCK`, but is required to make signatures competitive. The experiment results are displayed in fig. 3.

D.2 Time series classification

In this section we provide additional experimental details and ablation studies for the time series classification evaluation on the classical 112 UCR datasets [Middlehurst et al., 2024]. The UCR Time Series Classification Archive is distributed under CC-BY 4.0.[11]

Experimental Setup For all classification experiments, we follow the standard evaluation protocol established by the Rocket and Hydra literature [Dempster et al., 2020, 2021, Tan et al., 2022, Dempster et al., 2023]. We generate random convolutional features using either `SOCK`'s softmax-deviation pooling or the pooling mechanisms from Rocket/Hydra, and then fit a RidgeCV classifier (see e.g. `scikit-learn` [Pedregosa et al., 2011]). We use the standard training and test splits provided by the UCR archive [Middlehurst et al., 2024], reporting results averaged across 30 seeded resamples. We perform two types of experiments:

[11] https://zenodo.org/records/11198697.

1. **Controlled ablations and pooling comparisons:** We keep the preprocessing backbone fixed for all models and parameter choices, varying only one parameter at a time. This includes ablating the specific pooling operations used by SOCK, Rocket, and Hydra, measuring the effect of random bias, and analyzing SOCK's sensitivity to softmax temperature τ and the number of competing kernels k. For these experiments, we use only the `diff` input path augmentation to ensure consistency with the standard time series classification literature.

2. **State-of-the-art comparison:** We compare our best `SOCK` classifier against the top-performing time series classifiers in each model category as identified by Middlehurst et al. [2024]. We separate this from the controlled ablation because the SOTA baselines (like MultiRocket and Hydra) use slightly different internal preprocessing steps than our fixed backbone.

Pooling comparison In the right panel of fig. 4 we compare different choices for the temporal pooling from the Hydra and Rocket literature. To ensure a fair comparison between random convolutional models, we use the exact same pipeline for each, changing only the pooling operation.

Hydra [Dempster et al., 2023] forms two features from each group of competing kernels: it counts how often each kernel wins the local competition and accumulates the corresponding winning response values. In Dempster et al. [2023], these poolings are called "hard counting" and "soft counting". We refer to these two poolings as `h-count` and `h-value` respectively. And we refer to differentiable analogues of these two features, introduced and defined by us in Appendix B.3, as soft-count (`s-count`) and soft-value (`s-value`) respectively. Here, "soft" stands for the softmax operation used as part of these poolings; it is not related to the "soft" terminology used by [Dempster et al., 2023]. We further compare to Rocket's `max` and `ppv` poolings, and MultiRocket's `ppv`, `mpv`, `mipv`, and `lspv` poolings. With some minor abuse of notation, we refer to both the `SOCK` feature map and the resulting classifier as `"SOCK"`.

Single vs. multiple poolings.

We use a single temporal pooling for summarizing the soft kernel competitions in `SOCK`, while most other random convolutional feature maps use multiple poolings. In particular, Hydra concatenates features from two poolings. Concatenating multiple poolings could further improve the `SOCK` feature map, and therefore improve classifiers or generative models trained by feature matching. However, `soft-dev` already performs strongly as a single pooling (fig. 4), and concatenating multiple soft poolings does not always improve performance.

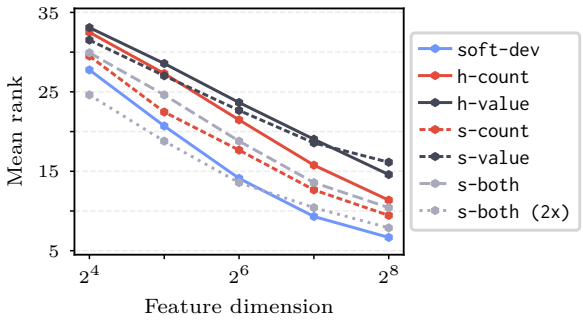

Figure 9: Mean rank on UCR classification tasks for different poolings: `SOCK`'s soft-deviation pooling (`soft-dev`), Hydra's count and value features (`h-count`, `h-value`), the differentiable analogues of Hydra's features (`s-count`, `s-value`), and their concatenation (`s-both`, which concatenates `s-count` and `s-value`). The `s-both` curve is compared at the same output feature dimension as the single-pooling curves, so it uses half as many convolution groups. The `s-both (2x)` curve uses the same number of convolution groups as the single-pooling curves and therefore has twice the output feature dimension.

To see this, consider the result shown in fig. 9, which compares single soft and Hydra-style poolings with concatenated poolings. When matched by the effective number of output features, combining soft-count and soft-value (`s-both`) performs worse than using more soft-count features alone (`s-count`). This comparison is not cost-matched: `s-count` uses more convolution groups to reach the same output dimension. We therefore also compare to `s-both (2x)`, which uses both poolings with the same number of convolution groups as the single-pooling curves and therefore doubles the output dimension. At small feature dimensions, this larger concatenated feature map outperforms `soft-dev`. At larger feature dimensions, the advantage disappears, and `soft-dev` attains the better mean rank.

