pySigLib - Fast Signature Kernels on CPU and GPU

Anonymous Author(s)
Affiliation
Address
email

Abstract

Signature-based methods have recently gained significant traction in machine learning for sequential data. In particular, signature kernels have emerged as powerful discriminators and training losses for generative models on time-series, notably in quantitative finance. However, existing implementations do not scale to the dataset sizes and sequence lengths encountered in practice. We present `pySigLib`, a high-performance library offering optimised implementations of signatures and signature kernels on CPU and GPU, fully compatible with PyTorch's automatic differentiation. Beyond an efficient software stack for large-scale signature-kernel computation, we introduce a novel differentiation scheme that delivers accurate gradients at a fraction of the runtime of existing libraries. We believe our library unlocks the practical use of signature kernels for generative modelling and the distillation of next-generation sequence models.

1 Introduction

Most forms of sequential data such as financial time series, on sufficiently fine time scales, can be represented as a continuous path $x : [0, 1] \to \mathbb{R}^d$. It was first shown by [8], and then explored in greater detail and generality in the context of *rough path theory* in [32, 19, 5], that any path may be faithfully represented, up to reparameterisation, by the collection of its iterated integrals known as the *path-signature*. The path-signature $S(x)$ is formally defined as the solution to the tensor differential equation $dy_t = y_t \otimes dx_t$ in the *free tensor algebra* $T((\mathbb{R}^d)) := \prod_{n=0}^{\infty}(\mathbb{R}^d)^{\otimes n}$, and thereby can be interpreted as a non-commutative analogue of the exponential function. By applying Picard iteration, one recovers the familiar formulation of the path-signature as the sequence of iterated integrals $S(x) = (\int_{0<t_1<\cdots<t_n<1} dx_{t_1} \otimes \cdots \otimes dx_{t_n})_{n\in\mathbb{N}}$. The Stone–Weierstrass theorem ensures that linear functionals acting on the range of the path-signature are dense in the space of continuous real-valued functions defined on compact subsets of *unparameterised paths*. This makes the signature a powerful representation, enabling the approximation of path-dependent functionals using linear models.

Thanks to this *universal approximation property* and a variety of other algebraic features, signature methods have experienced a rapid rise in popularity in recent years, with applications across a wide range of data science domains, including quantitative finance [40, 13, 1, 13, 4], cybersecurity [11], information theory [43, 45, 48], and even quantum computing [12]. Signatures have also served as a theoretical foundation for proving universality properties of *neural differential equations* [34, 2] and more recently of *state-space models* (SSMs) [35, 36, 49]. In generative modelling, signatures have enabled new approaches to synthesizing financial time series in a model-independent manner [6], been used as universal nonlinearities in Seq2Seq architectures [24], and provided representation spaces for training *score-based diffusion models* on time series [3]. For a detailed and pedagogical introduction to the subject, the reader is referred to [7], while [16] offers a survey of recent applications.

In practice, paths are typically obtained via piecewise linear interpolation of discrete time series. Combining a non-trivial algebraic relation known as *Chen's identity* with the elementary fact that the signature of a linear segment is the tensor exponential of its increment, yields a concise expression for the signature of a piecewise linear path $x = x^1 * \cdots * x^L$ as $S(x) = \exp(\Delta x^1) \otimes \cdots \otimes \exp(\Delta x^L)$.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

This expression underlies the implementation of signature computations in standard Python libraries such as `esig` [31], `iisignature` [42], and `signatory` [26]. It enables an efficient evaluation of the signature with time complexity $\mathcal{O}(Ld^n)$, where $n \in \mathbb{N}$ is the truncation level. However, due to the exponential growth in the dimension d, the method becomes computationally prohibitive for large n.

A widely adopted solution to this curse of dimensionality is the use of *signature kernels* [27]. These kernels take the form $\langle S(x), S(y)\rangle$, for suitable inner products $\langle \cdot, \cdot \rangle$ on $T((\mathbb{R}^d))$, and can be computed efficiently without explicitly evaluating the feature map $S(x)$. In particular, recent work has shown that the signature kernel satisfies a Goursat PDE [44, 28]. Signature kernels have since found applications in hypothesis testing [47, 30, 22], causality [33], kernel-based solvers for path-dependent PDEs in derivative pricing under rough volatility [38], kernel formulations of deep hedging [37], and have even been shown to arise as infinite-width limits of neural networks [10]. They have also been used to train neural SDEs for time-series generation across diverse fields, including fluid dynamics [46], computational neuroscience [21], and quantitative finance [23, 15, 20].

Despite these advances, **current software implementations of signature kernels do not scale to large datasets of long time series, particularly common in quant finance**. Moreover, when signature kernels are employed as loss functions, efficient and accurate backpropagation is crucial. Existing implementations compute derivatives using a second PDE [29]; while natural, this approach often yields inaccurate gradients, especially for short time series, leading to unreliable model training.