This comparison is meant to show that adding more poolings does not by itself imply better performance. It does not rule out gains from ensembling several poolings or several independent feature maps, either for classification or for generative modeling. It does show that `soft-dev` is competitive on its own, which motivates our focus on the simpler single-pooling feature map.

Random bias. Figure 10 presents pairwise scatter plots comparing each random pooling mechanism with and without the addition of a random bias term to the convolution output. We observe that `SOCK` and Hydra are, on average, insensitive to the addition of random bias. The `max` pooling is also naturally unaffected. In contrast, `ppv`, `mpv`, `mipv`, and `lspv` rely heavily on the random bias term for performance. Consequently, we include random bias for these pooling methods in our ranking comparisons. As shown in fig. 4, `SOCK` outperforms all other individual pooling operations when other parameters are held fixed. Notably, `SOCK` with only 512 random features matches the performance of the otherwise best pooling `ppv` using 4096 features.

Softmax Temperature τ The temperature τ controls the "softness" of the kernel competition in `SOCK`. As $\tau \to 0$, the operation resembles something similar to the hard `argmax` counting used in Hydra; as $\tau \to \infty$, the weights become uniform. We evaluate `SOCK` for $\tau \in (0.0001, 0.1)$ across different numbers of competing kernels $k \in \{2, 4, 8, 16, 32, 64\}$. We find that if we scale the temperature using the rule $\tau := \log(k)\tau_{base}$, the choice $\tau_{base} = 0.01$ remains optimal on average for all tested values of $k > 2$. These results are illustrated in fig. 11, using the aforementioned scaling rule.

The case $k = 2$ exhibits an interesting phenomenon. Since the softmax weights sum to one, we have that $P_t^{(g,2)} = 1 - P_t^{(g,1)}$. Moreover, because the standard deviation satisfies $\text{std}(X) = \text{std}(1 - X)$, the softmax-deviation pooling yields identical features $F^{(g,1)} = F^{(g,2)}$ for each competing kernel pair. Consequently, $k = 2$ is a degenerate case where the feature map produces redundant copies rather than capturing competitive dynamics, which explains its comparatively poor performance.

Number of Competing Kernels k We further ablate the number of competing kernels k within each group of random convolutional kernels. If g is the number of groups, the total number of random features outputted by the feature map is $k \times g$. We investigate which value of k yields the best classification performance given a fixed feature "budget" (target dimension). In fig. 12, we vary the target dimension and plot the rank of classifiers trained using different values of k. Using the scaling rule $\tau := \log(k)\tau_{base}$ established above, we find that $k = 8$ yields the best classification accuracy, consistent with the findings for Hydra [Dempster et al., 2023].

Comparison to SOTA The left panel of fig. 4 shows a Critical Difference (CD) diagram comparing `SOCK` against the top-performing models identified in the recent "Bake Off Redux" benchmark [Middlehurst et al., 2024]. The diagram presents the average relative rank of each model, indicating statistically significant clusters based on a Wilcoxon signed-rank test with Holm correction. This is a well-established methodology for comparing multiple algorithms across multiple datasets [Demšar, 2006, García and Herrera, 2008, Benavoli et al., 2016, Middlehurst et al., 2024].

We include the single best model from each category presented in [Middlehurst et al., 2024], excluding meta-ensembles (such as HIVE-COTE and HC2) to focus on single-model representations. The selected baselines are: **Proximity Forest** (Distance), **FreshPRINCE** (Feature), **QUANT** (Interval), **RDST** (Shapelet), **WEASEL 2.0** (Dictionary), **MultiRocket** (Random Convolution), and **H-InceptionTime** (Deep Learning). We also include standard **Rocket**, **Hydra**, and **Signatures** for context. We train a classifier with `SOCK` features using $k = 8$, path augmentations (`diff`, `posneg`), and a final number of output features equal to 8192. In the left panel of fig. 4, we observe that `SOCK` achieves the best average rank among all baseline models and is statistically tied with MultiRocket and H-InceptionTime. Crucially, `SOCK` outperforms Hydra and matches MultiRocket while using an order of magnitude fewer random features and a single pooling operation, rather than four. Moreover, since SOCK features are fully differentiable, this opens up for fine-tuning of random convolutional models via SGD, and inclusion within larger deep neural network architectures. We leave these end-to-end applications for future work as they are outside the scope of this manuscript.