Contribution We introduce `pySigLib`**, a high-performance computational library offering optimised CPU- and GPU-amenable implementations of signatures and signature kernels in C++ and CUDA, fully compatible with PyTorch's automatic differentiation**. `pySigLib` is significantly faster than existing packages on both CPU and GPU thanks to algorithmic improvements, optimized memory access, and hardware-level parallelism via SIMD instructions. It also supports efficient on-the-fly application of lead–lag and time-augmentation transformations, providing a substantial speed-up in financial applications. Furthermore, all functions are fully backpropagatable through a PyTorch API. The library is open source and available at `https://github.com/daniil-shmelev/pySigLib` , installable directly via `pip`, with documentation hosted at `https://pysiglib.readthedocs.io`.

2 Computation of signature kernels

We begin with a brief account of signature kernels and numerical techniques to compute them.

2.1 Signature kernels as the solution of a Goursat PDE

A key result by [44] established that signature kernels can be computed as the solution of a Goursat-type PDE. Notably, this result states that given two continuous paths $x, y : [0,1] \to \mathbb{R}^d$, the corresponding signature kernel $k_{x,y}(s,t) := \langle S(x)_s, S(y)_t \rangle$ satisfies the hyperbolic PDE

$$\frac{\partial^2 k_{x,y}}{\partial s \partial t} = \langle \dot{x}_s, \dot{y}_t \rangle_V \, k_{x,y}, \quad k_{x,y}(u, \cdot) = k_{x,y}(\cdot, v) = 1.$$

An effective discretisation of the above PDE is given by [14, 44, 50]

$$\begin{aligned} \widehat{\mathbf{k}}_{i+1,j+1} &= \left(\widehat{\mathbf{k}}_{i+1,j} + \widehat{\mathbf{k}}_{i,j+1}\right) A(\Delta_{i,j}) - \widehat{\mathbf{k}}_{i,j} B(\Delta_{i,j}), \\ A(\Delta_{i,j}) &= \left(1 + \frac{1}{2}\Delta_{i,j} + \frac{1}{12}\Delta_{i,j}^2\right), \quad B(\Delta_{i,j}) = \left(1 - \frac{1}{12}\Delta_{i,j}^2\right) \\ \Delta_{i,j} &= \left\langle x_{t_{i+1}} - x_{t_i}, y_{s_{j+1}} - y_{s_j} \right\rangle \end{aligned} \tag{1}$$

over the dyadically refined grid $P_{\lambda_1,\lambda_2} = \{(t_i, s_j) : 0 \le i \le 2^{\lambda_1} L_1,\ 0 \le j \le 2^{\lambda_2} L_2\}$ of order (λ_1, λ_2), where L_1, L_2 are the lengths of the discrete data streams x, y respectively.

2.2 An algorithm for signature kernels

Implementations of signature kernels exist for both CPU and GPU, most notably via the `sigkernel` and `sigkerax` packages [44]. The resulting algorithm is outlined in 1. In our implementation, several choices are notable: (1) as opposed to other packages, we allow $\lambda_1 \neq \lambda_2$, which can prove

useful when x and y are of significantly different lengths; (2) all values of $\Delta_{i,j}$ required for (1) are precomputed using a `matmul` operation. In cases where the dimension of the underlying path is large, this operation accounts for almost all of the algorithm's runtime, so it is critical that it is highly optimized. In `pySigLib`, this is realised using PyTorch's `bmm` function [39]; and (3) whereas other packages implementing signature kernels precompute the dyadically refined path, `pySigLib` applies the dyadic refinement on-the-fly, improving both computational and memory efficiency.

Algorithm 1 CPU Algorithm for Signature Kernels

Input: $\{x_i\}_{1\leq i\leq L_1}, \{y_i\}_{1\leq i\leq L_2}, \lambda_1, \lambda_2$ ▷ See (1) above

$dx = x_{1:L_1} - x_{0:L_1-1}; \quad dy = y_{1:L_1} - y_{0:L_1-1}$

$\Delta = \texttt{matmul}(dx^T, dy)$ ▷ See (2) above

for $s = 0, \dots, 2^{\lambda_1}L_1 - 1; \quad t = 0, \dots, 2^{\lambda_2}L_2 - 1$ **do**

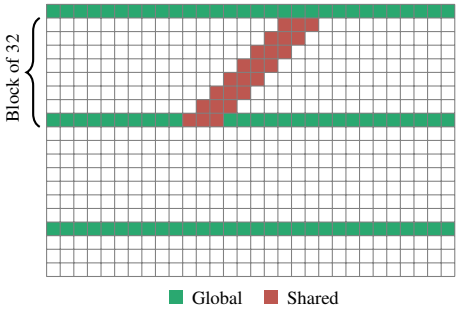
$p = \Delta[s \mathbin{/\!\!/} 2^{\lambda_1}, t \mathbin{/\!\!/} 2^{\lambda_1}]$ ▷ See (3) above

$\widehat{\mathbf{k}}[s,t] = (\widehat{\mathbf{k}}[s-1,t] + \widehat{\mathbf{k}}[s,t-1]) * (1 + \frac{1}{2}p + p^2) - \widehat{\mathbf{k}}[s-1,t-1] * (1 - p^2)$

end for

return $\widehat{\mathbf{k}}[-1,-1]$

2.3 Exploiting GPU parallelism



As suggested in [44], the algorithm can be effectively parallelised on a GPU, by noting that elements on any given anti-diagonal of the PDE grid exhibit no interdependencies, and so can be computed in parallel. Following this approach, we note that it is not necessary to store the entire PDE grid. Rather, one can store only the current anti-diagonal and the two before it. As the algorithm progresses, the three anti-diagonals are "rotated", so that the first anti-diagonal becomes the second, the second becomes the third, and the third is overwritten with new values to become the first. This dramatically reduces the required memory allocation, allowing for the three anti-diagonals to be kept in shared memory of the GPU, which has a significantly lower latency than the global memory which a full PDE grid would be forced to reside in.

As of writing, when using other signature kernel libraries for GPU computations, the length of the input streams is severely limited by the thread count of the GPU, as the algorithm attempts to assign one thread per entry of the diagonal. We avoid this problem by computing kernels in blocks of 32. For the first such block, a vector of initial conditions in global memory is populated with all ones, acting as the first row of the PDE grid. As the block progresses, the initial condition is overwritten by the final row in the block, which becomes the initial condition for the second block of 32. As such, one is never limited by the thread count, as only 32 threads are allocated per kernel computation. Naturally, this means the GPU is underutilized when computing a single kernel. However, when a batch of kernels are computed, blocks associated with different kernels are allowed to run asynchronously with respect to each other, meaning all available threads are used when the batch size is sufficiently large. We anticipate this should be the case in almost all practical applications.

2.4 Backpropagation through signature kernels

Existing methods for backpropagation approximate derivatives as the solution to another PDE [29]. The downside to this is that the gradients are not exact. When either the length of the underlying path or the dyadic order is too low, this difference is significant. To avoid this issue, we compute exact gradients by differentiating through the operations of the solver (1). As we will see, the resulting algorithm is significantly faster than the PDE approach, whilst producing exact gradients. Given a scalar function F and its derivative with respect to the signature kernel $\partial F / \partial \widehat{\mathbf{k}}_{L_1,L_2}$, we aim to compute $\partial F / \partial \Delta_{a,b}$ for all indices a, b, which can then easily be used to compute the derivatives with respect to the underlying paths x and y via a linear transformation. A naive approach may be to try and differentiate the above relation directly with respect to $\Delta_{a,b}$ to try and form an update rule for $\partial \widehat{\mathbf{k}}_{i,j} / \partial \Delta_{a,b}$. This approach would require traversing a PDE grid for each pair a, b, resulting in a

serial complexity of $\mathcal{O}(2^{\lambda_1+\lambda_2}L_1^2L_2^2)$. One can, however, achieve the same result in only one traversal of a PDE grid, by targeting instead $\partial F/\partial\widehat{\mathbf{k}}_{a,b}$. For simplicity, assume our grid is not dyadically refined, i.e. $\lambda_1 = \lambda_2 = 0$. For convenience, define $\partial F/\partial\widehat{\mathbf{k}}_{i,j} = 0$ if $i > L_1$ or $j > L_2$. Then we have the following relations, computable in one backward pass of a PDE grid:

$$\frac{\partial F}{\partial \widehat{\mathbf{k}}_{i,j}} = \frac{\partial F}{\partial \widehat{\mathbf{k}}_{i+1,j}} A(\Delta_{i,j-1}) + \frac{\partial F}{\partial \widehat{\mathbf{k}}_{i,j+1}} A(\Delta_{i-1,j}) - \frac{\partial F}{\partial \widehat{\mathbf{k}}_{i+1,j+1}} B(\Delta_{i,j}),$$

$$\frac{\partial F}{\partial \Delta_{i,j}} = \frac{\partial F}{\partial \widehat{\mathbf{k}}_{i+1,j+1}} \left[\left(\widehat{\mathbf{k}}_{i+1,j} + \widehat{\mathbf{k}}_{i,j+1} \right) \left(\frac{1}{2} + \frac{1}{6}\Delta_{i,j} \right) + \frac{1}{6}\widehat{\mathbf{k}}_{i,j}\Delta_{i,j} \right].$$

The same approach applies to dyadically refined grids, noting that in this case there are multiple indices a, b for which $\widehat{\mathbf{k}}_{a,b}$ depends on $\Delta_{i,j}$.