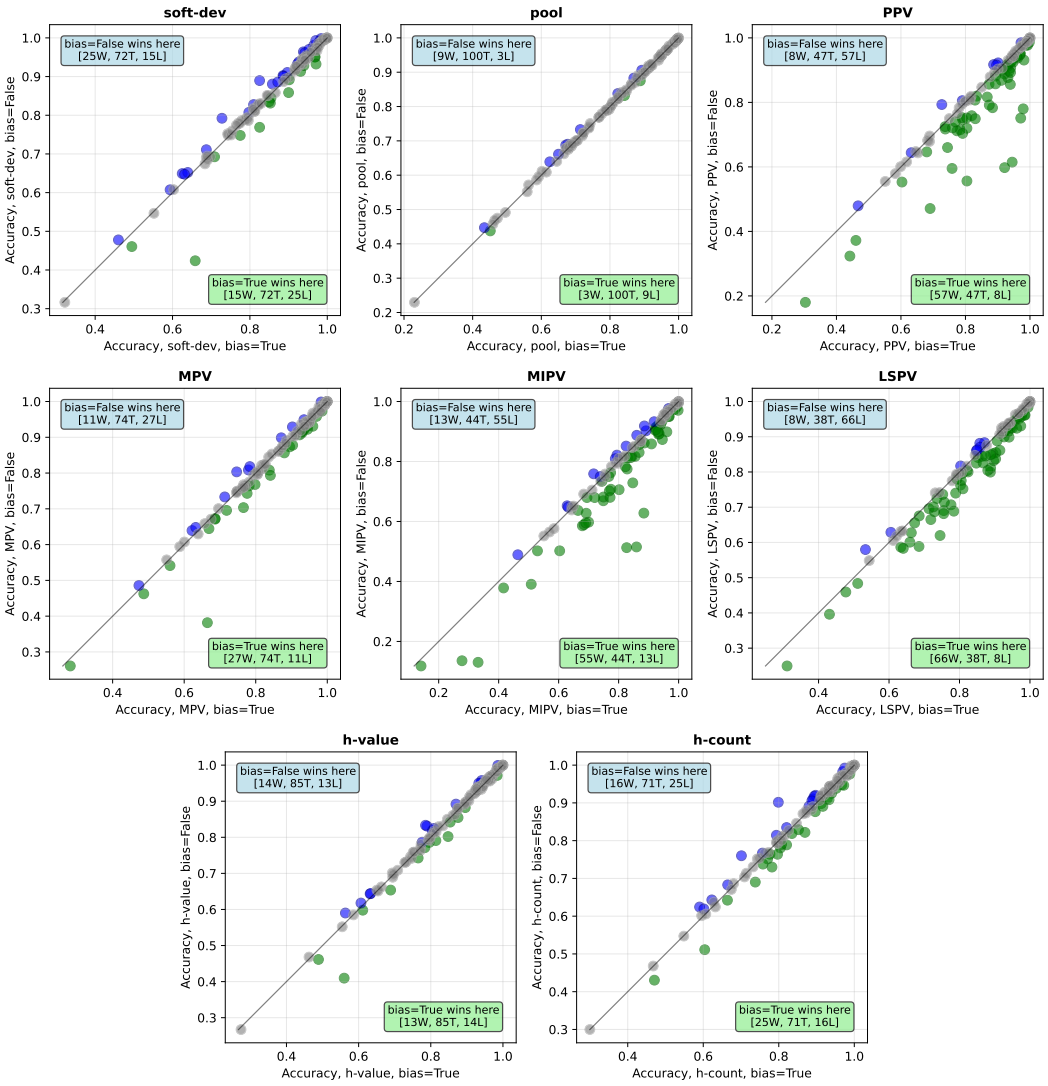

Figure 10: The effects of random bias for each pooling operation used by the random convolutional models SOCK, Rocket, MultiRocket, and Hydra. Results are reported as mean accuracies across 30 resamples on the 112 datasets of the UCR time series classification repository. Each point in the pairwise scatter plot represents one dataset. Accuracies within 0.01 are reported as ties in gray.

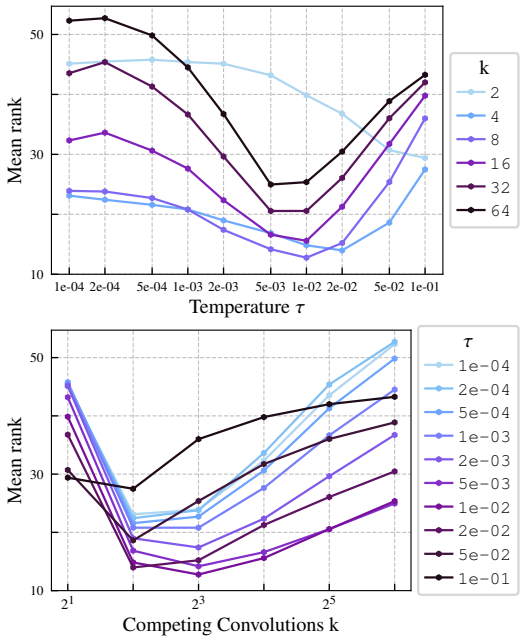

Figure 11: Parameter sensitivity analysis for SOCK on the 112 UCR datasets. The plots display orthogonal slices of the mean rank surface spanned by the softmax temperature τ and number of competing kernels k. **Top:** Mean rank vs. τ for fixed k. **Bottom:** Mean rank vs. k for fixed τ. Results are averaged across 30 resamples of the train and test data.