Algorithm 2 CPU Algorithm for Backpropagation through Signature Kernels

Input: $x_i \in \mathbb{R}^d$, $1 \le i \le L_1$ and $y_i \in \mathbb{R}^d$, $1 \le i \le L_2$
Input: Dyadic order for x, λ_1, and dyadic order for y, λ_2
Input: Dyadically refined kernel grid $\widehat{\mathbf{k}}$, precomputed or computed on-the-fly.
Input: Derivative $\partial F/\partial\widehat{\mathbf{k}}[s,t]$.
$dx = x_{1:L_1} - x_{0:L_1-1}, \quad dy = y_{1:L_1} - y_{0:L_1-1}; \quad \Delta = \texttt{matmul}(dx^T, dy)$
$\Delta = \texttt{matmul}(dx^T, dy)$
for $s = 2^{\lambda_1}L_1 - 1, \dots, 0; \quad t = 2^{\lambda_2}L_2 - 1, \dots, 0$ **do**
  $p_{i,j} = \Delta[i \,//\, 2^{\lambda_1}, j \,//\, 2^{\lambda_2}]$ for $(i,j) = (s,t), (s-1,t), (s,t-1)$
  $d_1[s,t] = (d_1[s+1,t] * A(p_{s,t-1}) + d_1[s,t+1] * A(p_{s-1,t}) - d_1[s+1,t+1] * B(p_{s,t}))$
  $d_2[s \,//\, 2^{\lambda_1}, t \,//\, 2^{\lambda_2}] \mathrel{+}= d_1[s,t] * \{(\widehat{\mathbf{k}}[s+1,t] + \widehat{\mathbf{k}}[s,t+1]) * A'(p_{s,t}) - \widehat{\mathbf{k}}[s,t] * B'(p_{s,t})\}$
end for
return d_2 ▷ Result is $\partial F/\partial\Delta$, which can be transformed into $\partial F/\partial x$ or $\partial F/\partial y$

Table 1 and Figure 1 show the runtime of signature kernels and their backpropagation. A 14-core i7-13700H CPU with 32GB of RAM and an NVIDIA RTX 4060 GPU are used. The minimum runtime is taken over 50 runs. Results are compared to the `sigkernel` package, which is limited by the thread count (1024) on GPU and memory on CPU.

(B, L, d)	Forward (CPU)		Forward (GPU)		Backward (CPU)		Backward (GPU)	
	sigkernel	pySigLib	sigkernel	pySigLib	sigkernel	pySigLib	sigkernel	pySigLib
(128, 256, 8)	0.3610	**0.0118**	0.0088	**0.0034**	2.3248	**0.0322**	0.1475	**0.0057**
(128, 512, 16)	1.3608	**0.0364**	0.0376	**0.0117**	15.0445	**0.1276**	15.3843	**0.0231**
(128, 1024, 32)	6.0874	**0.2325**	-	**0.0653**	-	**0.6019**	-	**0.1112**

Table 1: *Runtime (seconds) for a batch of B paths of length L and dimension d.*

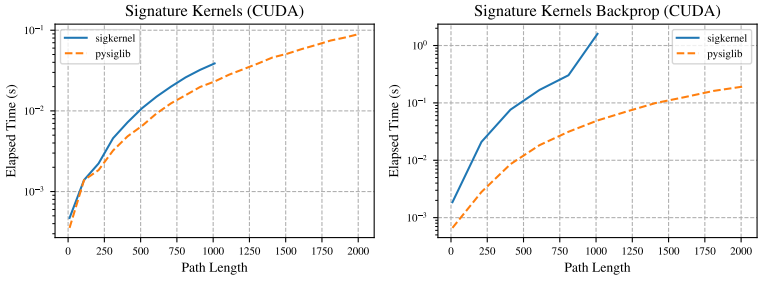


Figure 1: *Runtime (seconds) for a batch of 32 paths of dimension 5.*

3 Conclusion

We have introduced `pySigLib`, a computational library designed to overcome the challenges of applying signature-based methods to large-scale data. Our results show significant improvements in the computation of signature kernels, allowing for efficient processing of long time-series data. We presented a novel approach to backpropagation through signature kernels, which achieves accurate gradients at a fraction of the runtime of existing implementations. Concrete experiments integrating `pySigLib` into deep learning pipelines are beyond the scope of this paper and left for future work.

References

[1] E. Abi Jaber and L.-A. Gérard. Signature volatility models: pricing and hedging with fourier. *SIAM Journal on Financial Mathematics*, 16(2):606–642, 2025.

[2] I. P. Arribas, C. Salvi, and L. Szpruch. Sig-sdes model for quantitative finance. In *ACM International Conference on AI in Finance*, 2020.

[3] B. Barancikova, Z. Huang, and C. Salvi. Sigdiffusions: Score-based diffusion models for time series via log-signature embeddings. In *The Thirteenth International Conference on Learning Representations*.

[4] C. Bayer, L. Pelizzari, and J.-J. Zhu. Pricing american options under rough volatility using deep-signatures and signature-kernels. *arXiv preprint arXiv:2501.06758*, 2025.

[5] H. Boedihardjo, X. Geng, T. Lyons, and D. Yang. The signature of a rough path: uniqueness. *Advances in Mathematics*, 293:720–737, 2016.

[6] H. Buehler, B. Horvath, T. Lyons, I. P. Arribas, and B. Wood. A data-driven market simulator for small data environments. *arXiv preprint arXiv:2006.14498*, 2020.

[7] T. Cass and C. Salvi. Lecture Notes on Rough Paths and Applications to Machine Learning, 2024.

[8] K.-T. Chen. Integration of paths, geometric invariants and a generalized baker-hausdorff formula. *Annals of Mathematics*, pages 163–178, 1957.

[9] I. Chevyrev and A. Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.

[10] N. M. Cirone, M. Lemercier, and C. Salvi. Neural signature kernels as infinite-width-depth-limits of controlled resnets. In *International Conference on Machine Learning*, pages 25358–25425. PMLR, 2023.

[11] T. Cochrane, P. Foster, V. Chhabra, M. Lemercier, T. Lyons, and C. Salvi. Sk-tree: a systematic malware detection algorithm on streaming trees via the signature kernel. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 35–40. IEEE, 2021.

[12] S. Crew, C. Salvi, W. F. Turner, T. Cass, and A. Jacquier. Quantum path signatures. *arXiv preprint arXiv:2508.05103*, 2025.

[13] C. Cuchiero and J. Möller. Signature methods in stochastic portfolio theory. *arXiv preprint arXiv:2310.02322*, 2023.

[14] J. T. Day. A runge-kutta method for the numerical solution of the goursat problem in hyperbolic partial differential equations. *Comput. J.*, 9(1):81–83, 1966.

[15] P. Díaz, T. Lozano, and J. Vives i Santa Eulàlia. Neural stochastic differential equations for conditional time series generation using the signature wasserstein-1 metric. *Journal Of Computational Finance, 2023, vol. 27, num. 1, p. 1-23*, 2023.

[16] A. Fermanian, T. Lyons, J. Morrill, and C. Salvi. New directions in the applications of rough path theory. *IEEE BITS the Information Theory Magazine*, 3(2):41–53, 2023.

[17] G. Flint, B. Hambly, and T. Lyons. Discretely sampled signals and the rough hoff process. *Stochastic Processes and their Applications*, 126(9):2593–2614, 2016.

[18] L. G. Gyurkó, T. Lyons, M. Kontkowski, and J. Field. Extracting information from the signature of a financial data stream. *arXiv preprint arXiv:1307.7244*, 2013.

[19] B. Hambly and T. Lyons. Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, pages 109–167, 2010.

[20] M. Hoglund, E. Ferrucci, C. Hernández, A. M. Gonzalez, C. Salvi, L. Sánchez-Betancourt, and Y. Zhang. A neural rde approach for continuous-time non-markovian stochastic control problems. *arXiv preprint arXiv:2306.14258*, 2023.

[21] C. Holberg and C. Salvi. Exact gradients for stochastic spiking neural networks driven by rough signals. *Advances in Neural Information Processing Systems*, 37:31907–31939, 2024.

[22] B. Horvath, M. Lemercier, C. Liu, T. Lyons, and C. Salvi. Optimal stopping via distribution regression: a higher rank signature approach, 2023.

[23] Z. Issa, B. Horvath, M. Lemercier, and C. Salvi. Non-adversarial training of neural sdes with signature kernel scores. *Advances in Neural Information Processing Systems*, 36:11102–11126, 2023.

[24] P. Kidger, P. Bonnier, I. Perez Arribas, C. Salvi, and T. Lyons. Deep signature transforms. *Advances in neural information processing systems*, 32, 2019.

[25] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. *arXiv preprint arXiv:2001.00706*, 2020.

[26] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU, 2021.

[27] F. J. Király and H. Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 20(31):1–45, 2019.

[28] M. Lemercier, T. Lyons, and C. Salvi. Log-pde methods for rough signature kernels. *arXiv preprint arXiv:2404.02926*, 2024.

[29] M. Lemercier, C. Salvi, T. Cass, E. V. Bonilla, T. Damoulas, and T. J. Lyons. Siggpde: Scaling sparse gaussian processes on sequential data. In *International Conference on Machine Learning*, pages 6233–6242. PMLR, 2021.

[30] M. Lemercier, C. Salvi, T. Damoulas, E. Bonilla, and T. Lyons. Distribution regression for sequential data. In *International Conference on Artificial Intelligence and Statistics*, pages 3754–3762. PMLR, 2021.

[31] T. Lyons and al. Coropa computational rough paths (software library), 2010.

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

[33] G. Manten, C. Casolo, E. Ferrucci, S. W. Mogensen, C. Salvi, and N. Kilbertus. Signature kernel conditional independence tests in causal discovery for stochastic processes. In *The Thirteenth International Conference on Learning Representations*.

[34] J. Morrill, C. Salvi, P. Kidger, and J. Foster. Neural rough differential equations for long time series. In *International Conference on Machine Learning*, pages 7829–7838. PMLR, 2021.