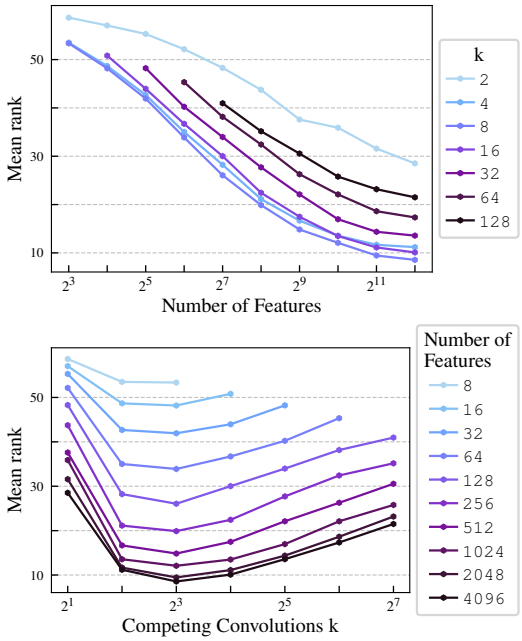

Figure 12: Feature budget analysis on the 112 UCR datasets. The plots correspond to slices of the mean rank surface over the number of competing kernels k and total number of random features (dimension). **Top:** Mean rank vs. total feature dimension for fixed k. **Bottom:** Mean rank vs. k for fixed total feature budgets. Results are averaged across 30 resamples of the train and test data.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and section 1 list the contributions and scope: random convolutional feature matching for conditional time series generation; the SOCK feature map; and evaluation on small-sample generation problems, hypothesis testing, and classification. The supporting method and experiments appear in sections 3 to 5.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 6 explicitly states the limitations of our work. The main limitation is that we evaluate SOCK feature matching only in the specific small-data regime that motivates the paper: conditional generation from a single historical path. We do not study how the method compares or scales with larger datasets, since random convolutional feature matching is designed for this small-data setting.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper does not state formal theoretical results. Like other random convolution feature maps, SOCK is heuristically motivated. Formally studying the expressiveness of such feature maps is important future work.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The paper specifies the SOCK feature map and training objective in sections 3 and 3.3, and gives implementation details, pseudocode, data construction, evaluation metrics, generator architecture, optimization settings, and baseline details in sections B and C.1 to C.4. The hypothesis-testing and classification experiments are specified in sections D.1 and D.2.

   Guidelines:

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

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

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

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

   Answer: [Yes]

   Justification: The code is publicly available at `https://github.com/konmue/sock`.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Sections 4 and 4.1 and sections C.1 to C.4 specify the data splits, training and evaluation protocol, feature-map settings, generator architecture, optimizer, learning-rate schedule, and baseline configurations. We use the same hyperparameters across all generative experiments, which avoids tuning to the very small real datasets. Details for the discriminative experiments are in sections D.1 and D.2.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

Justification: The main generative results in the tables of Appendix A report means and standard deviations for all metrics over five seeds, with section 4.1 specifying what the seeds vary on synthetic and real datasets. The main classification result in fig. 4 is reported as a critical-difference diagram, with the Wilcoxon signed-rank test and Holm correction described in section D.2.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Section C.3 reports the GPU, memory, approximate runtime per dataset and seed, sampling time, and main training bottleneck for the generative experiments.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We believe our research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Improved financial time series generation from small datasets can support risk analysis and stress testing, but risks overreliance if model uncertainty is ignored; we discuss this in section C.2.1.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release high-risk data or models.

    Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: The paper credits the external datasets and baselines used. Sections C.1 and D.2 state the relevant terms or licenses for Yahoo Finance, Binance, and UCR data.

Guidelines:

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

13. **New assets**

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

Answer: [N/A]

Justification: The paper does not release new assets.

Guidelines:

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

14. **Crowdsourcing and research with human subjects**

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

Answer: [N/A]

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

Guidelines:

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

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

Answer: [N/A]

Justification: The paper has no study participants, so IRB approval or equivalent review is not applicable.

Guidelines:

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

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method does not use LLMs as an important, original, or non-standard component.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.