[35] N. Muça Cirone, A. Orvieto, B. Walker, C. Salvi, and T. Lyons. Theoretical foundations of deep selective state-space models, 2024.

[36] N. Muça Cirone and C. Salvi. Parallelflow: Parallelizing linear transformers via flow discretization, 2025.

[37] N. Muça Cirone and C. Salvi. Rough kernel hedging, 2025.

[38] A. Pannier and C. Salvi. A path-dependent pde solver based on signature kernels, 2024.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[40] I. Perez Arribas. *Signatures in machine learning and finance*. PhD thesis, University of Oxford, 2020.

[41] J. Reizenstein. Iterated-integral signatures in machine learning. 2019.

[42] J. Reizenstein and B. Graham. The iisignature library: efficient calculation of iterated-integral signatures and log signatures. *arXiv preprint arXiv:1802.08252*, 2018.

[43] C. Salvi. *Rough paths, kernels, differential equations and an algebra of functions on streams*. PhD thesis, University of Oxford, 2021.

[44] C. Salvi, T. Cass, J. Foster, T. Lyons, and W. Yang. The signature kernel is the solution of a goursat pde. *SIAM Journal on Mathematics of Data Science*, 3(3):873–899, 2021.

[45] C. Salvi, J. Diehl, T. Lyons, R. Preiss, and J. Reizenstein. A structure theorem for streamed information. *Journal of Algebra*, 634:911–938, 2023.

[46] C. Salvi, M. Lemercier, and A. Gerasimovics. Neural stochastic PDEs: Resolution-invariant learning of continuous spatiotemporal dynamics. *Advances in Neural Information Processing Systems*, 35:1333–1344, 2022.

[47] C. Salvi, M. Lemercier, C. Liu, B. Horvath, T. Damoulas, and T. Lyons. Higher order kernel mean embeddings to capture filtrations of stochastic processes. *Advances in Neural Information Processing Systems*, 34:16635–16647, 2021.

[48] D. Shmelev and C. Salvi. Sparse signature coefficient recovery via kernels, 2024.

[49] B. Walker, L. Yang, N. M. Cirone, C. Salvi, and T. Lyons. Structured linear cdes: Maximally expressive and parallel-in-time sequence models. *arXiv preprint arXiv:2505.17761*, 2025.

[50] A.-M. Wazwaz. On the numerical solution of the goursat problem. *Applied mathematics and computation*, 59(1):89–95, 1993.

A Runtime for signature kernels on CPU

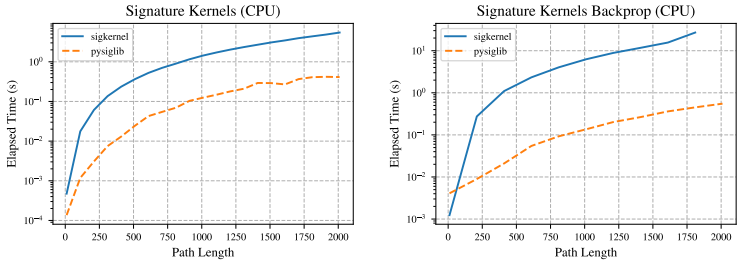


Figure 2: *Runtime (seconds) for signature kernels and backpropagation, computed for a batch of 32 paths of dimension 5 on a i7-13700H CPU and an NVIDIA RTX 4060 GPU. The minimum time is taken over 50 runs. The* `sigkernel` *package is limited by the thread count (1024) on GPU.*

B Path Transformations

Given a path, it is common to apply certain path-to-path transformations before computing a transform such as the signature. The most common path-to-path transformations are time-augmentation and the lead-lag transform. pySigLib provides backpropagatable implementations of these transforms. Where possible, pySigLib is capable of adapting algorithms internally to perform these transformations on-the-fly, which is often faster and more memory-efficient than pre-computing them. Given a path consisting of points $x_{t_i} \in \mathbb{R}^d$, the time augmented path is defined by $\hat{x}_{t_i} := (x_{t_i}, t_i) \in \mathbb{R}^{d+1}$. Time-augmentation is useful for introducing time-reparametrisation variance to the signature, in applications where the timing of events is critical. The lead-lag transformation is defined by $X^{LL}_{t_i} := (X^{\text{Lead}}_{t_i}, X^{\text{Lag}}_{t_i})$, where [9, 18, 17]

$$X^{\text{Lead}}_{t_i} := \begin{cases} X_{t_k} & \text{if } i = 2k, \\ X_{t_k} & \text{if } i = 2k-1, \end{cases}$$

$$X^{\text{Lag}}_{t_i} := \begin{cases} X_{t_k} & \text{if } i = 2k, \\ X_{t_k} & \text{if } i = 2k+1, \end{cases}$$

The lead-lag transform is particularly useful for feature-extraction applied to financial data streams, where the signature of the lead-lag path may be used as an approximation to the Itô-signature [17, 18].

C Truncated Signatures

We implement and test two algorithms for computing truncated signatures. The first of these is the direct approach, as used by `iisignature` [42]. Suppose we are given as input a path consisting of points $x_i \in \mathbb{R}^d, 1 \leq i \leq L$, and a truncation level N for the resulting signature. Suppose we have already used the first ℓ of these points to construct the signature $S(x_{1:\ell}) = (A_0, A_1, \dots, A_N)$. Then the update rule for the next step of the algorithm is given by

$$A_k = \sum_{i=0}^{k} A_i \otimes \frac{z^{\otimes(k-i)}}{(k-i)!}$$

where $z := x_{\ell+1} - x_\ell$. The resulting algorithm is outlined in 3. Given the computational complexity of signature computations, it is particularly important to minimise costly memory allocation and access operations within the algorithm. Two major design choices allow us to do this:

1. $(A_0, A_1, \dots, A_N)$ is stored as a single flattened contiguous array instead of using separate containers for each level, improving cache locality.
2. Since the update for any given level A_k depends only on A_i for $i < k$, the levels are updated in reverse order, starting at A_N and ending at A_1, allowing these to be written directly into the existing memory for $(A_0, A_1, \dots, A_N)$ without the need for storing intermediary computations.

Algorithm 3 Direct Algorithm for Truncated Signatures

Input: $x_i \in \mathbb{R}^d, 1 \leq i \leq L$
Input: Truncation level N

```
z = x_1 − x_0
(A_0, A_1, ..., A_N) = (1, z, ..., z^{⊗N}/N!)          ▷ See (1) above
for ℓ = 1, ..., L − 1 do
    z = x_{ℓ+1} − x_ℓ
    (Ã_0, Ã_1, ..., Ã_N) = (1, z, ..., z^{⊗N}/N!)
    for k = N, ..., 1 do                                ▷ See (2) above
        for i = k − 1, ..., 1 do
            A_k = A_k + A_i ⊗ Ã_{k−i}
        end for
        A_k = A_k + Ã_k
    end for
end for
```

Despite these optimisations, the direct algorithm remains complex, and is largely bound by costly memory access operations. These issues are somewhat improved by Horner's algorithm for polynomial multiplication, as used by `signatory` [25]. In Horner's algorithm, the update rule is rewritten as

$$\begin{aligned} A_k &= \sum_{i=0}^{k} A_i \otimes \frac{z^{\otimes(k-i)}}{(k-i)!} \\ &= \left(\left(\cdots\left(\left(\frac{z}{k} + A_1\right) \otimes \frac{z}{k-1} + A_2\right) \otimes \frac{z}{k-2} + \cdots\right) \otimes \frac{z}{2} + A_{k-1}\right) \otimes z + A_k, \end{aligned}$$

which minimises the number of multiplications necessary, and reduces the number of memory accesses to the right-hand tensor. In our implementation of Horner's algorithm, it will be useful to split the update rule as

$$\begin{aligned} A_k &= (B_k + A_{k-1}) \otimes z + A_k \\ B_k &:= \left(\cdots\left(\left(\frac{z}{k} + A_1\right) \otimes \frac{z}{k-1} + A_2\right) \otimes \frac{z}{k-2} + \cdots\right) \otimes \frac{z}{2} \end{aligned}$$

to allow for additional memory access and allocation optimisations, as we will see below. The resulting algorithm is outlined in 4.

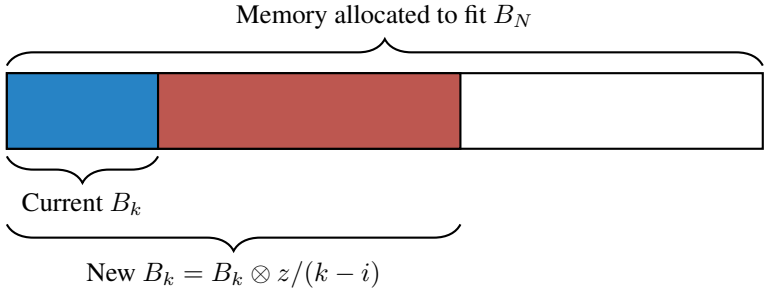
Algorithm 4 Horner's Algorithm for Truncated Signatures

Input: $x_i \in \mathbb{R}^d, 1 \leq i \leq L$
Input: Truncation level N

$z = x_1 - x_0$
$(A_0, A_1, \ldots, A_N) = (1, z, \ldots, z^{\otimes N}/N!)$ ▷ See (1) above
for $\ell = 1, \ldots, L-1$ **do**
  $z = x_{\ell+1} - x_\ell$
  for $k = N, \ldots, 2$ **do** ▷ See (2) above
    $B_k = z/k$
    for $i = 1, \ldots, k-2$ **do**
      $B_k = B_k + A_i$
      $B_k = B_k \otimes z/(k-i)$ ▷ See (3) below
    end for
    $B_k = B_k + A_{k-1}$
    $A_k = B_k \otimes z + A_k$ ▷ See (4) below
  end for
end for

The two design choices of Algorithm 3 also apply here. In addition, the following two choices are of particular importance:

3. A single, continuous block of memory is pre-allocated to fit B_N, and re-used by all intermediate computations involving B_k. The multiplication $B_k = B_k \otimes z/(k-i)$ is written directly into the same memory, but carried out in reverse order such that the new values of B_k (of which there are now d times as many) erase the old values only at the very end of the multiplication, when they are no longer required.



4. The last multiplication and addition $B_k \otimes z + A_k$ is written directly into the result, A_k.

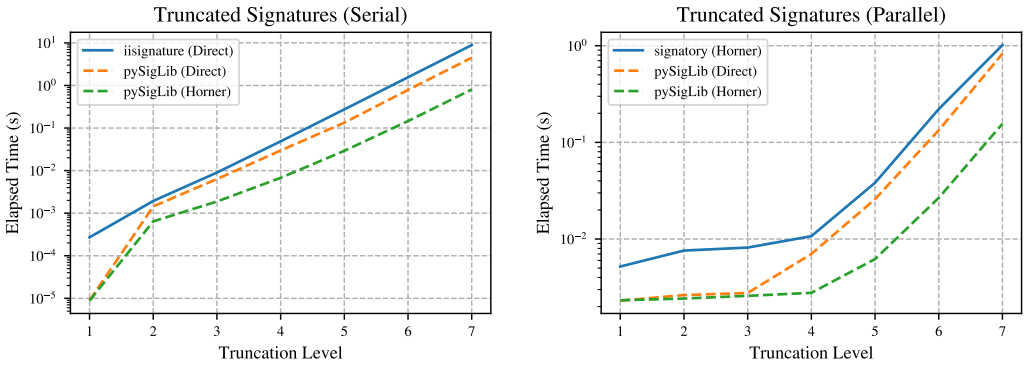


Figure 3: *Timings for a batch of* 32 *paths of length* 1024*, dimension* 5*. The minimum time is taken over* 50 *runs. Ran on a i7-13700H CPU.*

Elapsed Time (s)	CPU (Serial)			CPU (Parallel)	
	esig	iisignature	pySigLib	signatory	pySigLib
Signature	0.9093	0.2441	**0.0227**	0.0272	**0.0050**

Table 2: *Timings for a batch of* 32 *paths of length* 128*, dimension* 4*, and a truncation level of* 7*. The minimum time is taken over* 50 *runs. Ran on a i7-13700H CPU.*

Elapsed Time (s)	CPU (Serial)			CPU (Parallel)	
	esig	iisignature	pySigLib	signatory	pySigLib
Signature	9.0908	3.6238	**0.2805**	0.3423	**0.0590**

Table 3: *Timings for a batch of* 32 *paths of length* 1024*, dimension* 16*, and a truncation level of* 4*. The minimum time is taken over* 50 *runs. Ran on a i7-13700H CPU.*

C.1 Backpropagation through Truncated Signatures

Backpropagation is implemented efficiently by using the time-reversed path to deconstruct the signature. We refer the reader to [41, Section 4.9] for details of the algorithm. Our optimisations of the backpropagation are largely the same as for the forward pass of the signature. A slight modification is made to the original algorithm presented in [41], whereby Horner's algorithm is used to deconstruct the signature.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

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 claims made about the efficiency of the library accurately reflect the results presented in the paper (Section 2.4).
   Guidelines:
   - The answer NA 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 NA 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: Concrete experiments integrating the library into deep learning pipelines are beyond the scope of this paper and left for future work, as discussed in the conclusion of this work.

Guidelines:

- The answer NA 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: [NA]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer NA 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: All parameters for results concerning runtime are specified, and code reproducing the results is available in the public repository of the library.

   Guidelines:

   - The answer NA 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: Code reproducing the results is available in the public repository of the library.
   Guidelines:
   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?
   Answer: [Yes]

Justification: The paper does not involve training or testing of models. All parameters for results concerning runtime are specified, and code reproducing the results is available in the public repository of the library.

Guidelines:

- The answer NA 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: [No]

Justification: Error bars and statistical significance tests are not suitable for the results of the paper.

Guidelines:

- The answer NA 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: The exact processor type and available RAM is specified for the results.

Guidelines:

- The answer NA 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: The authors have reviewed and conform with the NeurIPS Code of Ethics.

Guidelines:

- The answer NA 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: [NA]

    Justification: There is no negative societal impact to the work. Positive societal impacts are discussed in the introduction, regarding the efficient training of machine learning models.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks, as it does not involve any data or models.

    Guidelines:

    - The answer NA 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: [NA]

Justification: The paper does not use existing assets.

Guidelines:

- The answer NA 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: [Yes]

    Justification: The library is documented and a URL to the documentation is provided in the introduction.

    Guidelines:

    - The answer NA 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: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA 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: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

- The answer NA 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 rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components

    Guidelines:

    - The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.